Aim

The aim of this project is to implement an object detection and tracking system using YOLOv8 for identifying vehicles in video frames, and to compute the speed of these vehicles as they cross predefined lines in a video.

Objective

1. **Object Detection:** Utilize YOLOv8 to detect and classify objects in video frames, focusing on vehicles.
2. **Object Tracking:** Track detected vehicles across frames using a custom tracker.
3. **Speed Calculation:** Calculate the speed of vehicles based on the time taken to cross two predefined lines in the video.
4. **Visualization:** Display annotated video frames with detected vehicles, bounding boxes, tracking IDs, and lines indicating measurement zones.

Summary

This project employs YOLOv8, a state-of-the-art object detection model, to detect vehicles in a video. Detected vehicles are tracked using a custom tracking algorithm. Two horizontal lines are drawn on the video frame to measure the speed of vehicles by calculating the time taken for vehicles to cross from one line to the other. The processed video is displayed with bounding boxes, tracking IDs, and additional visual markers.

Tools and Libraries Used

1. **YOLOv8 (Ultralytics):** For object detection.
2. **OpenCV:** For video processing and visualization.
3. **PyTorch and torchvision:** For model operations.
4. **Pandas:** For data manipulation.
5. **Custom Tracker Class:** For tracking objects across frames.

Procedure

1. **Setup Environment:**
    - Install required libraries and packages.
    - Import necessary modules and initialize the YOLOv8 model with pre-trained weights.
2. **Initialize Video Capture and Tracking:**
    - Open the video file using OpenCV.
    - Create a Tracker object for vehicle tracking.
3. **Process Video Frames:**

- Read frames from the video.
- Resize the frame for consistency.
- Run object detection using YOLOv8 to identify vehicles.

4. **Track and Annotate Vehicles:**
    - Extract bounding boxes for detected vehicles.
    - Update the tracker with new bounding boxes.
    - Draw bounding boxes, center points, and tracking IDs on the frame.
5. **Speed Calculation:**
    - Detect when a vehicle crosses the predefined lines.
    - Record timestamps and compute the speed based on the time difference and the distance between the lines.
6. **Display Results:**
    - Draw lines and annotations on the video frame.
    - Show the processed frame with visual markers.
7. **Save and Cleanup:**
    - Save the annotated frames to a video file.
    - Release video capture and close all windows.

CODE:

```
import cv2

import os

import pandas as pd

from ultralytics import YOLO

from tracker import Tracker


model = YOLO('yolov8s.pt')


cap = cv2.VideoCapture('highway.mp4')

#cap = cv2.VideoCapture('highway_mini.mp4')
```

```
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']


count = 0

tracker = Tracker()

down = {}

up = {}

counter_down = []

counter_up = []


red_line_y = 198

blue_line_y = 268

offset = 6


# Create a folder to save frames

if not os.path.exists('detected_frames'):

    os.makedirs('detected_frames')


fourcc = cv2.VideoWriter_fourcc(*'XVID')

out = cv2.VideoWriter('output.avi', fourcc, 20.0, (1020, 500))


while True:

    ret, frame = cap.read()

    if not ret:

        break
```

```
count += 1
# if count % 2 != 0:
#     continue
frame = cv2.resize(frame, (1020, 500))

results = model.predict(frame)
a = results[0].boxes.data
a = a.detach().cpu().numpy()
px = pd.DataFrame(a).astype("float")
list = []

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]
    if 'car' in c:
        list.append([x1, y1, x2, y2])
bbox_id = tracker.update(list)

for bbox in bbox_id:
    x3, y3, x4, y4, id = bbox
    cx = int(x3 + x4) // 2
    cy = int(y3 + y4) // 2

    # if red_line_y < (cy + offset) and red_line_y > (cy - offset):
    #     down[id] = cy
```

```
        # if id in down:
        #     if blue_line_y < (cy + offset) and blue_line_y > (cy - offset):
        #         cv2.circle(frame, (cx, cy), 4, (0, 0, 255), -1)
        #         cv2.putText(frame, str(id), (cx, cy), cv2.FONT_HERSHEY_COMPLEX, 0.8, (0, 255, 255), 2)
        #         counter_down.add(id)  # Add ID to set

        # if blue_line_y < (cy + offset) and blue_line_y > (cy - offset):
        #     up[id] = cy
        # if id in up:
        #     if red_line_y < (cy + offset) and red_line_y > (cy - offset):
        #         cv2.circle(frame, (cx, cy), 4, (0, 0, 255), -1)
        #         cv2.putText(frame, str(id), (cx, cy), cv2.FONT_HERSHEY_COMPLEX, 0.8, (0, 255, 255), 2)
        #         counter_up.add(id)  # Add ID to set



        if red_line_y<(cy+offset) and red_line_y > (cy-offset):
           down[id]=time.time()   # current time when vehichle touch the first line
        if id in down:


           if blue_line_y<(cy+offset) and blue_line_y > (cy-offset):
            elapsed_time=time.time() - down[id]  # current time when vehicle touch the second line. Also we a re minusing the previous time ( current time of line 1)
            if counter_down.count(id)==0:
               counter_down.append(id)
               distance = 10 # meters - distance between the 2 lines is 10 meters
               a_speed_ms = distance / elapsed_time
               a_speed_kh = a_speed_ms * 3.6  # this will give kilometers per hour for each vehicle. This is the condition for going downside
               cv2.circle(frame,(cx,cy),4,(0,0,255),-1)
```

```
            cv2.rectangle(frame, (x3, y3), (x4, y4), (0, 255, 0), 2)  # Draw bounding box
            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)



      #####going UP#####
      if blue_line_y<(cy+offset) and blue_line_y > (cy-offset):
        up[id]=time.time()
      if id in up:


        if red_line_y<(cy+offset) and red_line_y > (cy-offset):
          elapsed1_time=time.time() - up[id]
          # formula of speed= distance/time  (distance travelled and elapsed time) Elapsed time is It
represents the duration between the starting point and the ending point of the movement.
          if counter_up.count(id)==0:
            counter_up.append(id)
            distance1 = 10 # meters  (Distance between the 2 lines is 10 meters )
            a_speed_ms1 = distance1 / elapsed1_time
            a_speed_kh1 = a_speed_ms1 * 3.6
            cv2.circle(frame,(cx,cy),4,(0,0,255),-1)
            cv2.rectangle(frame, (x3, y3), (x4, y4), (0, 255, 0), 2)  # Draw bounding box
            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
)
```

```
    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

    cv2.rectangle(frame, (0, 0), (250, 90), yellow_color, -1)

    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)

    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 frame
    frame_filename = f'detected_frames/frame_{count}.jpg'
    cv2.imwrite(frame_filename, frame)

    out.write(frame)

    cv2.imshow("frames", frame)
    if cv2.waitKey(1) & 0xFF == 27:
    #if cv2.waitKey(0) & 0xFF == 27:
        break
```

```
cap.release()
out.release()
cv2.destroyAllWindows()
import os
import time
down = {}
up = {}
counter_down = []
counter_up = []

red_line_y = 198
blue_line_y = 268
offset = 6

# Create a folder to save frames
if not os.path.exists('detected_frames'):
    os.makedirs('detected_frames')

fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter('output.avi', fourcc, 20.0, (1020, 500))

while True:
    ret, frame = cap.read()
    if not ret:
        break
    count += 1
    # if count % 2 != 0:
    #     continue
```

```
frame = cv2.resize(frame, (1020, 500))


results = model.predict(frame)
a = results[0].boxes.data
a = a.detach().cpu().numpy()
px = pd.DataFrame(a).astype("float")
list = []

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]
    if 'car' in c:
        list.append([x1, y1, x2, y2])
bbox_id = tracker.update(list)

for bbox in bbox_id:
    x3, y3, x4, y4, id = bbox
    cx = int(x3 + x4) // 2
    cy = int(y3 + y4) // 2

    if red_line_y<(cy+offset) and red_line_y > (cy-offset):
       down[id]=time.time()   # current time when vehichle touch the first line
    if id in down:

       if blue_line_y<(cy+offset) and blue_line_y > (cy-offset):
```

```
        elapsed_time=time.time() - down[id]  # current time when vehicle touch the second line. Also we
a re minusing the previous time ( current time of line 1)
        if counter_down.count(id)==0:
          counter_down.append(id)
          distance = 10 # meters
          a_speed_ms = distance / elapsed_time
          a_speed_kh = a_speed_ms * 3.6  # this will give kilometers per hour for each vehicle. This is the
condition for going downside
          cv2.circle(frame,(cx,cy),4,(0,0,255),-1)
          cv2.rectangle(frame, (x3, y3), (x4, y4), (0, 255, 0), 2)  # Draw bounding box
          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)



    #####going UP blue line#####
    if blue_line_y<(cy+offset) and blue_line_y > (cy-offset):
      up[id]=time.time()
    if id in up:

      if red_line_y<(cy+offset) and red_line_y > (cy-offset):
        elapsed1_time=time.time() - up[id]
        # formula of speed= distance/time
        if counter_up.count(id)==0:
          counter_up.append(id)
          distance1 = 10 # meters  (Distance between the 2 lines is 10 meters )
          a_speed_ms1 = distance1 / elapsed1_time
          a_speed_kh1 = a_speed_ms1 * 3.6
          cv2.circle(frame,(cx,cy),4,(0,0,255),-1)
          cv2.rectangle(frame, (x3, y3), (x4, y4), (0, 255, 0), 2)  # Draw bounding box
```

```
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
)
```

```
    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

    cv2.rectangle(frame, (0, 0), (250, 90), yellow_color, -1)

    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)

    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 frame
    frame_filename = f'detected_frames/frame_{count}.jpg'
    cv2.imwrite(frame_filename, frame)
```

```
    out.write(frame)


    cv2.imshow("frames", frame)
    if cv2.waitKey(1) & 0xFF == 27:
    #if cv2.waitKey(0) & 0xFF == 27:
        break
cap.release()
out.release()
cv2.destroyAllWindows()
```

Highlights

1. **YOLOv8 Integration:** Utilizes the latest YOLOv8 model for high-accuracy object detection.
2. **Custom Tracking:** Implements a custom tracker to maintain object identities across frames.
3. **Speed Calculation:** Computes vehicle speed based on crossing time between two lines, demonstrating practical application of object tracking.
4. **Real-Time Visualization:** Annotates video frames in real-time with bounding boxes, tracking IDs, and speed calculations.

Conclusion

This project successfully demonstrates the application of YOLOv8 for vehicle detection and tracking in video. By integrating object detection with custom tracking and speed calculation methods, the system provides valuable insights into vehicle movement and speed. The use of visualization tools like OpenCV enhances the interpretability of the results, making it a robust solution for traffic analysis and similar applications. Future improvements could include refining the speed calculation accuracy, optimizing the tracker, and extending the system to handle more complex scenarios or multiple object types.