Code With Explanation in HASHTAGS

```
import os  # Import os for creating directories
import cv2  # Import OpenCV for video processing
import time  # Import time for capturing timestamps
import pandas as pd  # Import pandas for data manipulation

# Initialize dictionaries and lists for tracking vehicles
down = {}
up = {}
counter_down = []
counter_up = []

# Define the Y-coordinates for the red and blue lines
red_line_y = 198
blue_line_y = 268
offset = 6  # Offset for detecting when a vehicle crosses a line

# Create a folder to save detected frames, if it doesn't already exist
if not os.path.exists('detected_frames'):
        os.makedirs('detected_frames')

# Set up the video writer to save the output video in AVI format
fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter('output.avi', fourcc, 20.0, (1020, 500))

# Initialize a variable to count frames
count = 0

# Start processing video frames in a loop
while True:
        ret, frame = cap.read()  # Read a frame from the video
        if not ret:  # If the video has ended, break the loop
        break
        count += 1  # Increment the frame count

        # Resize the frame to a standard size
        frame = cv2.resize(frame, (1020, 500))

        # Predict objects in the frame using the pre-trained model
        results = model.predict(frame)
        a = results[0].boxes.data  # Extract bounding box data
        a = a.detach().cpu().numpy()  # Convert data to a NumPy array
        px = pd.DataFrame(a).astype("float")  # Convert to a pandas DataFrame

        list = []  # Initialize a list to store detected vehicles
```

```
    # Iterate over each detected object
    for index, row in px.iterrows():
    x1 = int(row[0])
    y1 = int(row[1])
    x2 = int(row[2])
    y2 = int(row[3])
    d = int(row[5])
    c = class_list[d]  # Get the class of the detected object

    # If the detected object is a car, add it to the list
    if 'car' in c:
    list.append([x1, y1, x2, y2])

    # Update the tracker with the current list of vehicles
    bbox_id = tracker.update(list)

    # Process each tracked vehicle
    for bbox in bbox_id:
    x3, y3, x4, y4, id = bbox
    cx = int(x3 + x4) // 2  # Calculate the center X-coordinate of the bounding box
    cy = int(y3 + y4) // 2  # Calculate the center Y-coordinate of the bounding box

    # Check if the vehicle crosses the red line (going down)
    if red_line_y < (cy + offset) and red_line_y > (cy - offset):
    down[id] = time.time()  # Record the time when the vehicle crosses the red line

    # If the vehicle is going down and crosses the blue line
    if id in down:
    if blue_line_y < (cy + offset) and blue_line_y > (cy - offset):
        elapsed_time = time.time() - down[id]  # Calculate the time taken to cross
between the lines
        if counter_down.count(id) == 0:  # If the vehicle hasn't been counted yet
        counter_down.append(id)  # Count the vehicle
        distance = 10  # Distance between the lines in meters
        a_speed_ms = distance / elapsed_time  # Calculate speed in meters per
second
        a_speed_kh = a_speed_ms * 3.6  # Convert speed to kilometers per hour

        # Draw a circle at the center of the vehicle
        cv2.circle(frame, (cx, cy), 4, (0, 0, 255), -1)
        # Draw a bounding box around the vehicle
        cv2.rectangle(frame, (x3, y3), (x4, y4), (0, 255, 0), 2)
        # Display the vehicle ID
        cv2.putText(frame, str(id), (x3, y3), cv2.FONT_HERSHEY_COMPLEX, 0.6,
(255, 255, 255), 1)
        # Display the speed of the vehicle
        cv2.putText(frame, str(int(a_speed_kh)) + 'Km/h', (x4, y4),
cv2.FONT_HERSHEY_COMPLEX, 0.8, (0, 255, 255), 2)
```

```
    ##### Check if the vehicle crosses the blue line (going up) #####
    if blue_line_y < (cy + offset) and blue_line_y > (cy - offset):
    up[id] = time.time()  # Record the time when the vehicle crosses the blue line

    # If the vehicle is going up and crosses the red line
    if id in up:
    if red_line_y < (cy + offset) and red_line_y > (cy - offset):
        elapsed1_time = time.time() - up[id]  # Calculate the time taken to cross
between the lines
        if counter_up.count(id) == 0:  # If the vehicle hasn't been counted yet
        counter_up.append(id)  # Count the vehicle
        distance1 = 10  # Distance between the lines in meters
        a_speed_ms1 = distance1 / elapsed1_time  # Calculate speed in meters per
second
        a_speed_kh1 = a_speed_ms1 * 3.6  # Convert speed to kilometers per hour

        # Draw a circle at the center of the vehicle
        cv2.circle(frame, (cx, cy), 4, (0, 0, 255), -1)
        # Draw a bounding box around the vehicle
        cv2.rectangle(frame, (x3, y3), (x4, y4), (0, 255, 0), 2)
        # Display the vehicle ID
        cv2.putText(frame, str(id), (x3, y3), cv2.FONT_HERSHEY_COMPLEX, 0.6,
(255, 255, 255), 1)
        # Display the speed of the vehicle
        cv2.putText(frame, str(int(a_speed_kh1)) + 'Km/h', (x4, y4),
cv2.FONT_HERSHEY_COMPLEX, 0.8, (0, 255, 255), 2)

    # Define colors for drawing text and lines on the frame
    text_color = (0, 0, 0)  # Black color for text
    yellow_color = (0, 255, 255)  # Yellow color for background
    red_color = (0, 0, 255)  # Red color for lines
    blue_color = (255, 0, 0)  # Blue color for lines

    # Draw a yellow rectangle to display vehicle counts
    cv2.rectangle(frame, (0, 0), (250, 90), yellow_color, -1)

    # Draw the red line and label it
    cv2.line(frame, (172, 198), (774, 198), red_color, 2)
    cv2.putText(frame, 'Red Line', (172, 198), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
text_color, 1, cv2.LINE_AA)

    # Draw the blue line and label it
    cv2.line(frame, (8, 268), (927, 268), blue_color, 2)
    cv2.putText(frame, 'Blue Line', (8, 268), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
text_color, 1, cv2.LINE_AA)

    # Display the number of vehicles going down
```

```
        cv2.putText(frame, 'Going Down - ' + str(len(counter_down)), (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, text_color, 1, cv2.LINE_AA)
        # Display the number of vehicles going up
        cv2.putText(frame, 'Going Up - ' + str(len(counter_up)), (10, 60),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, text_color, 1, cv2.LINE_AA)

        # Save the current frame to the detected_frames folder
        frame_filename = f'detected_frames/frame_{count}.jpg'
        cv2.imwrite(frame_filename, frame)

        # Write the frame to the output video file
        out.write(frame)

        # Display the current frame in a window titled "frames"
        cv2.imshow("frames", frame)
        if cv2.waitKey(1) & 0xFF == 27:  # Exit the loop if the 'Esc' key is pressed
        break

# Release the video capture and writer objects
cap.release()
out.release()
# Close all OpenCV windows
cv2.destroyAllWindows()
```

Code With Explanation in HASHTAGS (With NO plate recorder)

```
import os  # Module for interacting with the operating system
import cv2  # OpenCV library for image processing
import time  # Module to work with time
import pandas as pd  # Pandas library for data manipulation
from ultralytics import YOLO  # YOLO model for object detection
from tracker import Tracker  # Custom tracker class for tracking objects
import easyocr  # EasyOCR for optical character recognition

# Load the YOLO model with a pretrained 'yolov8s.pt' model file
model = YOLO('yolov8s.pt')

# Define a list of classes that YOLO can detect, focusing on vehicles of interest
class_list = [
        'person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus', 'train', 'truck', 'boat',
        'traffic light', 'fire hydrant', 'stop sign', 'parking meter', 'bench', 'bird', 'cat',
        'dog', 'horse', 'sheep', 'cow', 'elephant', 'bear', 'zebra', 'giraffe', 'backpack',
        'umbrella', 'handbag', 'tie', 'suitcase', 'frisbee', 'skis', 'snowboard', 'sports ball',
        'kite', 'baseball bat', 'baseball glove', 'skateboard', 'surfboard', 'tennis racket',
        'bottle', 'wine glass', 'cup', 'fork', 'knife', 'spoon', 'bowl', 'banana', 'apple',
        'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog', 'pizza', 'donut', 'cake', 'chair',
        'couch', 'potted plant', 'bed', 'dining table', 'toilet', 'tv', 'laptop', 'mouse',
        'remote', 'keyboard', 'cell phone', 'microwave', 'oven', 'toaster', 'sink', 'refrigerator',
        'book', 'clock', 'vase', 'scissors', 'teddy bear', 'hair drier', 'toothbrush'
]

# Initialize the tracker to keep track of detected objects
tracker = Tracker()

# Initialize counters and dictionaries to track vehicles crossing lines
count = 0  # Frame count
down = {}  # Dictionary to track vehicles going down (crossing the red line)
up = {}  # Dictionary to track vehicles going up (crossing the blue line)
counter_down = []  # List to count vehicles going down
counter_up = []  # List to count vehicles going up

# Define the vertical positions (Y-coordinates) for the lines
red_line_y = 198
blue_line_y = 268
offset = 6  # Offset for detecting if a vehicle crosses a line

# Load the video for processing
cap = cv2.VideoCapture('/home/arya/Desktop/Vs
Code/websec/Final/Speed-detection-of-vehicles/highway.mp4')

# Create directories to store detected frames and license plates if they don't already exist
if not os.path.exists('detected_frames'):
        os.makedirs('detected_frames')
if not os.path.exists('license_plates'):
        os.makedirs('license_plates')

# Set up the video writer to save the output video in AVI format
fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter('output.avi', fourcc, 20.0, (1020, 500))

# Initialize EasyOCR reader for license plate recognition
reader = easyocr.Reader(['en'])

# Process the video frame by frame
while True:
        ret, frame = cap.read()  # Read a frame from the video
        if not ret:  # Exit the loop if no frame is captured (end of video)
        break

        count += 1  # Increment the frame counter
        frame = cv2.resize(frame, (1020, 500))  # Resize the frame to a standard size

        # Use the YOLO model to predict objects in the frame
        results = model.predict(frame)
```

```
    a = results[0].boxes.data  # Extract bounding box data from the model
    a = a.detach().cpu().numpy()  # Convert the data to a NumPy array
    px = pd.DataFrame(a).astype("float")  # Convert data to a pandas DataFrame
    list = []  # List to store coordinates of detected vehicles

    # Iterate over each detected object
    for index, row in px.iterrows():
    x1 = int(row[0])  # Top-left X-coordinate of the bounding box
    y1 = int(row[1])  # Top-left Y-coordinate of the bounding box
    x2 = int(row[2])  # Bottom-right X-coordinate of the bounding box
    y2 = int(row[3])  # Bottom-right Y-coordinate of the bounding box
    d = int(row[5])  # Class ID of the detected object
    c = class_list[d]  # Class name based on the class ID

    # If the detected object is a car, add its coordinates to the list
    if 'car' in c:
    list.append([x1, y1, x2, y2])

    # Update the tracker with the current list of vehicles
    bbox_id = tracker.update(list)

    # Process each tracked vehicle
    for bbox in bbox_id:
    x3, y3, x4, y4, id = bbox  # Bounding box coordinates and unique ID
    cx = int((x3 + x4) // 2)  # Center X-coordinate of the bounding box
    cy = int((y3 + y4) // 2)  # Center Y-coordinate of the bounding box

    # Check if the vehicle crosses the red line (going down)
    if red_line_y < (cy + offset) and red_line_y > (cy - offset):
    down[id] = time.time()  # Record the time when the vehicle crosses the red line

    # If the vehicle is going down and crosses the blue line
    if id in down:
    if blue_line_y < (cy + offset) and blue_line_y > (cy - offset):
    elapsed_time = time.time() - down[id]  # Calculate the time taken to cross between the lines
    if counter_down.count(id) == 0:  # If the vehicle hasn't been counted yet
        counter_down.append(id)  # Count the vehicle
        distance = 10  # Distance between the lines in meters
        a_speed_ms = distance / elapsed_time  # Calculate speed in meters per second
        a_speed_kh = a_speed_ms * 3.6  # Convert speed to kilometers per hour

        # If the vehicle is speeding, capture its license plate
        if a_speed_kh > 25:
        plate_img = frame[y3:y4, x3:x4]  # Crop the image around the vehicle
        # Use OCR to read the license plate
        plate_text = reader.readtext(plate_img, detail=0,
allowlist='ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789')
```

```
        # Save the license plate image and write the information to a file
        cv2.imwrite(f'license_plates/vehicle_{id}_plate.jpg', plate_img)
        with open('license_plates/speeding_vehicles.txt', 'a') as f:
        f.write(f'Vehicle ID: {id}, Speed: {int(a_speed_kh)} km/h, License Plate:
{"".join(plate_text)}\n')

        # Mark the vehicle on the frame with a circle and bounding box
        cv2.circle(frame, (cx, cy), 4, (0, 0, 255), -1)
        cv2.rectangle(frame, (x3, y3), (x4, y4), (0, 255, 0), 2)
        # Display the vehicle's ID and speed on the frame
        cv2.putText(frame, str(id), (x3, y3), cv2.FONT_HERSHEY_COMPLEX, 0.6, (255,
255, 255), 1)
        cv2.putText(frame, str(int(a_speed_kh)) + 'Km/h', (x4, y4),
cv2.FONT_HERSHEY_COMPLEX, 0.8, (0, 255, 255), 2)

    # Check if the vehicle crosses the blue line (going up)
    if blue_line_y < (cy + offset) and blue_line_y > (cy - offset):
    up[id] = time.time()  # Record the time when the vehicle crosses the blue line

    # If the vehicle is going up and crosses the red line
    if id in up:
    if red_line_y < (cy + offset) and red_line_y > (cy - offset):
    elapsed1_time = time.time() - up[id]  # Calculate the time taken to cross between the lines
    if counter_up.count(id) == 0:  # If the vehicle hasn't been counted yet
        counter_up.append(id)  # Count the vehicle
        distance1 = 10  # Distance between the lines in meters
        a_speed_ms1 = distance1 / elapsed1_time  # Calculate speed in meters per second
        a_speed_kh1 = a_speed_ms1 * 3.6  # Convert speed to kilometers per hour

        # If the vehicle is speeding, capture its license plate
        if a_speed_kh1 > 25:
        plate_img = frame[y3:y4, x3:x4]  # Crop the image around the vehicle
        # Use OCR to read the license plate
        plate_text = reader.readtext(plate_img, detail=0,
allowlist='ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789')

        # Save the license plate image and write the information to a file
        cv2.imwrite(f'license_plates/vehicle_{id}_plate.jpg', plate_img)
        with open('license_plates/speeding_vehicles.txt', 'a') as f:
        f.write(f'Vehicle ID: {id}, Speed: {int(a_speed_kh1)} km/h, License Plate:
{"".join(plate_text)}\n')

        # Mark the vehicle on the frame with a circle and bounding box
        cv2.circle(frame, (cx, cy), 4, (0, 0, 255), -1)
        cv2.rectangle(frame, (x3, y3), (x4, y4), (0, 255, 0), 2)
        # Display the vehicle's ID and speed on the frame
        cv2.putText(frame, str(id), (x3, y3), cv2.FONT_HERSHEY_COMPLEX, 0.6, (255,
255, 255), 1)
```

```
                cv2.putText(frame, str(int(a_speed_kh1)) + 'Km/h', (x4, y4),
cv2.FONT_HERSHEY_COMPLEX, 0.8, (0, 255, 255), 2)

        # Define colors for text and lines on the frame
        text_color = (0, 0, 0)  # Black color for text
        yellow_color = (0, 255, 255)  # Yellow color for background
        red_color = (0, 0, 255)  # Red color for lines
        blue_color = (255, 0, 0)  # Blue color for lines

        # Draw a yellow rectangle for displaying the vehicle counts
        cv2.rectangle(frame, (0, 0), (250, 90), yellow_color, -1)

        # Draw the red and blue lines on the frame
        cv2.line(frame, (172, 198), (774, 198), red_color, 2)
        cv2.putText(frame, 'Red Line', (172, 198), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
text_color, 1, cv2.LINE_AA)

        cv2.line(frame, (8, 268), (927, 268), blue_color, 2)
        cv2.putText(frame, 'Blue Line', (8, 268), cv2.FONT_HERSHEY_SIMPLEX, 0.5, text_color,
1, cv2.LINE_AA)

        # Display the counts of vehicles going up and down
        cv2.putText(frame, 'Going Down - ' + str(len(counter_down)), (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, text_color, 1, cv2.LINE_AA)
        cv2.putText(frame, 'Going Up - ' + str(len(counter_up)), (10, 60),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, text_color, 1, cv2.LINE_AA)

        # Save the current frame with detected objects and information
        frame_filename = f'detected_frames/frame_{count}.jpg'
        cv2.imwrite(frame_filename, frame)

        # Write the processed frame to the output video
        out.write(frame)

# Release resources after the video processing is completed
cap.release()
out.release()
cv2.destroyAllWindows()
```