AI Enable car parking using OpenCV

Project Development Phase

Date	17 November 2023
Team Id	Team-592467
Project Name	AI Enable car parking using OpenCV

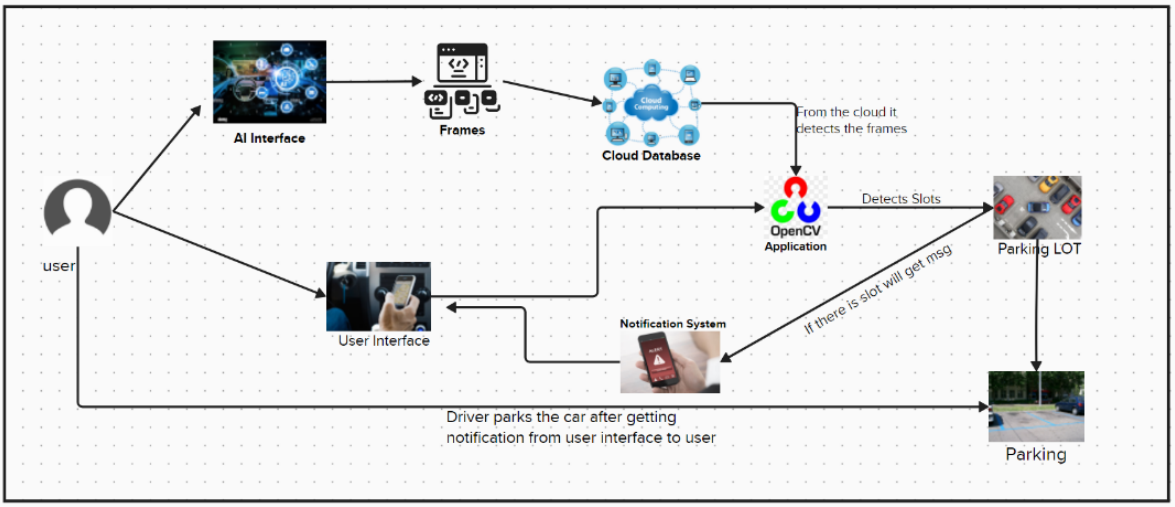
Car parking is a common problem faced by drivers in busy urban areas. For example, imagine you are driving to a shopping mall during peak hours. As you approach the mall, you notice that the parking lot is full, and several other cars are circling around looking for available spots.

You join the queue of cars, hoping to find an available spot soon. However, as time passes, you realize that the parking lot is overcrowded, and it's becoming increasingly difficult to find a spot. You start to feel frustrated and anxious, knowing that you might be late for your appointment or miss out on a great shopping opportunity.

AI-enabled car parking using OpenCV is a computer vision-based project that aims to automate the parking process. The project involves developing an intelligent system that can identify empty parking spaces and it gives the count of available parking spots.

The system uses a camera and OpenCV (Open Source Computer Vision) library to capture live video footage of the parking lot.

Architecture:



Pre-requisites:

1. **To complete this project you must have the following software versions and packages.**

- PyCharm (Download: https://www.jetbrains.com/pycharm/)
- Python 3.7.0 (Download: https://www.python.org/downloads/release/python-370/)
- Numpy
- cvzone

2. **To make a responsive python script you must require the following packages.**

Flask:

- Web framework used for building web applications.
- Flask Basics: Click here

If you are using anaconda navigator, follow below steps to download required packages:

- Open anaconda prompt.
- Type "pip install opencv-python" and click enter.
- Type "pip install cvzone" and click enter.
- Type "pip install Flask" and click enter.

If you are using Pycharm IDE, you can install the packages through the command prompt and follow the same syntax as above.

Project objectives:

By the end of this project you will:

- Know fundamental concepts and techniques of computer vision (openCV).
- Gain a broad understanding of image thresholding.

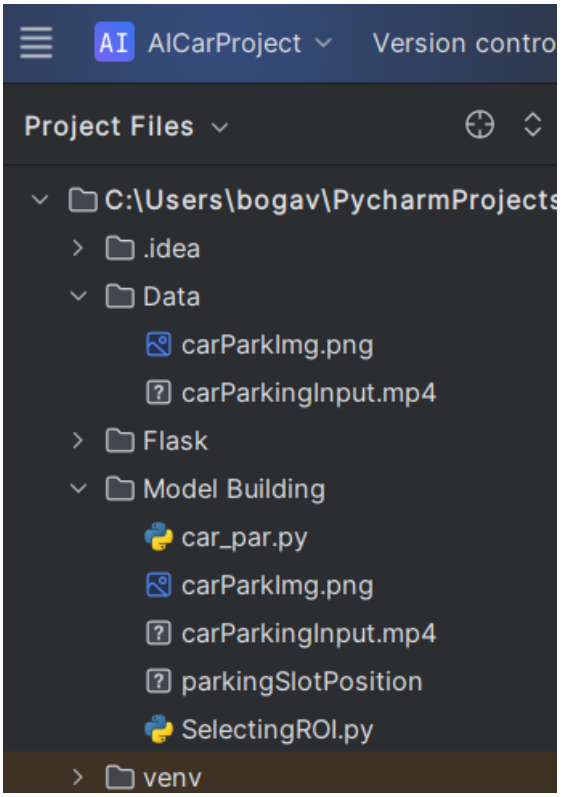
Project Flow:

- Data Collection
    - Download the dataset
- ROI (Region of interest)
    - Create python file
    - Import required libraries
    - Define ROI width and height
    - Select and deselect ROI
    - Denote ROI with BBOX

- Video Processing and object detection
    - Import required libraries
    - Reading input and loading ROI file
    - Checking for parking space
    - Looping the video
    - Frame processing and empty parking slot counters
- IBM Database Connection
    - Create IBM DB2 service and table
- Application building
    - Build HTML
    - Build python script for Flask

Project Structure:

Create a project folder which contains files as shown below:



- The Dataset folder contains the video and image file
- For building a Flask Application we needs HTML pages stored in the **templates** folder,CSS for styling the pages stored in the static folder and a python script **app.py** for server side scripting
- Model Building folder consists of car.py & selectingROI.py python script

Milestone – 1: Data Collection

Activity 1: Download the dataset

Click the below link to download dataset for AI Enabled Car Parking using OpenCV. Click here.

Milestone – 2: ROI (Region of interest)

ROI stands for Region of Interest, which refers to a specific rectangular portion of an image or video frame that is used for processing or analysis.

Activity 1: Create a python file

Create a python file in the directory and name it as 'selectingROI.py'. This python file is used for creating ROI and deleting ROI.

Activity 2: Importing the required packages

- **Opencv:** OpenCV is a great tool for image processing and performing computer vision tasks. It is an open-source library that can be used to perform tasks like face detection, objection tracking, landmark detection, and much more. It supports multiple languages including python, java C++.
- **Pickle:** The pickle library in Python is used for serialization and de-serialization of Python objects. Serialization is the process of converting a Python object into a stream of bytes that can be stored in a file or sent over a network. De-serialization is the process of converting the serialized data back into a Python object.

```
import cv2
import pickle
```

Activity 3: Define ROI width and height

- Calculating the ROI width and height (manually width and height are calculated and given as 107 & 48).
- An empty file parkingSlotPosition is created to save all the ROI values. Try and except combo is used.
- In Python, try and except are used for error handling, to catch and handle exceptions that may occur during program execution. The try block is used to enclose the code that may raise an exception, and the except block is used to define what should happen if an exception is raised.

```
width, height = 107, 48

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

Activity 4: Select and deselect ROI

- A function is defined as mouseClick. As parameters we are passing events (mouse action), x (ROI starting point), y (ROI ending point), flags (Boolean flag) and params (other parameters).
- In 1st if condition: After left click from the mouse, the starting and ending points will be added to posList by append method.
- In 2nd if condition: If ROI is selected in unwanted region. Then we can remove that unwanted ROI by right click from the mouse.
- Python object are converted into a stream of bytes and stored in parkingSlotPosition file.

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

Activity 5: Denote ROI with BBOX

- Reading the image with imread() method from cv2.
- All ROIs are saved in posList (refer activity 4).
- The rectangle is created as BBOX with the starting value (x) and ending value (y) from posList by rectangle() method. As the parameters give image source, starting value, ending value, (starting value + width, ending value + height), (color) and thickness.
- For displaying the image imshow() method is used.
- setMouseCallback() function is used to perform some action on listening the event which we have defined in activity 4.

```
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( winname: "Image", img)
    cv2.setMouseCallback("Image", mouseClick)
    cv2.waitKey(1)
```

Milestone - 3: Video processing and Object detection

Create a new python file to perform video processing, object detection and counters.

Activity 1: Import the required packages

Importing the required libraries to new python file.

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

Activity 2: Reading input and loading ROI file

- VideoCapture() method from cv2 is used to capture the video input. Download the input video from milestone 1.
- Load the parkingSlotPosition file by load() method from pickle library. The parkingSlotPosition file is created from milestone 2. The ROI values are presented in the parkingSlotPosition file.
- Define width and height which we have used in milestone 2.

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

Activity 3: Trackbar Initilization

```
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)
```

- empty: A placeholder function for the trackbar callback.
- Three trackbars (Val1, Val2, and Val3) are created to adjust parameters interactively.

Activity 4: Checking for parking space

```
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)), org: (x, y + h - 6), cv2.FONT_HERSHEY_PLAIN, fontScale: 1,
                    color, thickness: 2)

    cvzone.putTextRect(img, text: f'Free: {spaces}/{len(posList)}', pos: (50, 60), thickness=3, offset=20,
                       colorR=(0, 200, 0))
```

- The function is defined with the name as carParkingSpace. As a parameter image has to be passed.
- Taking the position from position list and saved to variable x and y.
- Cropping the image based on ROI (x and y value).
- To count the pixels from ROI, countNonZero() method is used from cv2. The BBOX (rectangle) is drawn in green color if the pixcel count is lesser than 900 else it is drawn in red color.
- The count of green color is displayed in the frame by putTextRect() method from cvzone library.

Activity 5: Looping the video

- Current frame position is compared with total number of frames. If it reaches the maximum frame, again the frame is set to zero. So, continuously it'll loop the frame. ● cv2.CAP_PROP_POS_FRAMES = Current frame
- cv2.CAP_PROP_FRAME_COUNT = Total no. of frame

```
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, value: 0)
```

Activity 6: Frame Processing and empty parking slot counters

- The captured video has to be read frame by frame. The read() method is used to read the frames.
- Opencv reads the frame in BGR (Blue Green Red).
- Converting BGR frame to gray scale image. And the gray scale image is blurred with GaussiamBlur() method from cv2.
- The adaptiveThreshold() method is used to apply threshold to the blurred image. And again blur is applied to the image. Click here to understand about adaptiveThreshold().
- The dilate() method is used to computing the minimum pixel value by overlapping the kernel over the input image. The blurred image after thresholding and kernel are passed as the parameters.
- The checkParkingSpace function is called with dilated image. As per the activity 3 we will get the count of empty parking slot.
- Display the frames in form of video. The frame will wait for 10 seconds and it'll go to next frame.

```
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, value: 0)
    # img = cv2.imread('img.png')
    imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    imgBlur = cv2.GaussianBlur(imgGray, ksize: (3, 3), sigmaX: 1)
    # ret, imgThres = cv2.threshold(imgBlur, 150, 255, cv2.THRESH_BINARY)

    val1 = cv2.getTrackbarPos( trackbarname: "Val1", winname: "Vals")
    val2 = cv2.getTrackbarPos( trackbarname: "Val2", winname: "Vals")
    val3 = cv2.getTrackbarPos( trackbarname: "Val3", winname: "Vals")
    if val1 % 2 == 0: val1 += 1
    if val3 % 2 == 0: val3 += 1
    imgThres = cv2.adaptiveThreshold(imgBlur, maxValue: 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                     cv2.THRESH_BINARY_INV, val1, val2)
    imgThres = cv2.medianBlur(imgThres, val3)
    kernel = np.ones( shape: (3, 3), np.uint8)
    imgThres = cv2.dilate(imgThres, kernel, iterations=1)

    checkSpaces()
    # Display Output

    cv2.imshow( winname: "Image", img)
    # cv2.imshow("ImageGray", imgThres)
    # cv2.imshow("ImageBlur", imgBlur)
    key = cv2.waitKey(1)
    if key == ord('r'):
        pass
```

Milestone - 4: Application building

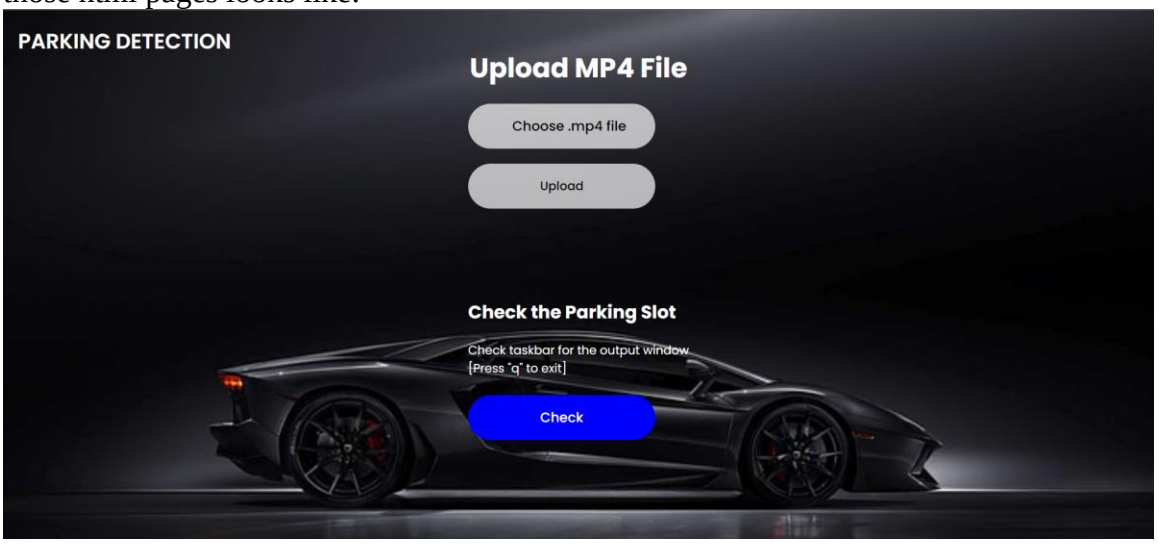
In this section, we will be building a web application that is integrated into the model we built. A UI is provided for the uses where he/she has to navigate to detect button. Then the video will be showcased on the UI.

This section has the following tasks

- Building HTML Pages
- Building server-side script

Activity1: Building Html Pages:

For this project we have created HTML files and saved them in the templates folder. Let's see how those html pages looks like:



Activity 2: Build Python code:

- Import the libraries

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

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


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

- Importing the flask module into the project is mandatory. An object of the Flask class is our WSGI application. Flask constructor takes the name of the current module (__name__) as an argument

- Here we will be using the declared constructor to route to the HTML page that we have created earlier. In the above example, the '/' URL is bound with the index.html function. Hence, when the home page of the web server is opened in the browser, the HTML page will be rendered.

- Create a decorator to route to '/predict_live'. Go to the milestone 3 and copy the entire code and paste it inside the function liv_pred.

- Main Function: Used to run the current module.

```
if __name__ == "__main__":
    app.run(debug=True)
```

Activity 3: Run the application

- Open the pycharm from the start menu
- Navigate to the folder where your python script is.
- Now type the "python app.py" command
- Navigate to the localhost where you can view your web page.
- Click on the predict button from the top right corner, enter the inputs, click on the submit button, and see the result/prediction on the web.

```
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 608-656-177
```

Output: