AI Enable Car Parking Using OpenCV

Participants Names :- Pragada Paul (20HQ1A0423)

Barla Uma Mahesh (20HQ1A0402)

Kilaparthi Reshma (20HQ1A0415)

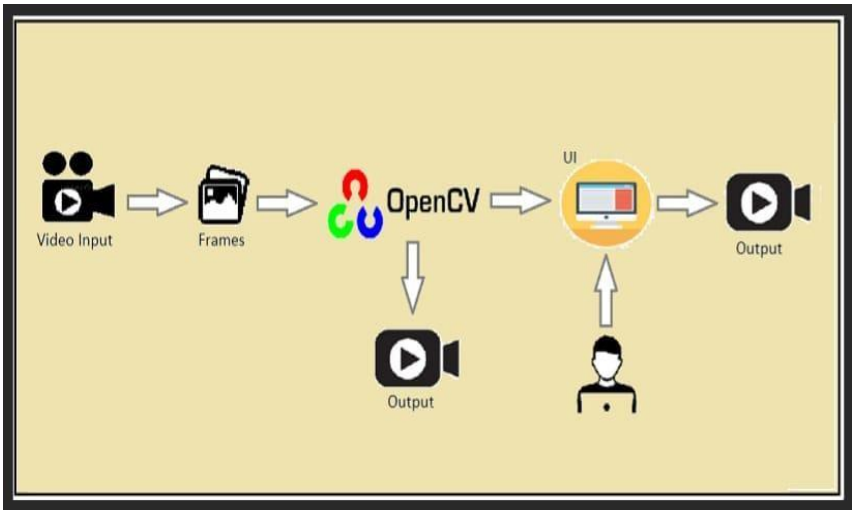
Kandi Lavanya (20HQ1A0414)

College Name :- Avanthi's Research and Technological Academy , Bhogapuram

ABSTRACT:

The problem of finding an appropriate parking space is a challenging one, particularly in large cities. With the increase in car ownership, parking spaces have become scarce. The growing demand for these spots coupled with limited availability has led to imbalances between supply and demand. A lack of adequate parking management systems has resulted in many streets being littered with illegally parked cars. A scalable, reliable, and efficient parking management system is needed to combat this problem. Deep learning-based computer vision techniques have emerged as promising solutions for such problems. These technologies have had a huge impact on the field of image recognition and processing.

A densely packed city center can be an unbearable place to park your car. Finding parking spaces can prove frustrating if you're not careful. Automatic smart parking systems promise to ease the burden of finding a spot in busy areas. To help drivers find a parking spot, we have developed a vision-based smartparking framework. First, we divided the parking lot into blocks and categorized each block to determine whether it was occupied or empty. Then we sent information about the availability of free or reserved parking to motorists on their smartphones. Our system demonstrates superior performance compared to commercially available solutions because it offers higher accuracy.



INTRODUCTION :

Most parking lots 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 The process of finding a free parking space can take a lot of time and involve driving around in circles. These days, parking spots are often occupied so badly that they're almost unusable. Poorly managed parking areas lead to inefficient utilization of the parking spaces. This causes a lot of traffic jams near the parking areas.

We propose a new method to improve the efficiency of parking lots by counting how much space is left in each parking zone and displaying that information to drivers via a smartphone app. We employ a camera to photograph the parking lot and use image processing approaches to determine if any vehicles are parked in each section. Whenever a vehicle moves into or out of a particular parking zone, the status of the whole lot changes. The growing demand for these spots coupled with limited availability has led to imbalances between supply and demand. A lack of adequate parking management systems has resulted in many streets being littered with illegally parked cars. A scalable, reliable, and efficient parking management system is needed to combat this problem. Deep learning-based computer vision techniques have emerged as promising solutions for such problems. These technologies have had a huge impact on the field of image recognition and processing.

Contents :

1. INSTALLATION

To begin, make sure you have the necessary software and packages installed. We recommend using PyCharm as the Integrated Development Environment (IDE) and Python 3.7.0 as the programming language. The following packages are required:

Pre-Requisites :

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

Required Packages:

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

2. DATA COLLECTION

In this section, we'll cover the steps for data collection, which includes downloading the dataset and defining the Region of Interest (ROI) for parking spot detection.

Downloading the Dataset :

"Click the below link to download the dataset for AI-Enabled Car Parking using OpenCV"
Link- https://drive.google.com/drive/folders/13vNq4XZLV15uxxXrVkj33Jr7VflrhKWr?usp=share_link

Download the necessary dataset that contains video and image files. The dataset should include various scenarios of parking lots with both empty and occupied parking spaces.

Creating ROI (Region of Interest) :

Develop a Python script to create and manage ROIs for the parking spots. The script will help in marking the regions on the video footage where parking spots are located.

- # Python script for creating and saving ROIs
- import cv2
- import pickle
  # Load the video and define the ROI points manually
- # Define the ROI coordinates (x, y, width, height) for each parking spot
- # Save the ROI data into a pickle file for future use

Selecting and Deselecting ROI :

Implement mouse event handlers to enable users to select and deselect ROIs on the video frame. This allows flexibility in defining parking spaces as needed.

```
# Python script for selecting
and deselecting ROIs import
cv2
import pickle

# Define a function to handle mouse click
events
# Add the selected/deselected ROI
coordinates to the list
# Save the updated ROI data into the
pickle file
```

Denoting ROI with BBOX :

Mark the selected ROIs using bounding boxes (BBOX) on the video frame for visualization purposes.

- **Code Snippet:**

```
# Python script for denoting ROIs with
BBOX
import cv2
import pickle
# Load the video and the ROI data from
the pickle file
# Draw BBOX for each ROI on the video
frame
# Display the video with BBOX to
visualize the ROIs
```

3. VIDEO DETECTION AND COLLECTION

This section focuses on processing the captured video frames, applying image processing techniques, and detecting parking spaces using OpenCV and the previously defined ROIs.

Reading Input and Loading the ROI File :

Capture the input video using the OpenCV VideoCapture() method and load the previously defined ROI file using the pickle library.

Code Snippet:

```
# Python script for video processing
and object detection import cv2
import pickle

# Load the video and the ROI data from the
pickle file
# Start processing the video frame by frame
```

Checking for Parking Space :

Implement a function to count the empty parking slots by processing each frame and analyzing the ROI's pixel values.

Code Snippet:

```
# Python script for checking
parking space import cv2
import pickle

# Load the video and the ROI data from the
pickle file
```

```
# Implement a function to count empty parking slots in each frame
# Display the number of empty parking slots on each frame
```

Looping the Video :

Loop through the video frames to continuously process the parking data and update the results.

Code Snippet:

```
# Python script for looping the video
import cv2
import pickle

# Load the video and the ROI data from the pickle file
# Loop through the video frames
# Apply parking space detection function to each frame
# Display the video with real-time parking slot availability
```

Frame Processing and Empty Parking Slot Counters :

Read the video frames and apply various image processing techniques such as grayscale conversion, blurring, thresholding, and dilation to prepare the frames for parking slot detection. Use the checkParkingSpace function to calculate the number of empty parking slots and display them on the frame.

Code Snippet:

 # Python script for frame processing and empty parking slot counters import cv2 import pickle

 # Load the video and the ROI data from the pickle file
 # Loop through the video frames
 # Apply image processing techniques to prepare the frames
 # Calculate the number of empty parking slots and display them on the frame # Display the video with real-time parking slot availability

CODES FOR ROI and VIDEO DETECTION:

```
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 = []
```

```python
def mouseClick(events, x, y, flags, params):
    # Adding ROI values to posList
    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 posList values to parkingSlotPosition file
    with open('parkingSlotPosition', 'wb') as f:
        pickle.dump(posList, f)
```

```python
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:
```

```python
while True:
    img = cv2.imread('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)
```

```python
    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
```

3. BUILDING HTML PAGES

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

Code Snippet:

```
<!-- HTML page for parking predictions -->
<!DOCTYPE html>
<html>
<head>
   <title>Parking Predictions</title>
</head>
<body>
   <h1>AI-Enabled Car Parking System</h1>
   <label for="parking_lot">Select Parking Lot:</label>
   <select id="parking_lot">
      <!-- Populate the dropdown with parking lot options -->
   </select>
   <br>
   <label for="timestamp">Enter Timestamp:</label>
   <input type="text" id="timestamp" placeholder="Enter timestamp..."> <br>
   <button onclick="predict()">Predict</button>
   <br>
   <div id="prediction_result">
      <!-- Display the prediction result here -->
```

```
    </div>
    <script>
        // JavaScript function to handle prediction and display
the result        function predict() {
            // Get the selected parking lot and timestamp
            // Call the API with the selected values
            // Display the prediction result on the page
        }
    </script>
</body>
</html>
```

Building Python Code :

Integrate the Flask web framework with the previously built model. Implement functions to route user inputs, process them, and showcase the prediction results on the web page.

Save the folder as app.py

```
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__)
```

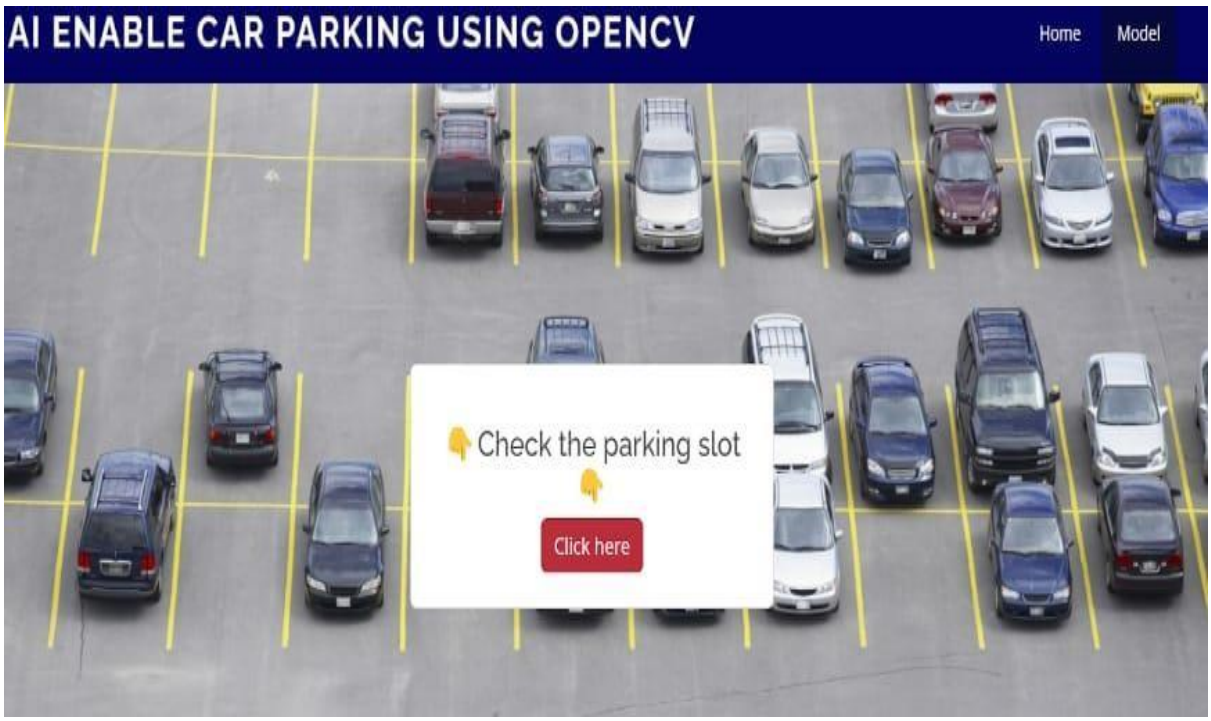
```python
@app.route('/')
def project():
    return render_template('index.html')
@app.route('/hero')
def home():
    return render_template('index.html')
@app.route('/model')
def model():
    return render_template('model.html')
```

```python
@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(1) & 0xFF == ord('q'):
    break
```

OUTPUTS



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.