Problem Statement

Vehicle cut-in detection using IDD

Category:

Artificial Intelligence, Machine Learning, Deep Learning, Autonomous Driving.

Participants: 1st-4th Semester Students.

Team name: Sapphire (1148)

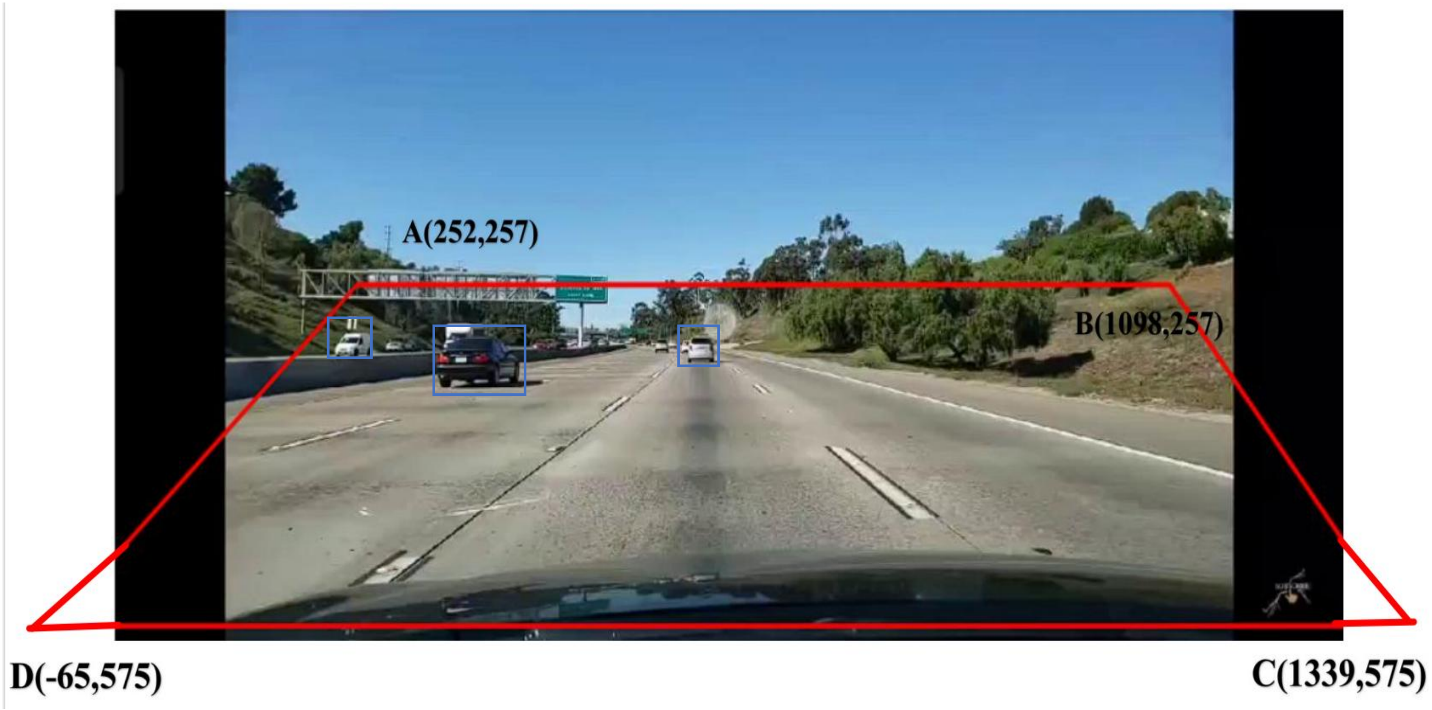
Team members:

Yogesh T N (919)

Vishnu A (936)

Unique Idea Brief (Solution):

The project aims to develop a vehicle cut-in detection system using the Indian Driving Dataset (IDD). The system will involve vehicle detection, calculation of detection distances, and estimation of time to collision (TTC). The project involves using a YOLO model for vehicle detection in video frames, followed by view transformation to map detected coordinates to a top-down perspective. Speed and time-to-collision calculations are performed based on the tracked positions. The process will include data extraction and pre-processing, model training, and deployment.



Raw image coordinates cant be used for measurements so, coordinates on image is transformed to actual coordinated in the road

Distance = The number of pixels bounding box moves.

Features Offered

Vehicle Detection:

- **Accurate Vehicle Detection**: The system can reliably detect vehicles in various traffic scenarios using advanced computer vision models.
- **Real-time Processing**: It provides real-time detection of vehicles, ensuring timely updates and alerts for dynamic driving environments.
- **Multi-class Detection**: The system can distinguish between different types of vehicles (e.g., cars, trucks, motorcycles), enhancing situational awareness.
- **Robust Detection in Adverse Conditions**: It maintains high detection accuracy under varying lighting and weather conditions, ensuring reliability in real-world scenarios.
- **Scalability**: The detection algorithm can handle multiple vehicles simultaneously, providing comprehensive coverage of the driving scene.

Distance Estimation:

- **Detection**: The YOLO model detects vehicles in each video frame, providing bounding box coordinates for each detected vehicle.
- **Point Extraction**: Specific points are extracted from these detection.
- **View Transformation**: The extracted points are transformed from the camera's perspective to a top-down view using a transformation matrix, which corrects for perspective distortion.
- **Tracking**: ByteTrack is used to track the transformed points across consecutive frames, maintaining a history of positions for each tracked vehicle.

- **Distance Calculation**: The distance a vehicle has moved is calculated based on the change in its transformed coordinates over time. This distance, combined with the frame rate, allows for speed estimation and further distance-related metrics like time-to-collision.

Time to Collision (TTC) Calculation:

The algorithm calculates the time to collision (TTC) between your car and the car in front, given their speeds and the distance between them:

Define Variables:

- V_your : Speed of your car.
- V_front : Speed of the car in front.
- D : Distance between your car and the car in front.

Calculate Relative Speed:

- V_relative =V_your − V_front

Determine Time to Collision (TTC):

- If V_relative ≤ 0 , no collision risk.

- If V_relative > 0
- TTC= D / V_relative .

Example:
V_your = 20 m/s
V_front = 15 m/s
V_relative = 5 m/s
D(Distance) = 100 meters

TTS = D / V_relative
= 100/5
Hence, **Time to collision = 20 seconds.**

This approach provides a straightforward method to assess collision risk based on the current speeds and distance between the vehicles.

Process flow

1. **Data Extraction and Preprocessing**
    - Extract the IDD dataset from its source.
    - Preprocess the dataset to ensure it is compatible with the chosen machine learning model requirements. This includes:
        - Annotating vehicle positions and movements.
        - Normalizing and augmenting data to improve model robustness.
        - Splitting the dataset into training, validation, and test sets.

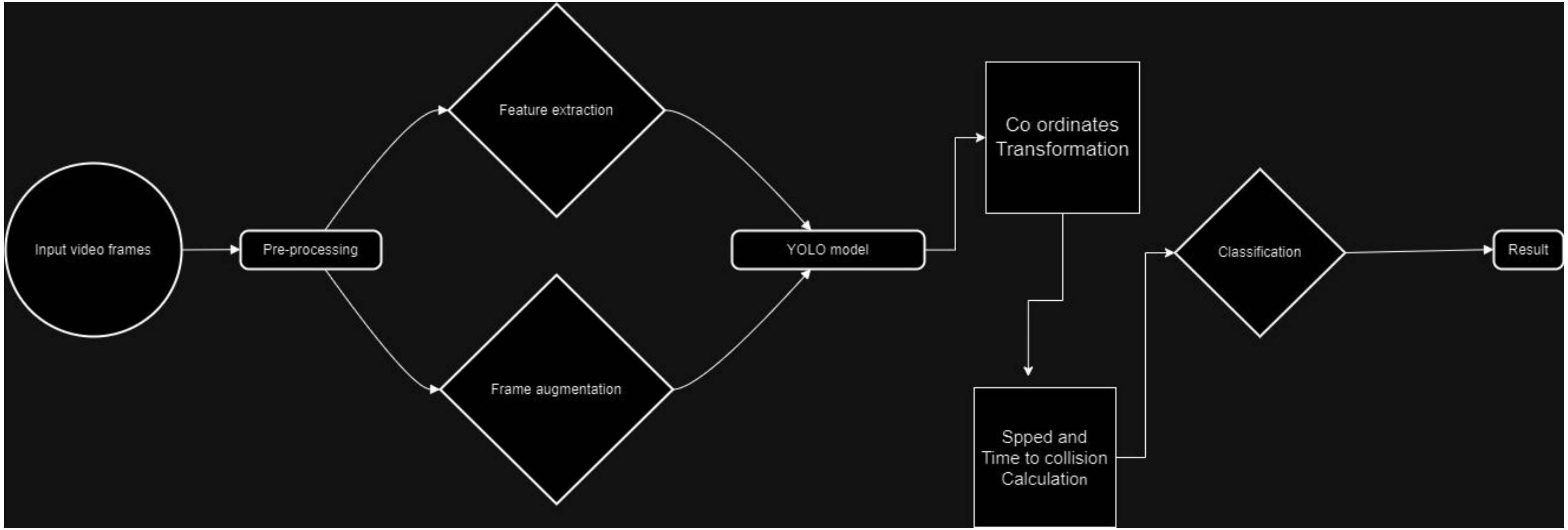
2. **Model Selection and Training:**

- A suitable model architecture for vehicle detection, YOLO.
- Train the model on the preprocessed IDD dataset to detect vehicles and their movements.
- Implement distance calculation by converting the coordinates of detected objects (vehicles) from the camera's perspective to another perspective, usually a top-down view.
- Calculate time to collision (TTC) using relative speed and distance metrics.

3. **Model Optimization and Evaluation:**

- Fine-tune the model to optimize performance metrics like precision, recall, and F1 score.
- Evaluate the model on the test dataset to ensure accuracy and reliability.
- Implement improvements based on evaluation results, such as adjusting hyperparameters or augmenting the training data.

Architecture Diagram:



Technologies used

- **Python**: The primary programming language for the implementation.
- **OpenCV**: Used for video processing and handling image operations.
- **Ultralytics YOLO**: YOLO (You Only Look Once) model for object detection.
- **Supervision**: For handling video frame extraction, annotations, and tracking.
- **ByteTrack**: A tracking algorithm used to maintain object identities across frames.
- **TQDM**: For displaying progress bars during video frame processing.
- **NumPy**: Used for numerical operations and transformations (e.g., handling matrices for view transformation).

Team members and contribution:

TEAM MEMBER 1 : YOGESH T N [919] :

Data Extraction, Preprocessing, and Tracking

Contributions:

- **Data Extraction**: Extracted data from the Indian Driving Dataset (IDD).
- **Data Preprocessing**: Preprocessed the data to convert it into a suitable format for vehicle detection and tracking. This involved cleaning the data, formatting video frames, and setting up the necessary pipeline for processing.
- **Tracking and Calculations**: Integrated ByteTrack for tracking detected vehicles and implemented speed and time-to-collision (TTC) calculations.

Team members and contribution:

TEAM MEMBER 2 : VISHNU A [936]:

Model Fine-Tuning and Application Development

Contributions:

- **Model Training**: Trained the YOLO model for vehicle detection using the preprocessed IDD data. Configured the training environment, set hyperparameters, and monitored the training process.

- **Data Upload**: Managed the upload and organization of the processed data for use in model training and validation.

- **Application Development**: Developed the video processing pipeline, including the view transformation, annotation of video frames, and generation of the output video with annotated frames showing tracked vehicles and calculated speeds.

Conclusion :

The initial phase of the project involved extracting and preprocessing data from the Indian Driving Dataset (IDD), ensuring the dataset was clean, formatted, and ready for vehicle detection and tracking. This meticulous preparation laid a strong foundation for the subsequent model training and processing stages.

In the next phase, the YOLO model was successfully trained using the preprocessed IDD data. This included setting up the training environment, configuring hyperparameters, and running the training process. The result was a model optimized for vehicle detection, which was then integrated with ByteTrack for tracking detected vehicles.

Finally, the development of a comprehensive video processing pipeline provided a robust solution for vehicle cut-in detection. This included view transformation for accurate position mapping, speed and time-to-collision (TTC) calculations, and annotation of video frames. The annotated frames, showing tracked vehicles and calculated speeds, were compiled into an output video, providing a clear visual representation of the detection and tracking results.

Overall, the project has achieved its goals, delivering a powerful and refined vehicle detection and tracking system. This comprehensive solution is now ready to enhance traffic analysis and safety measures, meeting our specific application needs.