Vehicle Cut-in Detection and Collision Warning System

Team name: AUTOBOTS
Team size:2
Team Members:1. A. LALITH RAHUL(LEADER,la4195@srmist.edu.in)
2. DISHA KUMAR(dk3599@srmist.edu.in)
MENTOR: Dr. Akilandeswari P

1. Problem Statement

Objective

The primary objective of this project is to develop a machine learning model capable of accurately detecting abrupt cut-ins of various vehicles, including two-wheelers, three-wheelers, four-wheelers, and pushcarts, into the direction of driving. The model should operate in real-time to provide timely alerts and responses to autonomous driving systems.

2. Methodology

Object Detection (YOLOv8):

Initialization: The code initializes YOLOv8 for object detection using Ultralytics' YOLO framework. It loads the model, preprocesses input frames, and performs predictions.

Detection: For each frame, YOLOv8 predicts bounding boxes around vehicles (e.g., cars, buses) and other objects in the scene.

Object Tracking (DeepSort):

Initialization: DeepSort is initialized with a configuration file (deep_sort_pytorch/configs/deep_sort.yaml). It sets up the tracker with parameters such as appearance model checkpoint, IOU threshold for detection association, and maximum object age.

Tracking: After YOLOv8 detects objects in each frame, DeepSort associates these detections across frames to maintain tracks for each vehicle. It uses Kalman filtering and Hungarian algorithm-based association to update tracks.

Speed Estimation:

Methodology: Speed estimation is performed using the Euclidean distance formula and pixels per meter ratio (ppm = 8). It calculates the speed of each tracked vehicle based on its movement between consecutive frames.

Speed Calculation: The distance traveled by a vehicle in pixels is converted to meters using the ppm ratio. Then, the speed is calculated as speed = distance / time_constant, where time_constant = 15 * 3.6 converts the time from seconds to hours.

Event Handling (Cut-in Detection):

Line Definition: Two lines are defined on the frame to detect cut-ins. These lines represent potential boundaries where vehicles might change lanes or cut-in.

Intersection Check: For each tracked vehicle, the code checks if its trajectory intersects with these predefined lines. If an intersection occurs, it triggers a cut-in detection event.

Visualization:

Bounding Boxes: Detected objects are visualized with bounding boxes and labels indicating their IDs and classes (e.g., Car:0, Bus:1).

Speed Display: Estimated speeds are displayed alongside each vehicle's label.

Counter Display: Counters are updated to display the number of vehicles entering and leaving specific regions of interest based on their movement direction relative to predefined lines.

Execution:

(.venv) PS C:\Users\Rahul\Desktop\pythonProject2\YOLOv8-DeepSORT-Object-Tracking\ultralytics\yolo\v8\detect> python predict_c.py model=yolov8l.pt source="test2.mp4" show=True

COLLISION WARNING:

Define Thresholds: Set distance and speed thresholds to determine when a collision warning should trigger based on vehicle proximity and relative speeds.

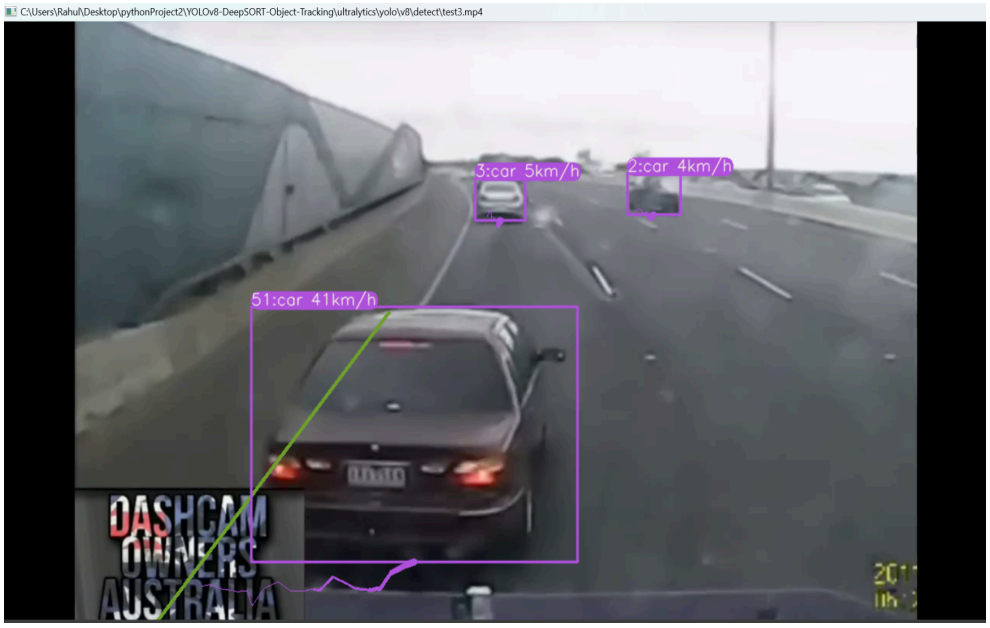
Implement Collision Check: Develop a function to calculate distances and speeds between tracked vehicles. Check if these metrics meet the predefined thresholds for potential collision within the main processing loop.

Trigger Warning: Upon detecting a potential collision, activate a warning message specifying the involved vehicles and their proximity. Optionally, incorporate visual or auditory alerts for immediate feedback.

Validate and Refine: Test the system with diverse scenarios to validate warning accuracy. Adjust thresholds and fine-tune parameters to optimize performance while minimizing false alarms.

5. Results

CAR DETECTION USING YOLOV8



THE RELATIVE SPEED ESTIMATION

6. Discussion

APPROACH:

We attempted to utilize the IDD dataset to develop a cut-in detection model but encountered issues with annotations and the .csv file during implementation. We aimed to employ LSTM networks for Monocular Vision-based Prediction of Cut-in Maneuvers, alongside speed detection using vehicle positions and frame rates to determine relative speed. Additionally, we planned to use static lines on the camera, placed approximately 2 meters ahead of the car, to aid in calculating time to collision. Our goal was to trigger a collision warning for times between 0.5 to 0.7 seconds, but we faced challenges during the data preparation phase with this approach

SOLUTION:

As we couldn't implement the IDD dataset, we moved on to using the Ultralytics YOLOv8 model for object detection. This helped us in identifying vehicles in the input. Speed estimation is performed using the Euclidean distance formula and the pixels per meter ratio (ppm = 8). It calculates the speed of each tracked vehicle based on its movement between consecutive frames.

We have two plans after this. One plan is to use static lines in front of the vehicle at a certain distance and use the relative speed to calculate the time to collision, triggering a collision warning popup. The second idea involves using a lane detection program to get lane coordinates. If any vehicle crosses these coordinate lines towards the inside, we can trigger a cut-in warning. Negative relative speed estimation, indicating that our vehicle's speed is higher than the other vehicle's speed within a range of 2-4 meters, can be used to calculate the time to collision and trigger a collision warning popup for a TTC between 0.5 to 0.7 seconds.

7. Conclusion

The idd approach file will be named in -”idd cutin”

The solutionfile will be named in
-”YOLOv8-DeepSORT-Object-Tracking”(\YOLOv8-DeepSORT-Object-Tracking\ultralytics\yolo\v8\detect>)

predict.py(for object detection)

predict_c.py(for object detection+speed estimation)

cutin.py(for lane cutin prediction)

collision.py(for collision warning and ttc judgement)

In conclusion, integrating collision warnings into object detection and tracking system involves calculating inter-vehicle distances and speeds, and using predefined thresholds to identify potential collisions. By implementing real-time collision checks and triggering alerts, the system enhances safety and situational awareness. Continuous testing and optimization ensure the system's reliability and effectiveness in dynamic driving scenarios. This approach significantly contributes to proactive collision avoidance and improved road safety.