OUTDOOR CAR PARK SLOT DETECTION SYSTEM USING COMPUTER VISION

NIPUN ANJANA

05-03-2024

INSTRUCTIONS

The increasing number of vehicles and limited parking spaces make finding empty spots time-consuming. To address this, car park slot detection systems powered by computer vision are being developed. These systems use cameras and algorithms to automatically identify and track available parking spaces in real-time. Implementing these systems in car parks can improve efficiency, reduce traffic congestion, and provide valuable data for parking management optimization.

Before accessing the code, you should know these things.

- There have the API codes inside the main.py and test.py. Before you run these codes without API, you should comment those lines. Here are the parts.

```python
import cv2
import pandas as pd
import numpy as np
from ultralytics import YOLO
import time
import socketio

#API part start.
authToken = "your_secret_token"  # Replace with your actual token
sio = socketio.Client()

@sio.event
def connect():
    print("Connected to server")

@sio.event
def disconnect():
    print("Disconnected from server")

@sio.event
def parkingStatus(data):
    # Handle parking status updates (replace with your logic)
    print("Parking space ID:", data["spaceId"], "Availability:", data["availability"])

# Function to simulate form submission (since Python doesn't have browser events)
def update_slots(spaceId, availability ):
    sio.emit("parkingStatus", {"spaceId": spaceId, "availability": availability})

sio.connect("http://localhost:5000", auth={"token": authToken})
#API part end.

#Load YOLO model
model=YOLO('yolov8s.pt')
```

```python
    print(f"Slot 12 = {isAvailable12}")

# pass and update data with database
    update_slots("S1",isAvailable1)
    update_slots("S2",isAvailable2)
    update_slots("S3",isAvailable3)
    update_slots("S4",isAvailable4)
    update_slots("S5",isAvailable5)
    update_slots("S6",isAvailable6)
    update_slots("S7",isAvailable7)
    update_slots("S8",isAvailable8)
    update_slots("S9",isAvailable9)
    update_slots("S10",isAvailable10)
    update_slots("S11",isAvailable11)
    update_slots("S12",isAvailable12)

# Draw parking areas with different colors based on occupancy
    if a1==1:
        cv2.polylines(frame,[np.array(area1,np.int32)],True,(
```

```python
url = 'http://localhost:3000/parkingSpaces'
#url = 'http://localhost:5000/parkingSpaces'

cap.release()
cv2.destroyAllWindows()
```

- In main.py, works with IP camera feeds. Before you run main.py, you should replace the correct IP addresses.

```
41  cv2.namedWindow('RGB')
42  cv2.setMouseCallback('RGB', RGB)
43
44  # set ip camera addresses.
45  camera_address1 = 'http://192.168.8.100:8080/video'  # Camera 1 for slots 1-3
46  camera_address2 = 'http://192.168.8.100:8080/video'  # Camera 2 for slots 4-6
47
48  #Open video capture and read frames
49  cap1 = cv2.VideoCapture(camera_address1)
50  cap2 = cv2.VideoCapture(camera_address2)
51
```

- Before you draw the spot location manually, you should freeze the frame. It's easy to draw the lines. Change this 0 and 1 as you want.
    - 0 = freeze the frame. You can play the video using arrow keys.
    - 1 = play the video or IP camera feed continuously.

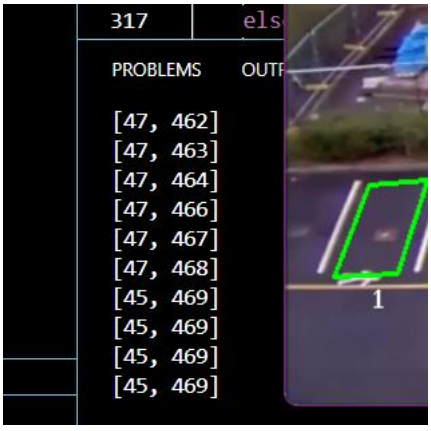
```
304     #time.sleep(3)
305     if cv2.waitKey(0) & 0xFF == 27:
306         break      # Exit if 'ESC' key is
307
```

Freeze the frame (0)

```
304     #time.sleep(3)
305     if cv2.waitKey(1) & 0xFF == 27:
306         break      # Exit if 'ESC' key is
```

Play the frame continuously (1)

- When you in the freeze frame, you can see the mouse pointer location in the terminal. Using this, you can mark the spot location manually.

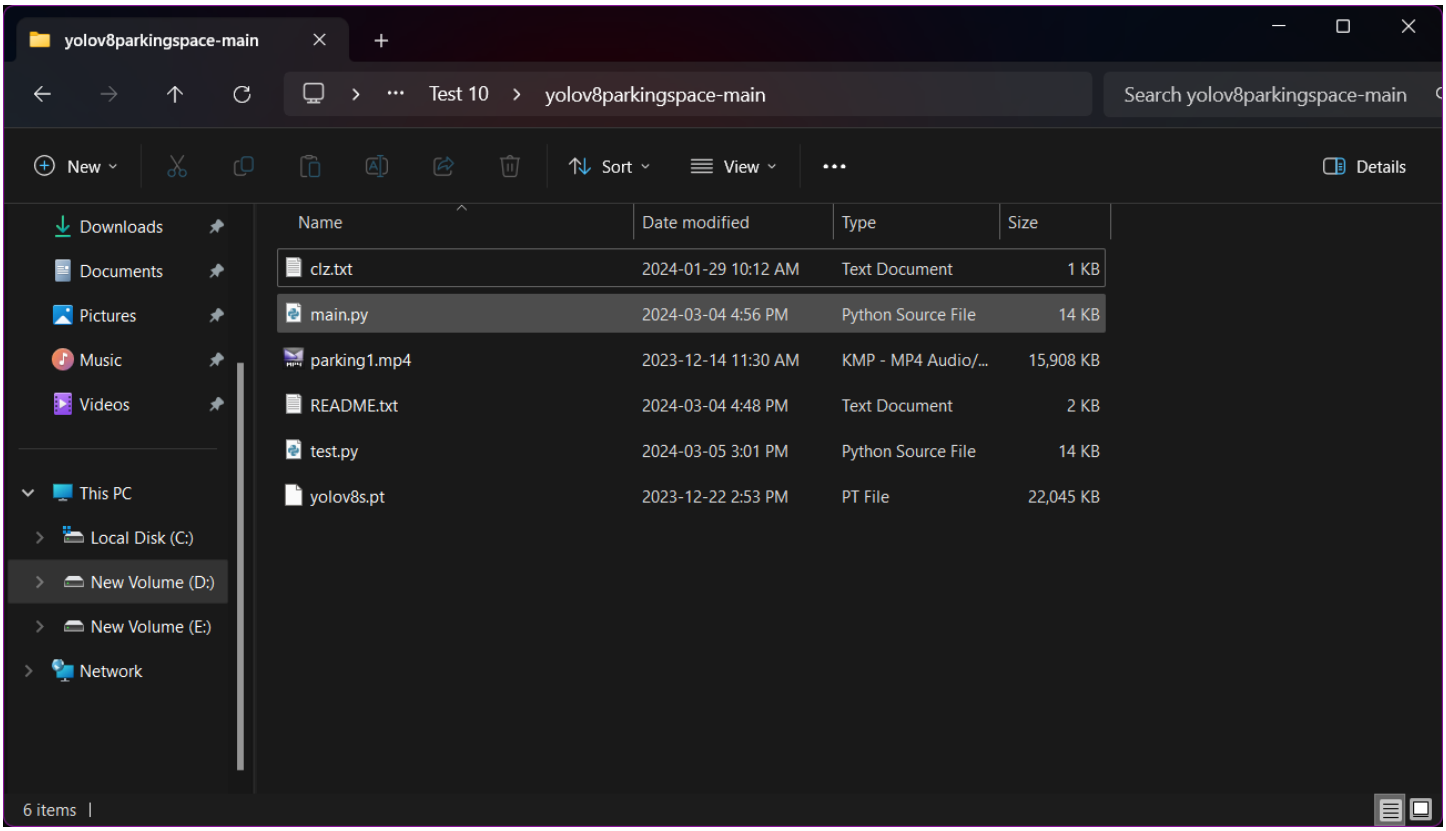


Show the mouse pointer location in the terminal when run the code,

```
63
64  # Camera 1 (slots 1-3)
65  area1_1 = [(36,537),(28,629),(77,635),(84,534)]
66  area1_2 = [(79,534), (77,635), (123, 632), (124, 531)]
67  area1_3 = [(117, 531), (117, 630), (169,627), (166,528)]
68
69  # Camera 2 (slots 4-6)
70  area2_4 = [(1118,493),(1162,581),(1209,569),(1168,481)]
71  area2_5 = [(1168,481),(1209,569),(1256,554), (1210,475)]
72  area2_6 = [(1210,475),(1256,554),(1294,541),(1242,466)]
73
```

Using that mouse pointer location, you can change those values to draw the vehicle spots.

About the folder file explanation. (yolov8parkingspace-main)



- Clz.txt – In this file, there are the names of the objects around us. This file call in the code to get car as an object to identify.
- Main.py – this is the main code. In here, used 2 IP camera feed as an input. These 2 video feed finally show in the combined frame as an output. At that time these 2 video feed define the 2 frames. When run the code, object detection system adds each frames individually. After combined the frames, add again object detection that frame.
- Parking1.mp4 – this is the demo video for testing the system.
- Readme.txt – this file has how to set up and install libraries and etc.
- Test.py – this is the testing file. In here, when run the code that parking.mp4 video play with object detection system part. Before modify the main.py, we can test modification using this file and video.
- Yolov8s.pt – this is the yolov8 file. It's help to detect objects in your pc locally when you run the code.

Code explanation

Test.py:

Import libraries:

- cv2: OpenCV library for computer vision tasks like image processing and object detection.
- pandas: Used for data manipulation and analysis (specifically converting data into a DataFrame).
- numpy: Used for numerical computations and array manipulations.
- from ultralytics import YOLO: Imports the YOLO object detection model for identifying cars in the video frames.
- time: Provides functions for measuring elapsed time.
- socketio: Enables real-time communication between the Python script and a server using web sockets.

API setup:

- Defines authToken (replace with your actual token) which is likely used for authentication with the server.
- Establishes a connection to the server at http://localhost:5000 using the socketio library.
- Defines functions to handle different events:
    - connect(): Prints a message when the connection is established.
    - disconnect(): Prints a message when the connection is lost.
    - parkingStatus(data): Receives updates about parking space IDs and their availabilities from the server and prints them.
- Defines a function update_slots(spaceId, availability) to send information about parking space availability to the server using socketio.emit.

Load YOLO model and define functions:

- Loads the YOLO model (yolov8s.pt) for car detection.
- Defines a function RGB(event, x, y, flags, param) to handle mouse events on the video frame but is not used in the main loop.

Open video capture and define areas:

- Opens a video capture using cv2.VideoCapture(). You can replace the file path with a URL to stream from a camera.
- Reads the class names (likely "car" in this case) from a text file (clz.txt).
- Defines twelve 2D lists (area1 to area12) representing the corner coordinates of each parking space.

Main loop:

- The code enters an infinite loop that reads frames from the video.
- Each frame is resized to (1020, 500) pixels.
- The YOLO model is used to detect objects in the frame (results = model.predict(frame)).
- The bounding boxes and class labels for detected objects are extracted and stored in a pandas DataFrame (px).
- Twelve empty lists (list1 to list12) are created to store detected cars in each parking area.
- Iterates through each detection:
    - Extracts bounding box coordinates and class label.
    - If the class is "car", the car's center is calculated.
    - For each parking area, checks if the car's center falls within its defined polygon using cv2.pointPolygonTest().
    - If a car is found in a parking area, it is marked with a rectangle, circle, and text label on the frame, and its corresponding list (list1 to list12) is appended with the class name ("car").

Count available spaces and check individual space availability:

- Counts the number of cars in each parking area using the lengths of the respective lists (list1 to list12).
- Calculates the total number of occupied spaces (o) and available spaces (space).
- Checks the availability (occupied or not) of each individual parking space (isAvailable1 to isAvailable12) by checking the length of the corresponding list.

Update server and draw parking areas:

- Sends updates about the availability of each parking space (S1 to S12) to the server using the update_slots function.
- Iterates through each parking area:
    - If a car is parked, the area is drawn with a red border and its slot number is displayed.
    - If no car is parked, the area is drawn with a green border and its slot number is displayed.

Display frame and exit:

- Displays the frame with detections and parking areas using cv2.imshow().
- Waits for a key press. If the 'ESC' key is pressed, the loop breaks, and the program exits.
- Releases the video capture and destroys all OpenCV windows.

Main.py:

Imports: The code imports necessary libraries such as OpenCV (**cv2**), pandas (**pd**), numpy (**np**), ultralytics.YOLO for object detection, time for timing operations, and socketio for handling socket communication.

Socket.io Setup: It establishes a connection to a server using socket.io, with event handlers for connection (**connect**) and disconnection (**disconnect**). It also defines an event handler for receiving parking status updates (**parkingStatus**).

Function Definitions:

- **update_slots**: This function simulates form submission by emitting a **parkingStatus** event with the provided space ID and availability.
- **RGB**: This function is a callback for mouse events. It prints the coordinates of the mouse cursor when moved over the window named 'RGB'.

YOLO Model Loading: It loads the YOLO model for object detection.

Camera Setup: It defines the addresses of two IP cameras for video streaming.

Polygonal Area Definitions: It defines polygonal areas for each parking space using lists of coordinates.

Main Loop: It starts an infinite loop to process frames from the video streams, performs object detection, checks for cars in parking areas, and updates the parking status.

Object Detection and Processing: It detects cars in each frame using YOLO, checks if the detected cars are within defined parking areas, and updates the parking status based on car occupancy.

Drawing: It draws rectangles and circles on the combined frame to indicate detected cars and parking areas. It also updates the count of available spaces and displays it on the frame.

Display: It displays the combined frame with detections and parking areas.

Key Press Handling: It waits for a key press event and exits the loop if the 'ESC' key is pressed.

HTTP Request: There's commented-out code for sending HTTP requests to a specific URL.

Cleanup: It releases the video capture resources and closes all OpenCV windows.

Additional:

- In here I used visual studio as my IDE.
- Download the libraries. Run these commands in terminal to install. It's better use the virtual environment.
    - `pip install opencv-python`
    - `pip install pandas`
    - `pip install numpy`
    - `pip install ultralytics`
    - `pip install python-socketio`