Vehicle Detection, Tracking and Speed Estimation

Aim

To develop a real-time system for detecting vehicles and estimating their speed using video footage, leveraging YOLOv8 for object detection and custom tracking algorithms.

Objective

1. To detect and classify vehicles in video frames using YOLOv8.
2. To track the movement of detected vehicles across video frames.
3. To calculate the speed of the vehicles based on their movement across predefined lines in the video.
4. To display and record the speed and movement data in real-time.

Software Requirements

- **Operating System:** Windows, macOS, or Linux
- **Programming Language:** Python 3.x
- **Libraries and Frameworks:**
    - OpenCV
    - Pandas
    - Ultralytics YOLOv8
    - Custom Tracker (provided)
- **Development Environment:** Any Python IDE (e.g., PyCharm, VSCode, Jupyter Notebook)

Hardware Requirements

- **Processor:** Multi-core processor with at least 2.5 GHz clock speed
- **RAM:** Minimum 8 GB (16 GB recommended for better performance)
- **Graphics Card:** Dedicated GPU (NVIDIA recommended) for faster object detection using YOLOv8
- **Storage:** Minimum 10 GB free space for storing video files and frames
- **Camera:** High-definition camera (optional, if capturing real-time footage)

Setup

1. **Install Dependencies:** Ensure you have the required libraries installed:

   ```
   pip install opencv-python pandas ultralytics
   ```

2. **Load the YOLO Model:**

   ```
   from ultralytics import YOLO
   ```

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

3. **Initialize the Video Capture:**

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

4. **Define the List of Classes:**

class_list = ['person', 'bicycle', 'car', ... , 'hair drier', 'toothbrush']

5. **Initialize the Tracker and Other Variables:**

```
python
Copy code
tracker = Tracker()
down = {}
up = {}
counter_down = []
counter_up = []

red_line_y = 198
blue_line_y = 268
offset = 6
```

6. **Create Folder to Save Frames and Initialize Video Writer:**

```
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))
```

Main Loop

The main loop processes each frame of the video, performs vehicle detection, tracking, and speed estimation.

1. **Read and Resize Frame:**

```
ret, frame = cap.read()
if not ret:
    break
frame = cv2.resize(frame, (1020, 500))
```

2. **Detect Vehicles using YOLOv8:**

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

3. **Filter Detected Objects to Only Include Cars:**

```
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])
```

4. **Track Vehicles:**

```
bbox_id = tracker.update(list)
```

5. **Calculate Speed:**

```
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()
    if id in down:
        if blue_line_y < (cy + offset) and blue_line_y > (cy - offset):
            elapsed_time = time.time() - down[id]
            if counter_down.count(id) == 0:
                counter_down.append(id)
                distance = 10
                a_speed_ms = distance / elapsed_time
                a_speed_kh = a_speed_ms * 3.6
                cv2.circle(frame, (cx, cy), 4, (0, 0, 255), -1)
                cv2.rectangle(frame, (x3, y3), (x4, y4), (0, 255, 0), 2)
                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)

    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]
            if counter_up.count(id) == 0:
```

```
                    counter_up.append(id)
                    distance1 = 10
                    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)
                    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)
```

6. **Draw Lines and Count:**

```
text_color = (0, 0, 0)
yellow_color = (0, 255, 255)
red_color = (0, 0, 255)
blue_color = (255, 0, 0)

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)
```

7. **Save Frame and Show Result:**

```
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:
    break
```

8. **Release Resources:**

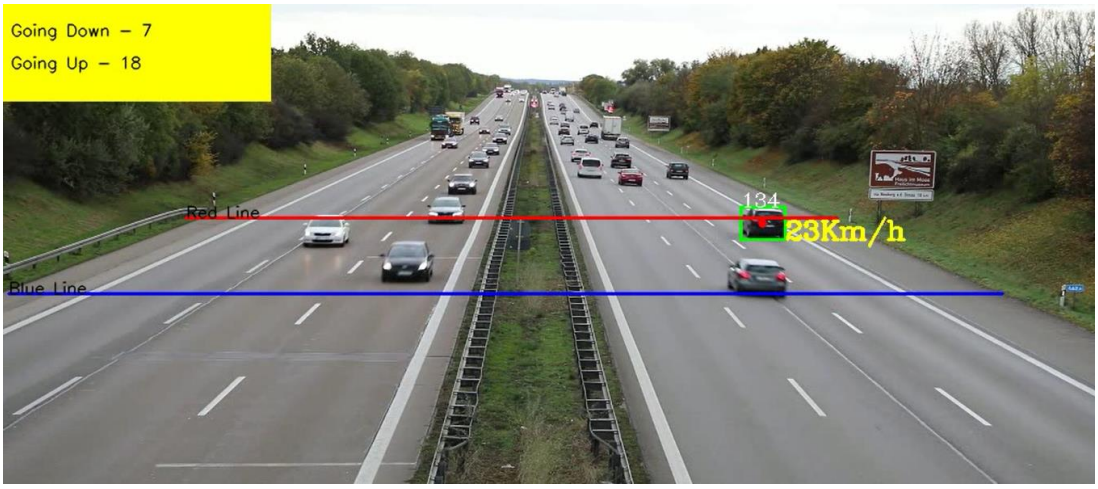
```
cap.release()
out.release()
cv2.destroyAllWindows()
```

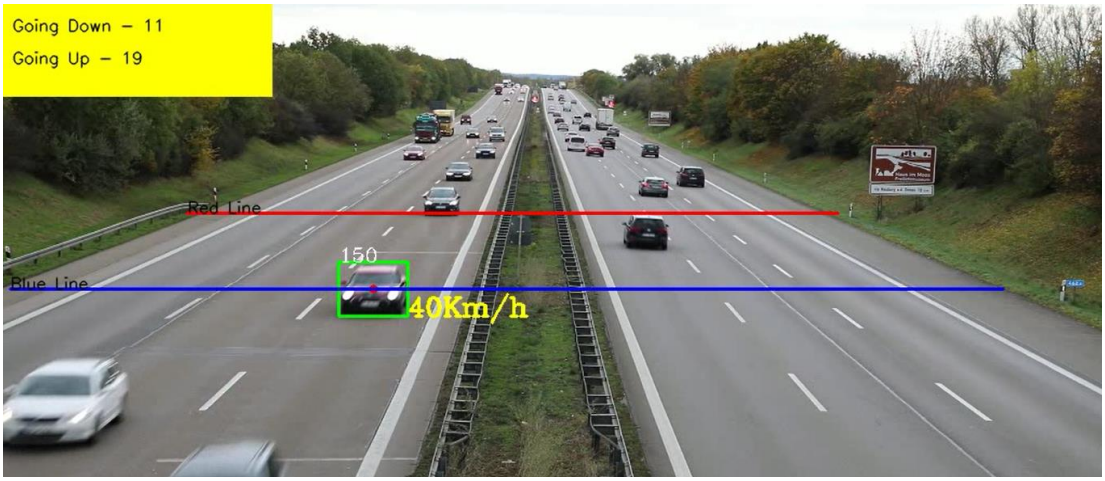
tracker.py:

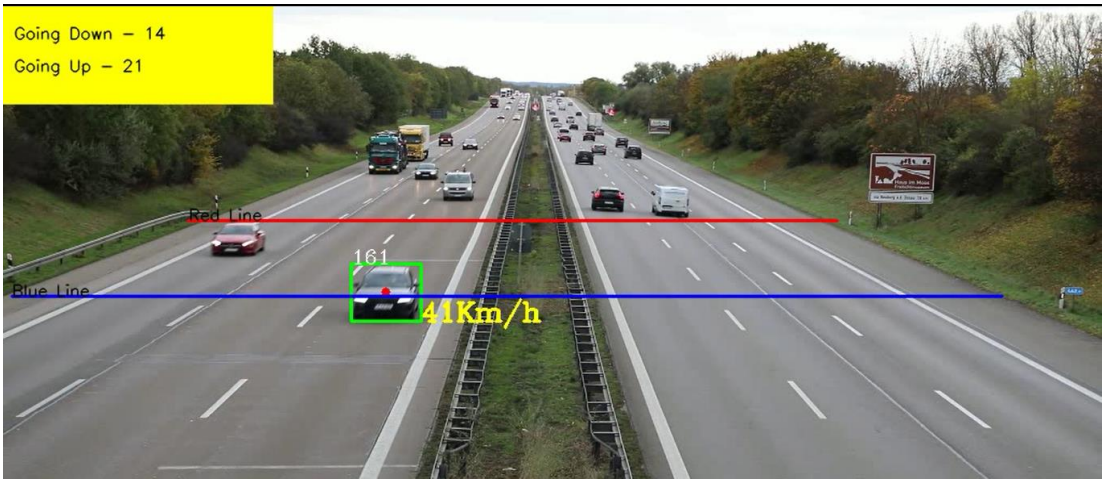
```
import math

class Tracker:
    def __init__(self):
        # Store the center positions of the objects
        self.center_points = {}
        # Keep the count of the IDs
        # each time a new object id detected, the count will increase by one
        self.id_count = 0


    def update(self, objects_rect):
        # Objects boxes and ids
        objects_bbs_ids = []

        # Get center point of new object
        for rect in objects_rect:
            x, y, w, h = rect
            cx = (x + x + w) // 2
            cy = (y + y + h) // 2

            # Find out if that object was detected already
            same_object_detected = False
            for id, pt in self.center_points.items():
                dist = math.hypot(cx - pt[0], cy - pt[1])

                if dist < 35:
                    self.center_points[id] = (cx, cy)
                    print(self.center_points)
                    objects_bbs_ids.append([x, y, w, h, id])
                    same_object_detected = True
                    break

            # New object is detected we assign the ID to that object
            if same_object_detected is False:
                self.center_points[self.id_count] = (cx, cy)
                objects_bbs_ids.append([x, y, w, h, self.id_count])
                self.id_count += 1

        # Clean the dictionary by center points to remove IDS not used anymore
        new_center_points = {}
        for obj_bb_id in objects_bbs_ids:
            _, _, _, _, object_id = obj_bb_id
            center = self.center_points[object_id]
            new_center_points[object_id] = center
```

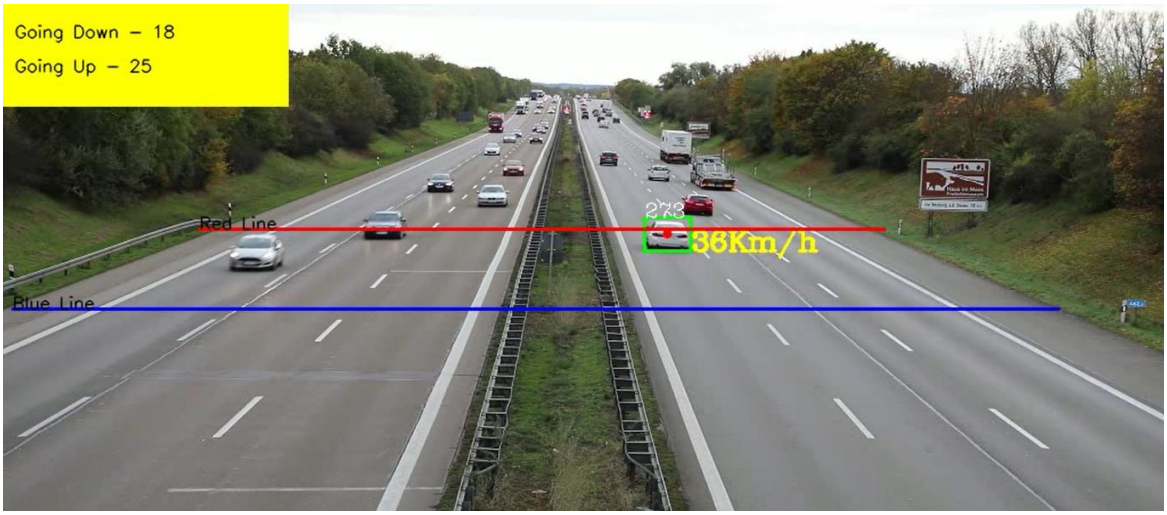
```
        self.center_points = new_center_points.copy()
        return objects_bbs_ids
```

OUTPUT:









Challenges

1. **Real-time Processing:**
    - Ensuring the detection and tracking process runs in real-time can be challenging due to the high computational load.
2. **Accurate Speed Calculation:**
    - Precise calculation of speed requires accurate detection of vehicles at the exact moments they cross the predefined lines. Small timing errors can lead to significant speed estimation errors.
3. **Occlusions:**
    - Vehicles can be occluded by other objects or vehicles, making tracking and speed estimation more difficult.
4. **Lighting Conditions:**
    - Variations in lighting conditions can affect the performance of the YOLO model and the tracker.

Results

- The system successfully detects vehicles and tracks their movements across predefined lines.
- It calculates the speed of vehicles in both directions (up and down) and displays this information in real-time.
- The processed frames are saved, and an output video is generated showing the detection, tracking, and speed estimation results.