Modelling, Simulation and Virtual validation of Adaptive Cruise Control (ACC) Algorithm based on Sensor Fusion

Ragavendran R[1], Manoj Kumar A[2], Bhavan C[3]

{ragavendranr2002@gmail.com[1], amanojkumar2000@gmail.com[2], bhavanka2001@gmail.com[3]}

Department of Mechanical Engineering, Sri Sairam Engineering College[1,2,3]

Abstract. Adaptive Cruise Control (ACC) systems are rapidly becoming an integral part of modern vehicles, contributing to both enhanced driving comfort and increased road safety. Traditional approaches for ACC often rely on a single type of sensor, which could be limiting in dynamically evolving driving scenarios. To improve the system's performance in such complex driving conditions, an enhanced approach using Sensor fusion technique has been discussed. Further, based on the detections the Most Important Object (MIO) lead car is obtained, with which a robust stateful control algorithm determines the appropriate set speed for the Ego vehicle to maintain safe distance with the preceding (lead car) in the same lane, based on the states of Ego and Lead vehicle. This set speed of the Ego Vehicle is maintained by a Vehicle Longitudinal controller. The modelled system has been validated in a closed-loop Virtual Environment with custom scenarios for intricate analysis of system performance.

Keywords: Adaptive Cruise Control, Advanced Driver Assistance Systems, ACC, ADAS, Virtual Validation, Simulink, Sensor fusion.

1 Introduction

The aim of the Adaptive Cruise Control (ACC) algorithm is to regulate vehicular speed based on a driver-defined setpoint, while concurrently ensuring a safe inter-vehicular gap. This functionality enhances both driving comfort and safety, particularly in extended driving scenarios. The enhanced algorithm proposed in this paper is validated by analysing its performance in a virtual simulation environment [1,2] using a co-simulation framework by Automated Driving Toolbox - MATLAB, which facilitates the modelling of Algorithms in Simulink and performance visualisation in a simulated environment rendered through Unreal Engine by Epic Games.

ICSETPSD 2023, November 17-18, Coimbatore, India
Copyright © 2024 EAI
DOI 10.4108/eai.17-11-2023.2342869

2 Sensor Data from Virtual Environment:

The aim of the Adaptive Cruise Control (ACC) algorithm is to regulate vehicular speed based on a driver-defined setpoint, while concurrently ensuring a safe inter-vehicular gap. This functionality enhances both driving comfort and safety, particularly in extended driving scenarios. The enhanced algorithm proposed in this paper is validated by analysing its performance in a virtual simulation environment using a co-simulation framework by Automated Driving Toolbox – MATLAB [3], which facilitates the modelling of Algorithms in Simulink and performance visualisation in a simulated environment rendered through Unreal Engine by Epic Games.

2.1 Camera data:

To have a clear point of view the camera was placed on the centre point of the ego vehicle (in the simulation environment). The camera's intrinsic properties have been modified in a way that helps the system by collecting video frames more suitable for Object and Lane detection algorithms. The parameters include Focal length, Optical centre and Image size. A four-element vector of the form [x y width height] is contained in each row of bounding boxes. This vector indicates the upper left corner and size of that corresponding bounding box in pixels.

2.2 Radar data:

A synthetic data containing the reports of detection of objects from the environment is obtained from the Driving Radar Data generator, which is in the form of Clustered data. In the aim of obtaining the best possible outcome the placement of radar and its azimuth value has been made in such a way that it only covers the drive vehicle's lane. The radar was placed on the front of the ego vehicle (in the simulation environment) with the azimuth value as 5 deg and the distance of the waveform as 35 metres [4], this provides a good area to cover the lane of the ego vehicle. The radar's frequency was set at 10Hz.

Once an object falls under the range of the radar, the sensor detects and delivers information in the form of four vectors with respect to the ego vehicle [x, y, z, vx, vy]. The variables x and y represent the longitudinal and lateral distance and the variables vx and vy represent the relative longitudinal and lateral velocity. Upon encountering an object, the system generates a unique detection ID for each target when the detections are clustered, and each detection represents the centroid of that target.

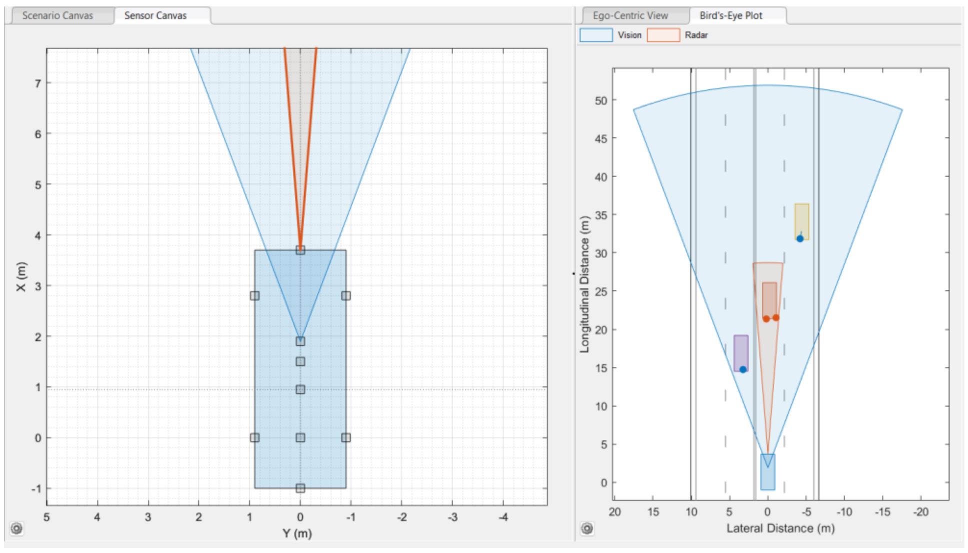


Fig. 1 Sensor coverage and placements

3 Vehicle Detection:

In the context of Adaptive Cruise Control (ACC) within Advanced Driver Assistance Systems (ADAS), object detection serves two primary purposes: it helps identify the lead vehicle to follow and ensures a safe following distance. By accurately detecting vehicles and other obstacles in real-time, the ACC system can autonomously adjust speed and maintain safe driving conditions. In this case, it categorises them into predefined classes, and specifies their positions with bounding boxes in spatial coordinates in a Projective Camera Image.

3.1 ACF:

The ACF (Aggregate Channel Features) object detector is a type of feature extraction algorithm that detects objects in images using a set of predefined features. ACF operates by dividing an input image into small, overlapping regions called "windows". It then extracts a set of low-level features from each window, such as colour, texture, and edges. These low-level features are then combined into a higher-level feature representation, known as "Channel Features".

Channel features describe the presence of certain visual patterns in the image, such as edges or corners, and are used to classify whether the window contains an object or not. The ACF used consists of a AdaBoost classifier to learn how to distinguish between object and non-object windows, based on their channel features.

Imaging data, captured across diverse scenarios and encompassing a range of cars with varying colour profiles, was utilised for model training. A pre-trained ACF detector was used for automated data annotation, supplemented by manual labelling to correct inaccuracies. Further, this newly-labelled data was used to fine-tune the detector model to ensure better detection in the Virtual Environment. As a result, the ACF detector has undergone additional training to improve its ability to recognize specific objects in the custom scenarios as shown in Figure 2.

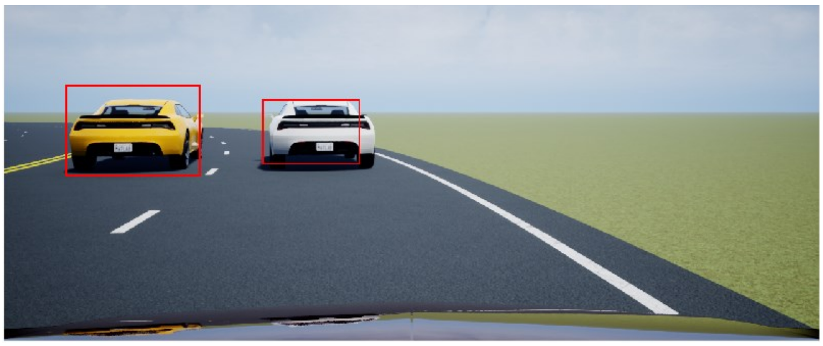


Fig. 2 Vehicle Detection using ACF Detector

4 Lane Detection:

In the context of lane detection, although the vehicle-mounted camera captures a broad field of environmental objects, the primary interest resides on the road surface. To enhance detection accuracy, the captured image is converted into a bird's-eye view image through the Inverse Perspective Mapping (IPM) technique [5], as shown in Fig. 3.2. After which, multiplying the image's RGB matrix by a transformation matrix, which is derived from specific camera sensor parameters such as focal length, principal point, and overall image dimensions.

Lane markings are generally yellow or white, making these the relevant colours for image-based lane detection. In the bird's-eye view, Colour Thresholding is used to isolate these specific pixels. Binary masks are then applied to create a binary image containing only the lane-relevant pixels [6], as shown in Figure 4.

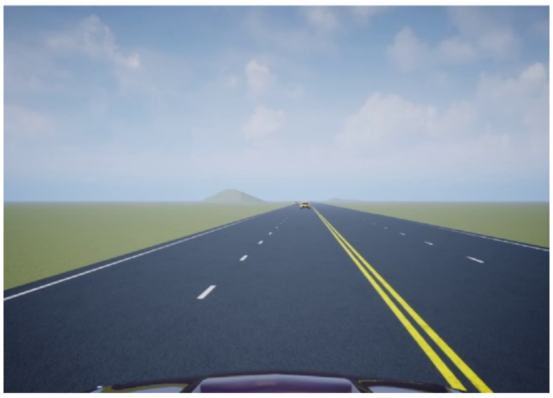


Fig. 3-A. Camera data obtained from 3D environment placed near Ego vehicle's rear-view mirror

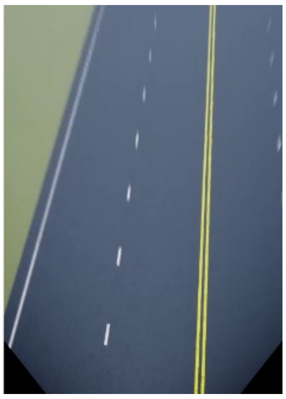


Fig. 3-B. Camera data after Inverse Perspective mapping

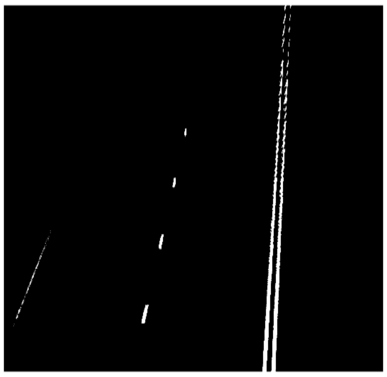


Fig. 4. Binary Image with only Lane markings

However, in some cases where the car in front is yellow or white, the colour thresholding will have noise which causes unexpected results in Lane detection, as the White car will create a large region of white space (1 value in binary image). To resolve this, Object detection data was used to identify the region of image that has the White/Yellow vehicle and nullify the region (make values 0 in binary image) [7]. This approach removes noise to considerable extent, but object detection is only done in the original image and not bird's eye view.

For this, the coordinates of the bounding box from object detection were converted to bird's eye view with the transformation matrix that was used in IPM. With the bounding box coordinates in bird's eye view, a Region of Interest was created where the binary image was nullified, as shown in Figure 6.

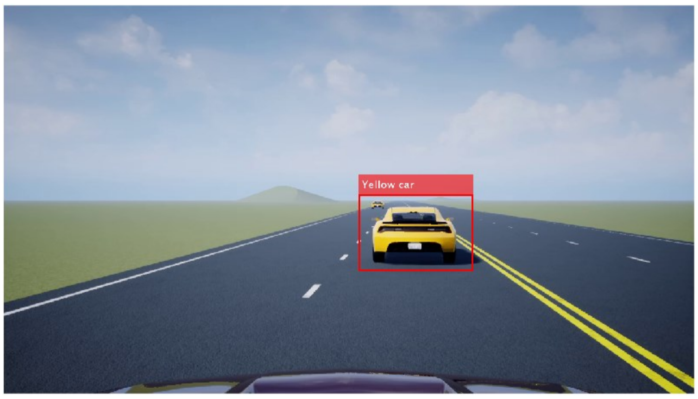


Fig. 5. Scenario with yellow car in front of ego car from Object detection

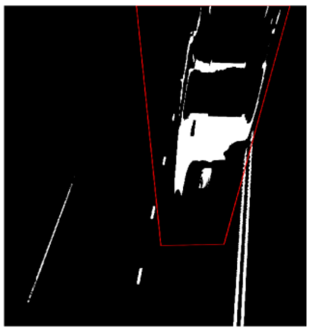


Fig. 6-A. Bird's eye view Colour thresholding with Noise

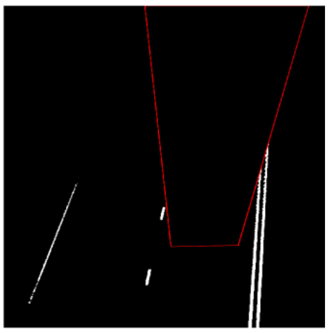


Fig. 6-B. Nullified ROI – red bbox

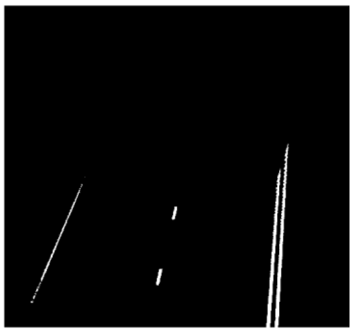


Fig. 6-C. Binary Image after removing Noise due to Yellow-coloured car

The concentration of true values shows the region where the lane lines are present. These concentration values were obtained from the x, y values of the Binary Image's matrix. The distribution of true (Boolean) values across the horizontal axis of the image were obtained as shown in Figure 7.

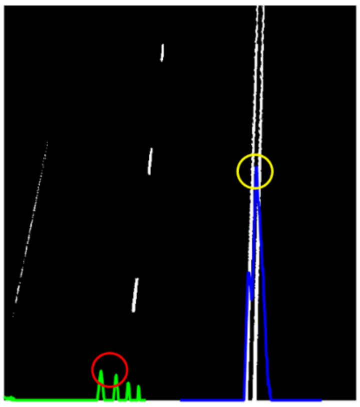


Fig. 7. True value concentration along horizontal axis of Binary image

The lane boundaries in this region are then obtained by Sliding Windows Technique [8,9], (Figure 8.) where a Region of Interest (ROI) of a fixed size slides up along the lane boundaries. The ROI The centroid of this boundary area is used to Slide the window such that it retrieves the boundary points along the lane.

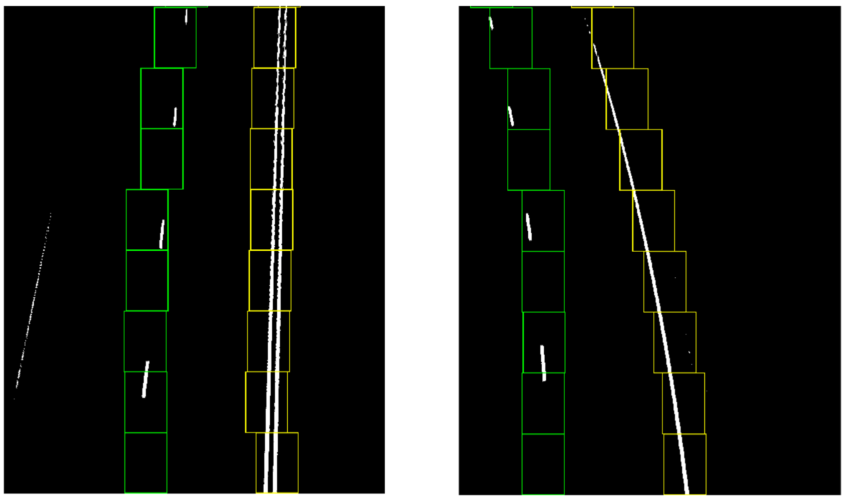


Fig. 8. Lane boundaries tracked using Sliding-Window Technique (both straight and curved lanes)

These boundary points [x y] are extrapolated for ‘y’ till Image height, using the lane’s curve/line equation obtained (curve-fitting) as show in Figure 8-B., which is then transformed to normal image coordinates by multiplying the Inverse of Transformation matrix used while transforming image to Bird’s eye view. The curve is then plotted on the Original Image as show in Figure 9.

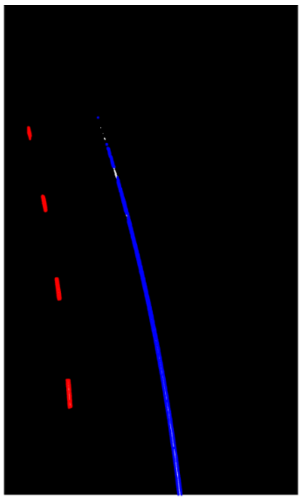


Fig. 8-A. Lane boundaries collected [x, y] points

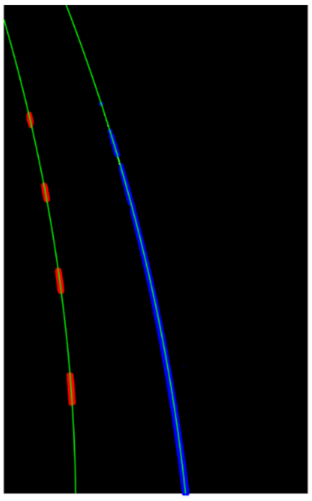


Fig. 8-B. Curve fitting based on [x y] points obtained.

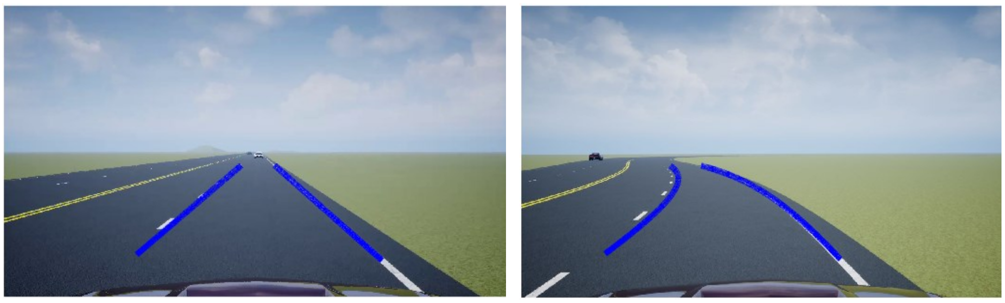


Fig. 9. Lane Markings in Original Image data

5 Sensor Fusion Algorithm:

The bounding boxes generated by the ACF detector are utilised to locate the Most Important Object (lead vehicle); however, the detector could still occasionally provide several bounding boxes for the same vehicle because of multiple detections. Non-maximal suppression was used to address this issue, where overlapping bounding boxes were deleted depending on their confidence scores. For the purpose of preventing false detections, the boundary boxes with scores under 90% were eliminated.

Now, to find the lead car (MIO) the x-y points of the lane detected are compared to that with the boundary boxes of the cars detected. The detection that is within the detected lane's limits (inside the lane) is considered the lead car. This way the lead vehicle is precisely determined using the sensor data even though different vehicles are detected in parallel lanes, because of leveraging Lane detection results for determining MIO [10].

These x, y points of the MIO obtained from post-processing object detection data are then compared with detection measurements of Radar data. Given the spatial coordinates of the lead vehicle detection from vision data, the aim is to find the radar data point that is spatially closest to the vision data point. This association is based on the premise that the corresponding detections across modalities are spatially proximal. The radar data point with the least distance to the selected vision data point is considered as the corresponding radar detection for the selected vehicle. The unique ID of the Radar detection's point data which is in close proximity to the MIO Point data is used for Speed controller inputs.

Through this straightforward approach, the data from radar and vision sensors is bridged, paving the way for a more comprehensive object recognition and tracking mechanism [11].

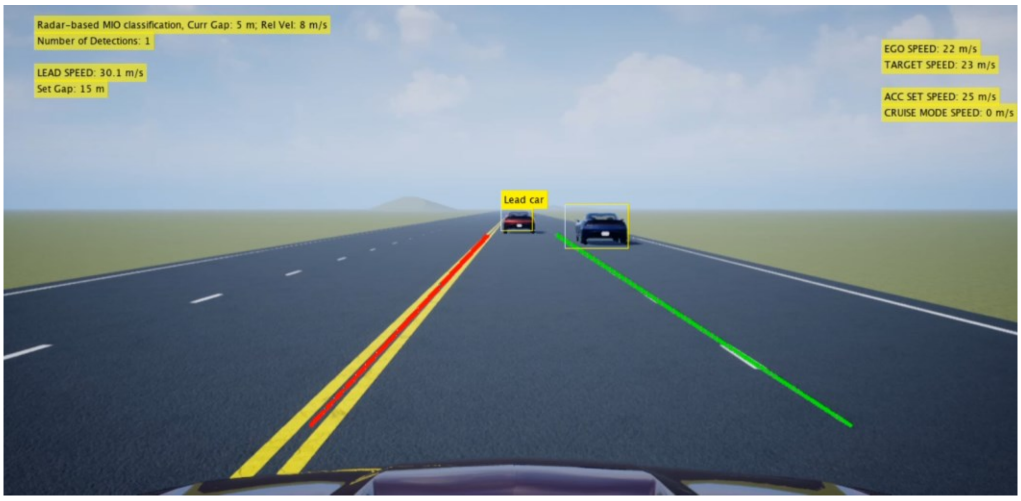


Fig. 10. Sensor fusion algorithm results.

6 Ego Vehicle Target Speed Determination:

The speed of the Ego vehicle is adaptively controlled based on both the user-defined speed settings and the speed of a lead vehicle detected. If the lead vehicle's speed is lower than the user-defined speed, the Ego vehicle's target speed is adjusted to match the lead vehicle's speed, maintaining safe following distance.

However, if the lead car speed is less than user-defined speed and the current gap is less than the set gap the system automatically reduces the speed of the vehicle to increase the gap between vehicles thus preventing any sudden collision that may occur. Conversely, if the lead vehicle's speed is higher, the Ego vehicle's target speed remains at the user-defined speed, ensuring user comfort and safety.

MATLAB’s Stateflow® was used to implement the logic and control of Ego vehicle’s longitudinal position. Primarily two major states were formulated,

1. State 1: Lead car detected
2. State 2: Lead car not detected (no lead car)

When a Lead car is detected, the Algorithm computes the target speed and safe distance provided by the user to the system to confirm safe distance is maintained between ego and lead vehicle. This is done by constantly comparing the gap and monitoring the speed of the lead vehicle and states of both ego and lead car in a longitudinal axis. The meticulously designed Algorithm dynamically changes between states 1 and 2 as soon as there is a change in gap or lead car speed. If, Lead car is not detected, the ego goes to Cruise speed mode (Speed set by the driver).

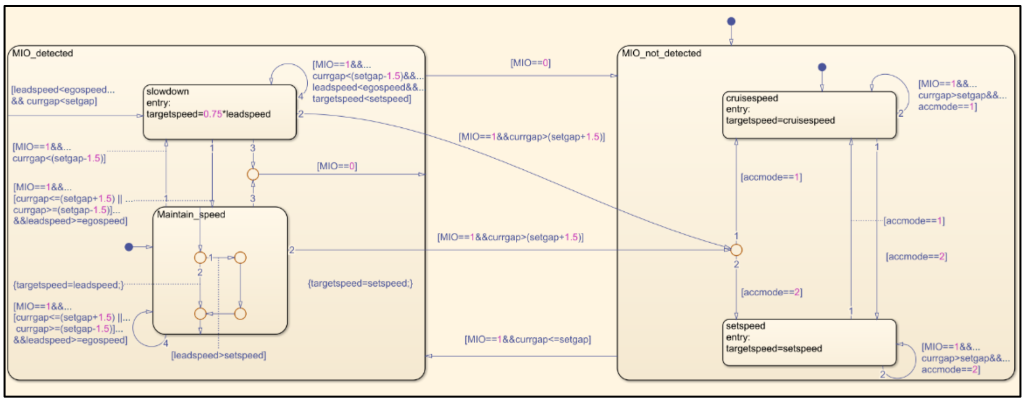

MIO_detected
[leadspeed<egospeed... && currgap<setgap]
slowdown entry: targetspeed=0.75*leadspeed
[MIO==1&&... currgap<(setgap-1.5)&&... leadspeed<egospeed&&... targetspeed<setspeed]
[MIO==1&&... currgap<(setgap-1.5)]
[MIO==0]
[MIO==1&&... [currgap<=(setgap+1.5) || ... currgap>=(setgap-1.5)]... &&leadspeed>=egospeed]
Maintain_speed
{targetspeed=leadspeed;}
{targetspeed=setspeed;}
[MIO==1&&... [currgap<=(setgap+1.5) ||... currgap>=(setgap-1.5)]... &&leadspeed>=egospeed]
[leadspeed>setspeed]
[MIO==0]
[MIO==1&&currgap>(setgap+1.5)]
[MIO==1&&currgap>(setgap+1.5)]
[MIO==1&&currgap<=setgap]
MIO_not_detected
cruisespeed entry: targetspeed=cruisespeed
[MIO==1&&... currgap>setgap&&... accmode==1]
[accmode==1]
[accmode==1]
[accmode==2]
[accmode==2]
setspeed entry: targetspeed=setspeed
[MIO==1&&... currgap>setgap&&... accmode==2]

Fig. 11. Ego vehicle’s speed State flow logic.

7 Simulation environment & Analysis:

The algorithm's reliability and safety were verified within a 3D Unreal Engine simulation environment integrated with Simulink. The trajectory waypoints for both the ego vehicle and other actors, including the lead vehicle, were carefully designed to activate the speed control logic. To ensure the system's capability to make sound decisions and prevent accidents, it was thoroughly tested against a comprehensive range of potential driving scenarios.

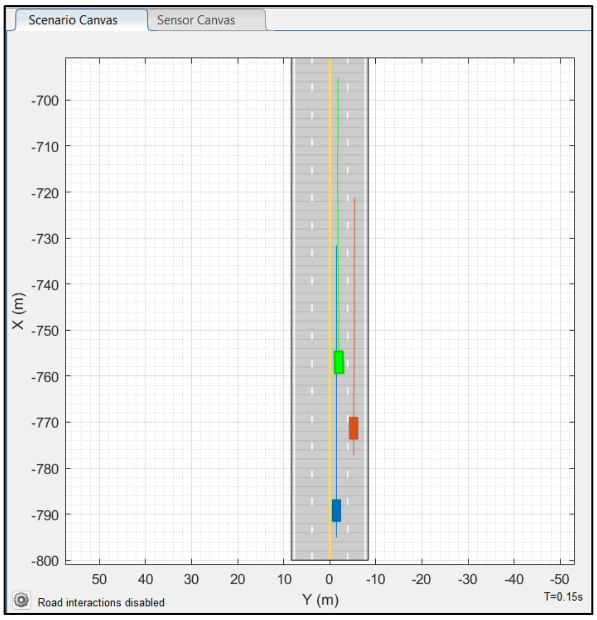
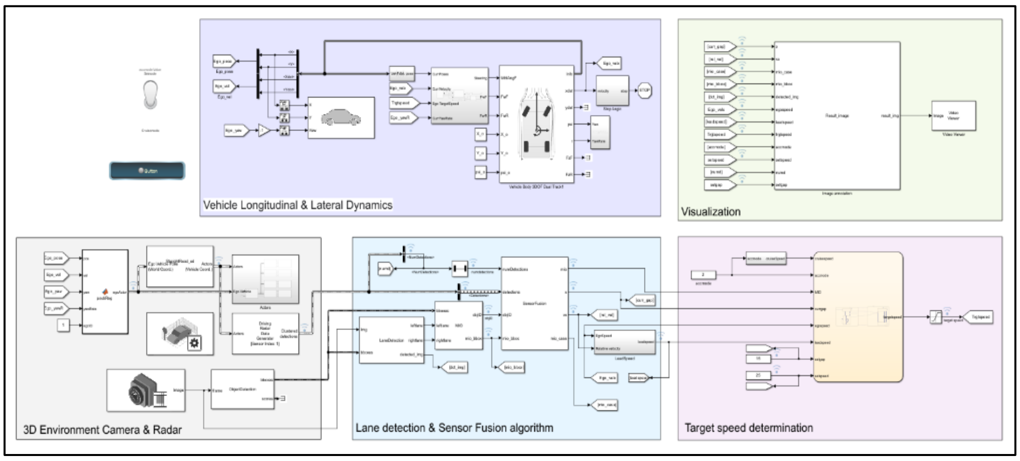


Fig. 12. Defined vehicle trajectories.

A lateral controller was employed to predict the steering angle of the front wheels by considering the ego vehicle's trajectory and speed. This estimated wheel angle was then utilised within a 3D Vehicle Body model with three degrees of freedom to compute the X and Y coordinates, yaw rate and XY velocities of the ego vehicle. This structured data was then used to localise the Ego vehicle in the 3D environment to be able to visualise the Longitudinal and Lateral dynamics of the Ego vehicle [12,13].



Vehicle Longitudinal & Lateral Dynamics
Visualization
3D Environment Camera & Radar
Lane detection & Sensor Fusion algorithm
Target speed determination

Fig. 13. MATLAB ACC Simulink model.

8 Results

To validate the performance of the developed algorithm, a 3D-environment validation was done as shown in Figure 10. Further, to precisely monitor the changes in speed of the Ego vehicle, the speeds of both lead and ego vehicles were logged. From Figure 14, In the Scenario A, where the Relative gap between detected Lead vehicle and Ego vehicle is larger than the set gap (15m), the ego accelerates to its set cruise speed (by driver, 25m/s) as shown in Figures 15 & 16. As the Ego vehicle's speed is higher, the gap starts reducing. In Scenario B, once the relative gap comes below the set gap (15m), the Ego starts slowing down as the Target speed for Ego, is reduced. After some deceleration, the gap starts increasing to a point where the relative gap is more than the required safe distance. Therefore, the Ego goes back into acceleration mode, as per driver's set speed which is in the Scenario C from the graphs.

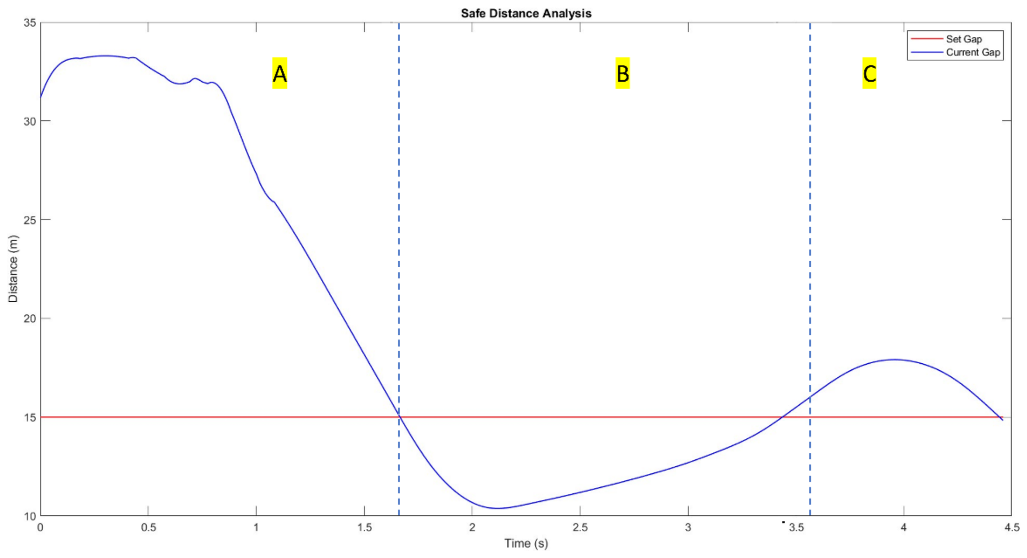


Fig. 14. Set gap & Current gap

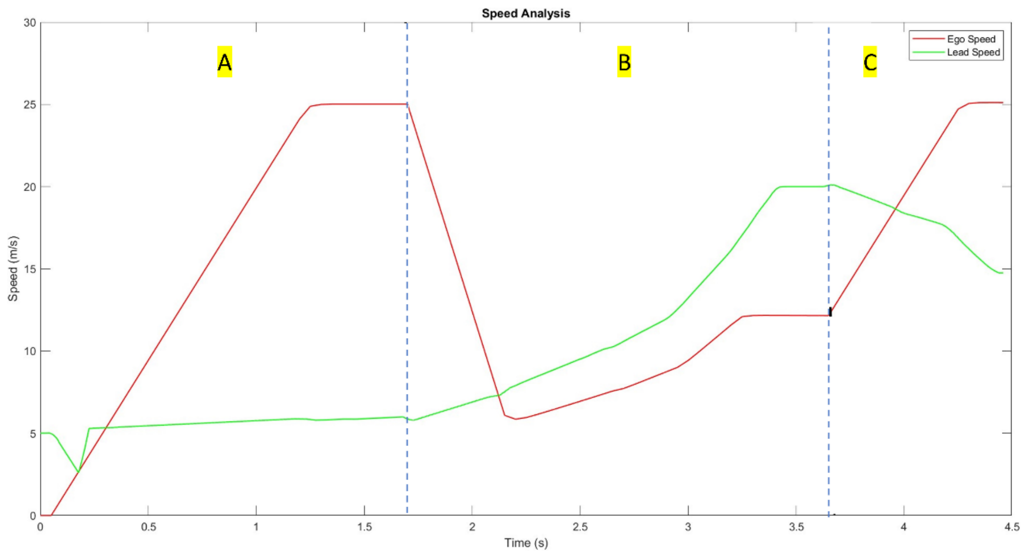


Fig. 15. Ego speed & Lead speed.

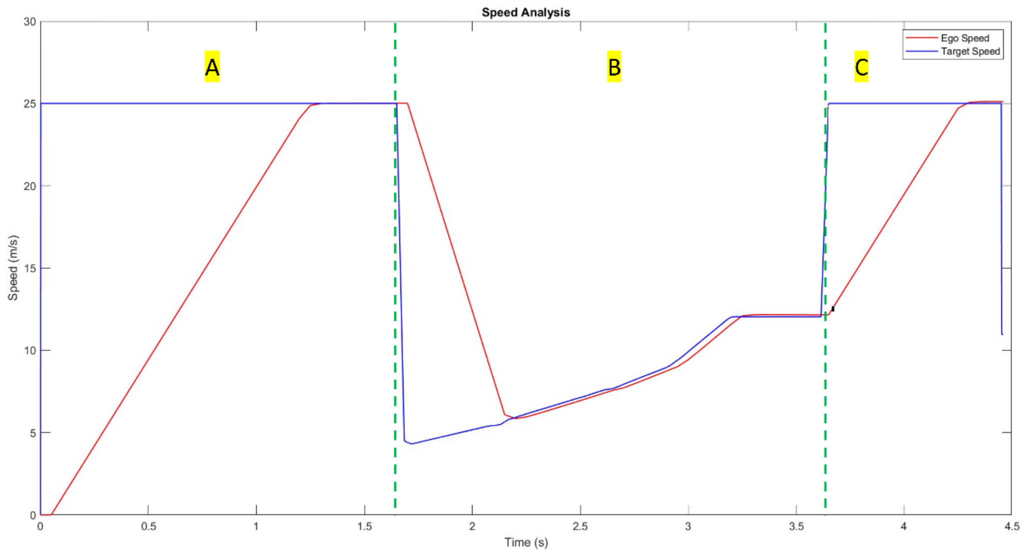


Fig. 16. Ego speed & Target speed.

Conclusion:

An Adaptive cruise control algorithm was modelled and validated through co-simulation of Simulink and 3D unreal Engine in MATLAB. Camera data and Radar detections data were obtained from this Unreal Engine environment in addition, a computer vision-based lane detection method was developed and implemented in the Sensor Fusion algorithm together with Radar detection data. From the camera data, an ACF detector was trained to detect cars. The sensor fusion approach used both vehicle detections from Camera data as well as radar detection data to precisely detect the lead vehicle with which the Ego vehicle's target speed (for speed control) was determined. With this, the sensor fusion algorithm was designed in such a way that it obtains data of the lead car only, the vehicle in front of the Ego vehicle. To validate the algorithm, the Ego vehicle's lateral and longitudinal dynamics were also simulated to ensure realistic performance. Scenarios with different speed and safety gaps (between ego and lead car) were validated. Therefore, a comprehensive model of the Adaptive cruise control algorithm was designed, simulated and validated in this analysis. In future studies, lane detection algorithms based on Neural networks can be used in place of the computer vision-based algorithm used in this paper. Also, a robust controller like the Model-Predictive controller can be used in place of the Longitudinal Stanley controller used in this analysis.

References

[1] Ilic, Velibor, et al. "Development of sensor fusion based ADAS modules in virtual environments." 2018 Zooming Innovation in Consumer Technologies Conference (ZINC). IEEE, 2018.

[2] Y. Ying and O. Odunayo Solomon, "Research on Adaptive Cruise Control Systems and Performance Analysis Using Matlab and Carsim," 2017 5th International Conference on Mechanical, Automotive and Materials Engineering (CMAME), Guangzhou, China, 2017, pp. 249-253, doi: 10.1109/CMAME.2017.8540177.

[3] Nie, Z.; Farzaneh, H. Adaptive Cruise Control for Eco-Driving Based on Model Predictive Control Algorithm. Appl. Sci. 2020, 10, 5271. https://doi.org/10.3390/app10155271.

[4] Yeong, D.J.; Velasco-Hernandez, G.; Barry, J.; Walsh, J. Sensor and Sensor Fusion Technology in Autonomous Vehicles: A Review. Sensors 2021, 21, 2140. https://doi.org/10.3390/s21062140

[5] Kim, Jong Bae. "Efficient vehicle detection and distance estimation based on aggregated channel features and inverse perspective mapping from a single camera." Symmetry 11.10 (2019): 1205.

[6] Du, Hongyi. "Lane line detection and vehicle identification using monocular camera based on matlab." Acad J Comput Inf Sci 4.2 (2021): 66.

[7] Son, Yeongho, Elijah S. Lee, and Dongsuk Kum. "Robust multi-lane detection and tracking using adaptive threshold and lane classification." Machine Vision and Applications 30 (2019): 111-124.

[8] Zhang, Qiang, Jianze Liu, and Xuedong Jiang. 2023. "Lane Detection Algorithm in Curves Based on Multi-Sensor Fusion" Sensors 23, no. 12: 5751. https://doi.org/10.3390/s23125751

[9] Ross Kippenbrock (2017) Finding Lane Lines for Self Driving Cars https://github.com/rosskipp/pydata-berlin-2017

[10] MIoAbdelgawad, Kareem, et al. "A modular architecture of an interactive simulation and training environment for advanced driver assistance systems." Int. J. Adv. Softw. IARIA 8 (2015): 247-261.

[11] Chavez-Garcia, Record, OF. Multiple sensor fusion and classification for moving object detection and tracking. IEEE Trans. Intell. Transp. Syst. 2016, 17, 525–534

[12] Bueno Sánchez, Miguel Ángel. Design and evaluation of virtual environments for testing Advanced Driver Assistance Systems. BS thesis. Universitat Politècnica de Catalunya, 2019.

[13] Kim, Junhyung, and Yonghwan Jeong. "Design of robust path tracking controller using model predictive control based on steady state input." Journal of Mechanical Science and Technology 37.8 (2023): 3877-3886.