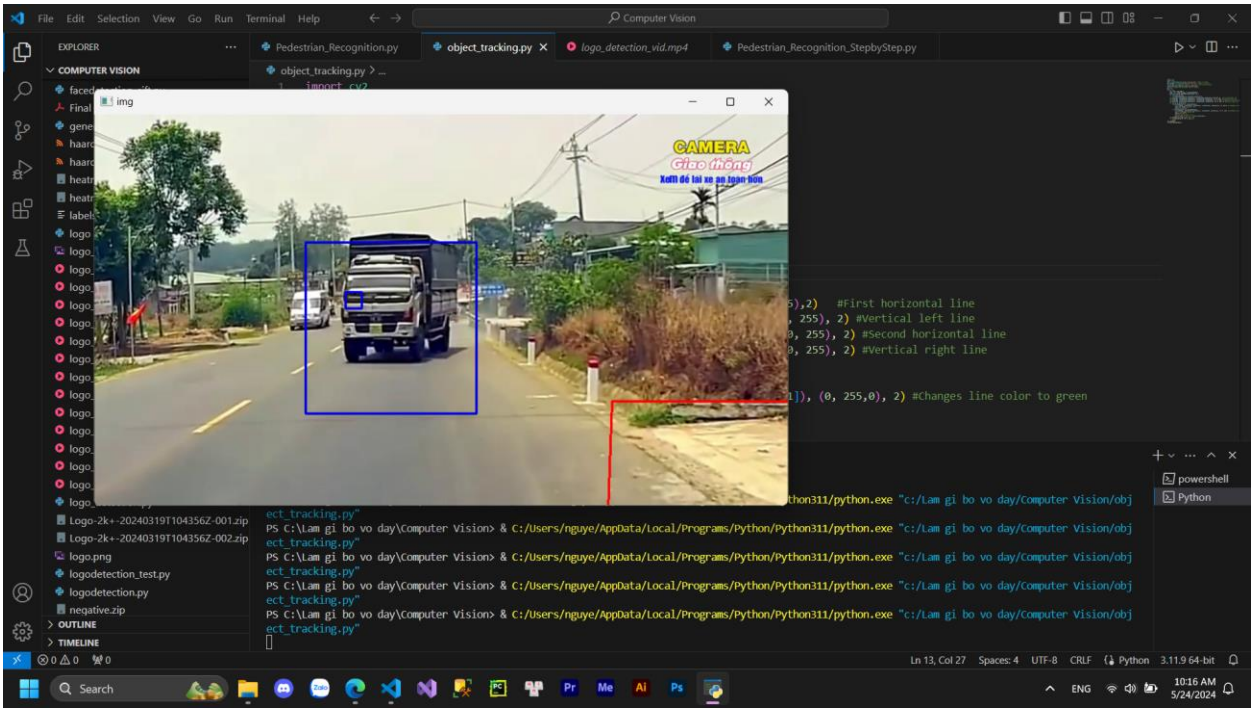
```
import cv2
import time
cap = cv2.VideoCapture('object_tracking.mp4')  #Path to footage
car_cascade = cv2.CascadeClassifier('cars.xml')  #Path to cars.xml
#Coordinates of polygon in frame::: [[x1,y1],[x2,y2],[x3,y3],[x4,y4]]
coord=[[637,352],[904,352],[631,512],[952,512]]
#Distance between two horizontal lines in (meter)
dist = 3
while True:
    ret, img = cap.read()
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    cars=car_cascade.detectMultiScale(gray,1.8,2)
    for (x,y,w,h) in cars:
        cv2.rectangle(img,(x,y),(x+w,y+h),(225,0,0),2)
    cv2.line(img,
(coord[0][0],coord[0][1]),(coord[1][0],coord[1][1]),(0,0,255),2)   #First
horizontal line
    cv2.line(img, (coord[0][0],coord[0][1]), (coord[2][0],coord[2][1]), (0, 0,
255), 2) #Vertical left line
    cv2.line(img, (coord[2][0],coord[2][1]), (coord[3][0], coord[3][1]), (0, 0,
255), 2) #Second horizontal line
    cv2.line(img, (coord[1][0],coord[1][1]), (coord[3][0], coord[3][1]), (0, 0,
255), 2) #Vertical right line
    for (x, y, w, h) in cars:
        if(x>=coord[0][0] and y==coord[0][1]):
            cv2.line(img, (coord[0][0], coord[0][1]), (coord[1][0], coord[1][1]),
(0, 255,0), 2) #Changes line color to green
            tim1= time.time() #Initial time
            print("Car Entered.")
        if (x>=coord[2][0] and y==coord[2][1]):
            cv2.line(img, (coord[2][0],coord[2][1]), (coord[3][0], coord[3][1]),
(0, 0, 255), 2) #Changes line color to green
            tim2 = time.time() #Final time
            print("Car Left.")
            #We know that distance is 3m
            print("Speed in (m/s) is:", dist/((tim2-tim1)))
    cv2.imshow('img',img) #Shows the frame
    if cv2.waitKey(20) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```




```python
import cv2
import numpy as np
from collections import deque

# Open the video capture
cap = cv2.VideoCapture('object_tracking.mp4')

# Create the background subtractor object
fgbg = cv2.createBackgroundSubtractorMOG2(history=500, varThreshold=100,
detectShadows=True)

# Define the distance conversion factor (e.g., pixels to meters)
# This value needs to be measured/calibrated for your specific video
pixels_per_meter = 20.0  # Example value, adjust according to your setup

# Define the deque to store the positions of the detected cars
trackers = []

# Define the deque to store the speeds of the detected cars
speeds = deque(maxlen=50)

# Define the time interval between frames in seconds (video frame rate)
frame_rate = 30.0  # Example value, adjust according to your video frame rate
time_interval = 1.0 / frame_rate
```

```
while True:
    ret, frame = cap.read()
    if not ret:
        print("[INFO] End of video file reached.")
        break

    # Apply the background subtractor to get the foreground mask
    fgmask = fgbg.apply(frame)

    # Perform morphological operations to remove noise and fill gaps
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5))
    fgmask = cv2.morphologyEx(fgmask, cv2.MORPH_OPEN, kernel)
    fgmask = cv2.morphologyEx(fgmask, cv2.MORPH_CLOSE, kernel)

    # Find contours in the foreground mask
    contours, _ = cv2.findContours(fgmask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    # Current positions of detected cars
    current_positions = []

    # Loop over the contours
    for contour in contours:
        # Only consider contours with a significant area to avoid noise
        if cv2.contourArea(contour) > 500:
            # Get the bounding box for each contour
            x, y, w, h = cv2.boundingRect(contour)
            # Draw the bounding box on the frame
            cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
            # Save the center point of the bounding box
            current_positions.append((x + w // 2, y + h // 2))

    # Track the cars
    if len(trackers) == 0:
        trackers = [deque(maxlen=2) for _ in range(len(current_positions))]

    # Update the trackers with the current positions
    for i, pos in enumerate(current_positions):
        if i < len(trackers):
            trackers[i].append(pos)
        else:
            trackers.append(deque(maxlen=2))
            trackers[i].append(pos)
```

```
    # Calculate the speed for each tracker
    for i, tracker in enumerate(trackers):
        if len(tracker) == 2:
            dx = tracker[1][0] - tracker[0][0]
            dy = tracker[1][1] - tracker[0][1]
            distance_pixels = np.sqrt(dx**2 + dy**2)
            distance_meters = distance_pixels / pixels_per_meter
            speed_mps = distance_meters / time_interval
            speed_kmph = speed_mps * 3.6
            speeds.append(speed_kmph)
            # Draw the speed on the frame
            cv2.putText(frame, f"{speed_kmph:.2f} km/h", (tracker[1][0],
tracker[1][1] - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 255), 2)

    # Display the result
    cv2.imshow('Car Speed Detection', frame)

    # Exit if 'q' is pressed
    if cv2.waitKey(30) & 0xFF == ord('q'):
        break

# Release the video capture and close windows
cap.release()
cv2.destroyAllWindows()
```