Problem Statement:

Vehicle cut-in detection is a critical challenge in automotive collision avoidance systems. Current state-of-the-art solutions rely on three key components: vehicle detection, distance estimation, and time-to-collision (TTC) calculation. However, these systems must rapidly and accurately identify potential cut-in scenarios in complex, dynamic traffic environments. The problem lies in integrating these components effectively, processing data in real-time, and making split-second decisions to prevent collisions while minimizing false alarms. Addressing this challenge requires advanced sensor fusion, trajectory prediction, and robust decision-making algorithms capable of handling diverse driving conditions and vehicle behaviors.

Unique Idea:

- The code presents a vehicle cut-in detection system using computer vision and object detection.
- It combines YOLO-based object detection with distance estimation and Time-to-Collision (TTC) calculation to identify potential collision risks in real-time.
- It's designed to process either a video feed or a batch of images, annotating detected vehicles with distance, TTC, and collision risk information.
- Implement a dynamic TTC calculation that considers the speed and trajectory of detected vehicles. This enhancement would use object tracking techniques to predict the future position of vehicles based on their current speed and direction.
- By integrating this feature, the system can provide more accurate and timely collision warnings, reducing the risk of accidents.

Features Offered:

- Vehicle detection using YOLO model
- Real-time processing of video streams and batch processing of images
- Distance estimation using known object width and focal length
- Time-to-Collision (TTC) calculation
- Collision risk assessment
- Visual annotations including bounding boxes, object class, confidence score, distance, TTC, and collision risk
- Color-coded warnings (red for high collision risk)
- Support for multiple vehicle types (bicycle, car, motorcycle, bus, truck)
- Can use any input type like video or image directory

Process flow:

a. User selects input type (video or image directory)

b. Load YOLO model and define class names and colors

c. For each frame (video) or image:

It perform object detection using YOLO and filter detected objects for vehicle classes

d. For each detected vehicle it estimate the following:

Calculate perceived width in pixels

Estimate distance using pinhole camera model

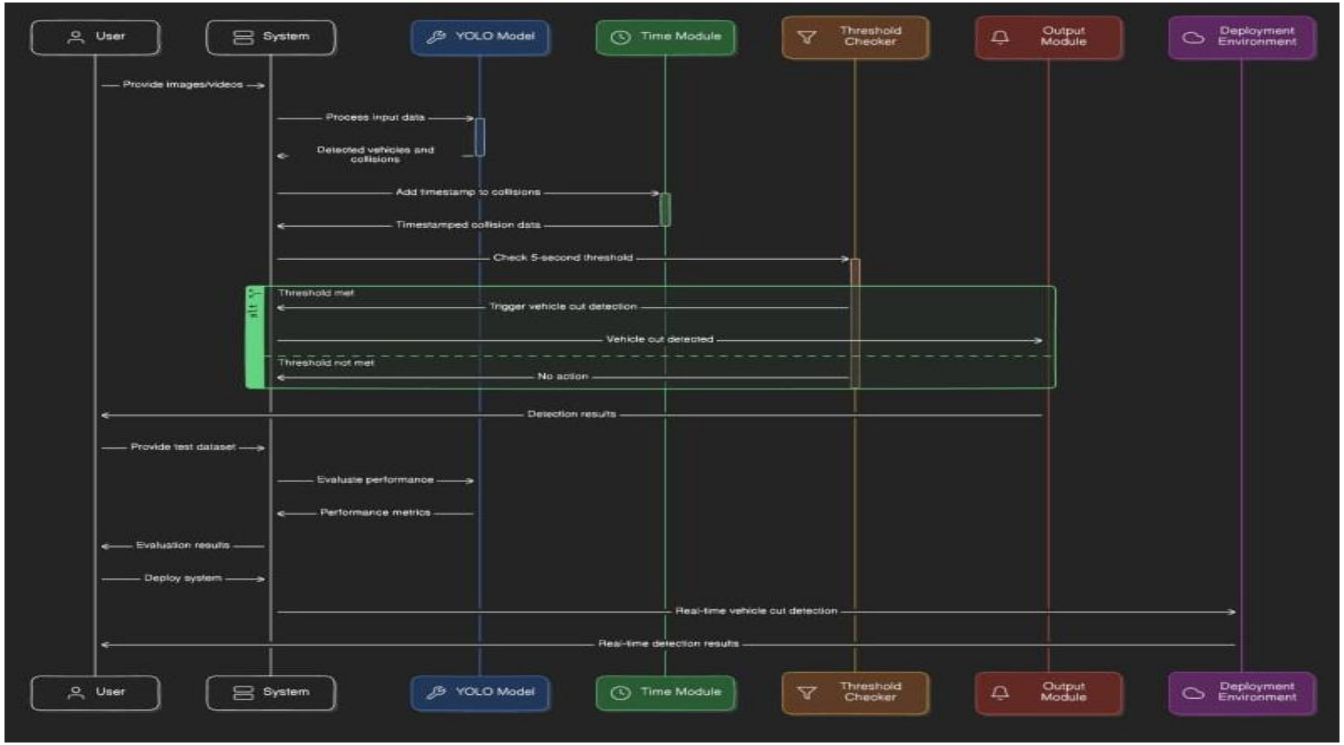
Calculate TTC using placeholder relative speed

Assess collision risk based on distance threshold

Draw bounding box and annotate with object info, distance, TTC, and collision risk.

Collision risk will be assisted using red color-coded warning for high collision risk.

Architecture Diagram:



Technologies used:

- Python: The primary programming language used for implementing the entire solution, leveraging its versatility in handling image processing tasks and integrating with deep learning frameworks like YOLO.
- OpenCV (cv2): Used for image and video processing, drawing annotations, and displaying results.
- YOLO (You Only Look Once): Specifically, YOLOv8 from Ultralytics for object detection.
- cvzone: A computer vision library used for adding text rectangles to the images.
- NumPy: Implied use for numerical operations.

Team members and contribution:

Jerik Brahmos A(URK22CS5036) played a crucial role in laying foundation of the project, providing the initial framework and guiding the development process. His expertise and support were instrumental in bringing the code to its completion.

Aflah R Basheer (URK22CS7017) took the lead in driving the project to its conclusion. He implemented TTC and distance calculation with various levels of collision detection, and made it adaptable for both image and video processing. Additionally, he created the video presentation for the project.

Liziyo J(URK22CS5057) and Abhanedh Jies(URK22CS1062) were pivotal in managing the project's documentation and presentation. They set up the GitHub repository, created a comprehensive PowerPoint presentation, and authored a detailed report. Their collaborative efforts and assistance were vital in wrapping up the project smoothly.

Conclusion:

We can conclude that the code effectively implements a vehicle cut-in detection system using YOLO, designed to identify and track multiple types of vehicles. It accurately utilizes the Ultralytics YOLO model for object detection, identifies vehicle classes from the COCO dataset, and estimates distances using known object width and focal length, calculates collision risk based on proximity, it annotates detected vehicles with bounding boxes, distance measurements, Time-to-Collision (TTC) calculations, and collision risk assessments and provides clear visual feedback.