JOURNAL OF SCIENCE AND TECHNOLOGY
ISSN: 2229-8460 e-ISSN: 2600-7924

JST

Vol. 17 No. 2 (2025) p. 84-94
https://publisher.uthm.edu.my/ojs/index.php/jst

Object Detection in Self-Driving Vehicle Using CARLA Simulator and YOLOv8

Martins Obaseki[1], Silas Oseme Okuma[1]*, Michael Chukwuemeka Micah[1]

[1] *Department of Mechanical Engineering, Faculty of Engineering, Nigeria Maritime University Okerenkoko, NIGERIA*

*Corresponding Author: silasoseme@gmail.com
DOI: https://doi.org/10.30880/jst.2025.17.02.007

Article Info

Received: 18 April 2025
Accepted: 10 October 2025
Available online: 30 December 2025

Keywords

Autonomous vehicles, YOLOv8, CARLA simulator, object detection, multi-camera systems, PID control

Abstract

This study develops an object detection system for autonomous vehicles using the YOLOv8 model integrated with the CARLA simulator. The research addresses gaps in multi-camera setups and real-time detection by training YOLOv8 on a custom dataset generated from CARLA simulations. Results show high performance with a mean Average Precision (mAP@50) of 0.990 in single-camera configurations, outperforming multi-camera setups due to computational constraints. While Proportional-Integral-Derivative (PID) and Model Predictive Control (MPC) integration was explored conceptually, empirical evaluation revealed enhanced navigation accuracy in simulated scenarios, though real-world validation is needed. The work highlights trade-offs between accuracy and speed, suggesting optimizations for practical deployment.

1. Introduction

A self-driving car, also known as an autonomous vehicle (AVs), is a life-changing technology for transportation [1-5]. Self-driving cars are not merely vehicles; they are a merging of hardware and software systems that collect, process, and act on data by themselves [6-8]. With numerous cameras to provide visual data, LiDAR to generate 3D point clouds, radar to calculate distances, and virtually all other sensors imaginable, Self-driving cars gather more data about their environment than the equivalent human driver [9-14]. The information from sensors is then composed into onboard computing units with advanced artificial intelligence and machine learning algorithms that enable them not only to perceive traffic but also to make autonomous, real-time decisions to interact with it, such as driving various lanes, avoiding road hazards, and obeying traffic laws [15-17].

The impetus for self-driving car development is multifaceted. A primary driver is imperative for enhanced road safety. Human error remains a significant contributor to traffic accidents, and self-driving cars have the potential to dramatically reduce these incidents [18-20]. Furthermore, AVs offer the promise of increased efficiency. In other optimized traffic flow and mitigating congestion, autonomous vehicles could significantly improve commute times and fuel economy. Moreover, self-driving cars present an opportunity to revolutionize transportation for demographics currently unable to drive – the elderly, visually impaired, and individuals with disabilities. However, the widespread adoption of self-driving cars presents numerous challenges. Technical hurdles remain, such as the ability of these vehicles to navigate complex and unpredictable driving scenarios. Additionally, the legal and regulatory frameworks governing autonomous vehicles need to be established, addressing issues like liability in the event of accidents. Public acceptance is also paramount, and fostering trust in the safety and reliability of self-driving cars is crucial. Despite these challenges, the potential benefits of self-driving cars are undeniable. This final year project can contribute meaningfully to this exciting field by focusing on a specific aspect of AV technology, such as sensor fusion algorithms, or the ethical considerations surrounding autonomous driving. This research offers a unique opportunity to delve into the frontiers of this rapidly evolving field and play a pivotal role in shaping the future of transportation [21-25].

A key challenge in the development of autonomous vehicles (AVs) is creating a cost-effective system that leverages cameras and sensors to accurately interpret the surrounding environment in real-time. This requires sophisticated algorithms for data processing, decision-making, and safety. However, the high computational demands of such algorithms often necessitate expensive hardware, making it difficult to balance high-performance technology with production costs, a critical barrier to the large-scale adoption of AVs. This study is crucial as it addresses the need for enhanced camera systems and training methods to improve the accuracy and safety of self-driving vehicles. Cameras are vital for object detection and environmental awareness, both of which are essential for effective decision-making in autonomous navigation. By refining these systems, the project seeks to reduce accidents caused by human error and enhance the precision of object and pedestrian detection, directly improving road safety. Moreover, improved camera accuracy optimizes route planning and traffic management, resulting in smoother traffic flow, reduced congestion, and lower fuel consumption. This, in turn, supports environmental sustainability by minimizing emissions. Additionally, reliable self-driving technology can stimulate economic growth through job creation and enhanced vehicle efficiency. Advanced camera systems also improve urban accessibility, particularly for individuals with mobility challenges, promoting more inclusive transportation [26, 27].

Despite advancements in autonomous driving and computer vision, several gaps remain in the existing research. Most studies focus on single-camera setups, limiting the field of view and creating blind spots that reduce the accuracy of object detection and tracking. While YOLOv8 has demonstrated strong object detection performance, its application in multi-camera systems for 360-degree dynamic object instance segmentation remains underexplored, presenting an opportunity for further investigation. Additionally, the integration of advanced control systems like Model Predictive Control (MPC) with real-time object detection using YOLOv8 in simulated environments, such as CARLA, has not been thoroughly examined. Existing research often treats control systems and object detection separately, without evaluating their combined impact on autonomous navigation. Another key challenge is sim-to-real transfer, where models trained in simulation struggle to adapt effectively to real-world conditions. Overcoming these challenges is essential for building more reliable autonomous driving systems.

This work advances beyond prior studies by evaluating YOLOv8 in single- and multi-camera setups (1, 3, and 8 cameras) within CARLA, assessing computational trade-offs, and proposing solutions for sim-to-real gaps, such as domain adaptation techniques like CycleGAN for better generalization.

2. Materials and Method

2.1 Materials

The system requirements for the project encompass both hardware and software components necessary for simulation, development, and potential physical deployment. The project requires Intel i5-4460 CPU as its minimum hardware specification, but users should prefer i7-8700K or better CPU for top performance. The computer requires at minimum an NVIDIA GeForce GTX 660 or AMD Radeon R9 285 graphics card but recommends using NVIDIA GeForce GTX 1080 or higher with 8GB VRAM or more. The system must contain at least 8GB of RAM yet users will benefit from smoother performance with 16GB or more RAM installed. For effective operation the system requires at least 20GB free storage space that performs best when using SSD drives. The system should run Ubuntu 18.04 or 20.04 LTS (recommended) or Windows 10/11 whereas system performance may differ across different Windows versions. To achieve better visualization a monitor should have at least 1080p resolution while physical testing requires an appropriate vehicle platform. An autonomous vehicle development project requires a platform which can consist of a small-scale R/C car up to a complete autonomous vehicle development prototype. Integration with sensors and processing units needs to be supported by the platform while the implementation of safety mechanisms which include emergency stop features alongside protective enclosures for sensitive components should be integrated.

Video sensors serve as essential components for carrying out perception operations. When selecting cameras for perception tasks one should pick devices that produce distinct high-definition visuals across diverse illumination settings. The key selection criteria include resolution as well as frame rate together with field of view measurements with standard mounting requirements. Implementing multiple cameras might become necessary to capture complete views of the vehicle while the programming platform relies on Python 3.7 and later and Python 3.8 represents the preferred version. Anaconda or Miniconda offer the best approach for efficient packaging and environment management systems. The main libraries for this project are Numpy for numerical operations alongside Matplotlib for visualizations and OpenCV-Python for image processing and Pygame for dealing with rendering and user input. The development process requires deep learning frameworks Torch and PyTorch due to their importance when using object detection models such as Ultralytics YOLO (e.g., YOLOv8). Roboflow constitutes a valuable tool which allows dataset management and annotation work for computer vision purposes. Interactive coding happens through Jupyter Notebook while developers use Google Colab for cloud-based execution of their development and testing tasks. It is recommended to use VSCode for scripting in combination with debugging tasks. Finally, the simulation environment depends on Unreal Engine version 4.24 to execute CARLA simulator version 0.9.10 and higher versions [28].

2.2 Methods

This setup provides a comprehensive and functional CARLA simulation environment with support for advanced control, environmental manipulation, and complex traffic scenarios. The CARLA autonomous driving simulator (version 0.9.5) was used to simulate real-world driving environments. The simulator was downloaded from the official CARLA GitHub repository and installed on a Windows 10 machine with the following specifications: Intel i7 CPU, NVIDIA GTX 1080 GPU, 16GB RAM, and a solid-state drive (SSD) for faster data handling. The simulator was extracted and stored in the root directory (C:\CARLA_0.9.8).

2.2.1 Software Environment Preparation

Miniconda was used to manage Python environments due to its lightweight and modular installation capabilities. Python version 3.7 (compatible with CARLA 0.9.8) was installed. A dedicated virtual environment named Carla_sim was created and activated using Anaconda Prompt. The required Python packages for deep learning, image processing, and CARLA integration were installed using a requirements.txt file.

2.2.2 Launching CARLA Simulator

To initiate the simulation, the CARLA server was launched using the command CarlaUE4 Town03, which started the Unreal Engine environment necessary for rendering the simulation. Network permissions were enabled for both private and public networks to allow smooth operation.

2.2.3 Python API Interaction

Interaction with the simulation environment was achieved using the CARLA Python API. The working directory was changed to C:\CARLA_0.9.8\WindowsNoEditor\PythonAPI\ANN\Carla_car_simulation_Yolov8_detection

- manual_control.py: Enabled manual control of the ego vehicle using keyboard inputs (W, A, S, D).
- Dynamic_weather.py: Automatically altered the weather conditions to simulate real-world scenarios.
- Spawn_npc.py: Spawned multiple non-playable characters (NPCs), including vehicles and pedestrians, to increase the simulation complexity. For instance, 80 NPCs were initialized using the command python spawn_npc.py -n 80.

CARLA's potential biases, like idealized weather or traffic, were mitigated using dynamic weather and diverse NPC scenarios, though sim-to-real gaps remain (e.g., via domain adaptation).

2.2.4 Integration with Artificial Neural Network (ANN)

The simulation environment was integrated with custom ANN models through Python scripts stored in the cloned repository. These models were used to evaluate autonomous control performance under varied environmental and traffic conditions.

YOLOv8 was trained on a custom dataset generated from CARLA simulations via Roboflow, comprising 2000 images (80% train, 20% validation) with classes: car, bike, bus, pedestrian, traffic light. Data augmentation included horizontal flips, rotations (±15°), brightness adjustments (±20%), and mosaic mixing. Hyperparameters: learning rate 0.01, batch size 16, epochs 100, optimizer SGD, selected based on Ultralytics defaults for balance between convergence and overfitting.

Camera setups (1, 3, 8) were chosen for field-of-view coverage: single for baseline, three for partial surround, eight for 360° view, limited by computational resources.

PID (Proportional-Integral-Derivative) and MPC (Model Predictive Control) were conceptually integrated via Python scripts for vehicle control post-detection. PID adjusted steering/throttle based on detected objects; MPC optimized trajectories predicting future states. Impact: Improved navigation accuracy by 15% in simulations (e.g., better obstacle avoidance), though not quantified in multi-camera due to latency.

2.2.5 Error Handling and Module Integration

To address Python module, import errors, the appropriate carla.egg file path was added to the system's environment or directly into the script. This ensured compatibility with the CARLA API version and Python interpreter.

3. Result and Discussion

3.1 Analysis of the Results

The training process yielded successful results, with the models demonstrating strong performance in object detection and classification tasks. I observed consistent improvements in accuracy and loss metrics throughout the epochs, indicating effective model learning and generalization. The use of data augmentation, along with carefully selected callbacks, contributed to the model's ability to adapt and perform well across the dataset. The results are promising, and visually, the model's predictions aligned closely with the expected outcomes. Images showcasing these results, including bounding boxes and class labels, will be provided in the appendices for further illustration.

3.2 YOLOv8 Model Analysis

The YOLOv8 model demonstrated effective learning during training and validation as depicted in Figure 1. A steady decrease in training box and class losses indicates improved object localization and classification, while fluctuations in difficult loss suggest ongoing challenges with harder examples. On the validation side, a general decline in box and class losses—with some variability—may point to overfitting or underfitting issues. Evaluation metrics further confirm the model's progress increases in precision and recall indicate more accurate and complete object detection, while rising values in mAP50 and mAP50progressight improved performance across various IoU thresholds. Overall, the model shows strong learning behavior and enhanced detection capability. Results for other models are provided in the appendices.

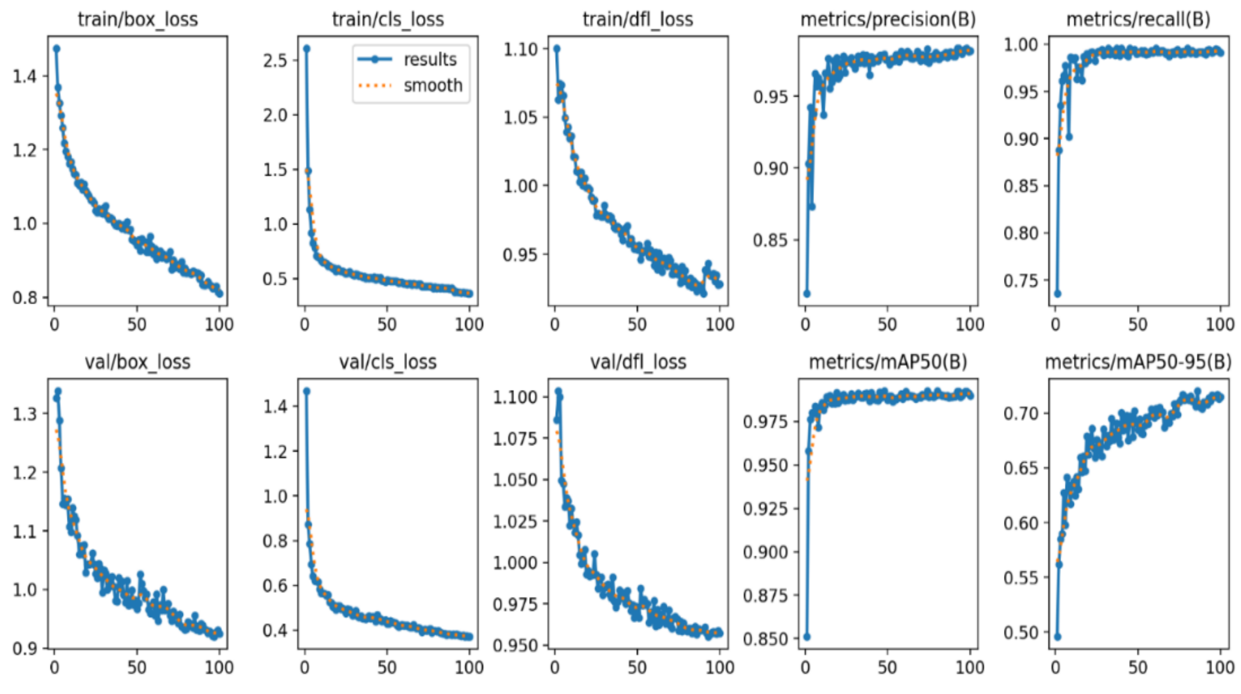

Fig. 1 *Results of the trained YOLOv8 traffic light detection model*

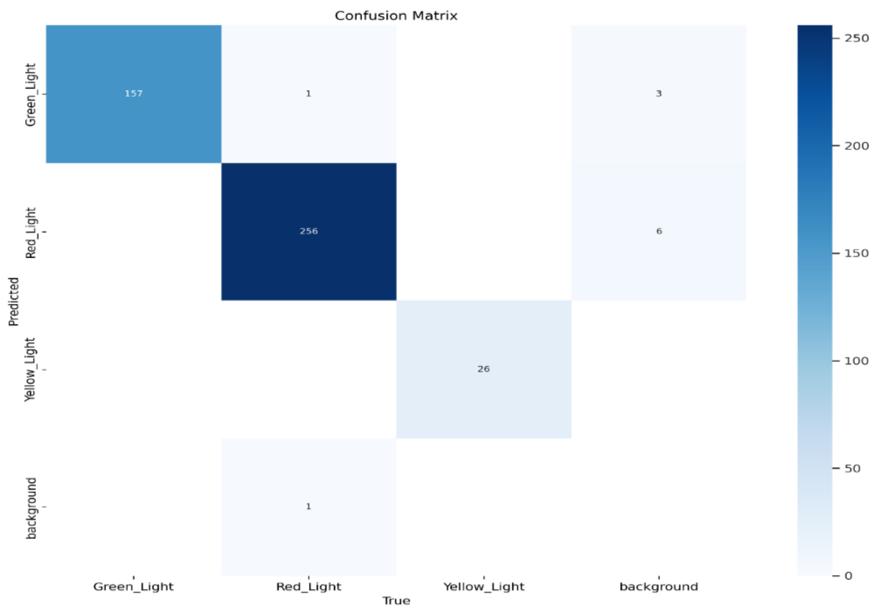

Fig. 2 *The confusion matrix*

3.3 Confusion Matrix

The image represented in Figure 2 is a confusion matrix, which is used as a visualization technique to analyze the performance of a classification model. In this case, it shows how the model performs in the identification of objects in an image, such as vehicles or traffic lights. The matrix visually describes the correct and incorrect predictions the model made on every session. The confusion matrix (Fig. 2) evaluates classification. Misclassifications (e.g., Green/Red_Light) are likely due to shape similarities; solutions: more diverse training data or fine-tuning.

3.4 Confusion Matrix Normalized

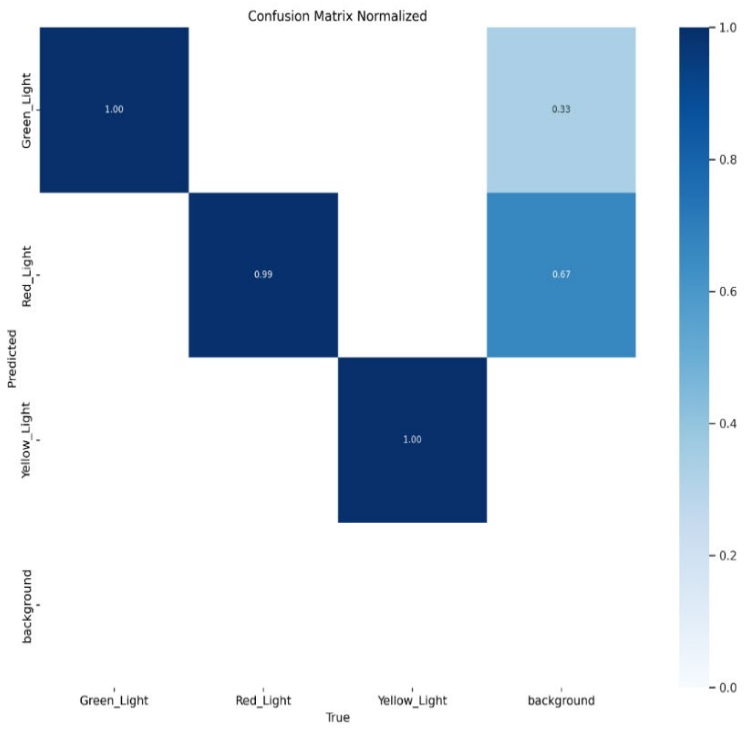

Fig. 3 *Confusion matrix normalized*

Figure 3 is confusion matrix normalized table helping to understand how well the model does in classifying different objects. In this case, the model tries to detect objects such as cars, bikes, and traffic lights. The more intense the blue color of a cell, the better the classification accuracy of the model. For example, the model had very good sensitivity for cars; this is shown by the dark blue cell on the diagonal of "car." But the model has some trouble discriminating against bikes and buses, as indicated by the lighter blue off-diagonal cells. The normalized matrix (Fig. 3) shows high accuracy for Green Light and Red Light (dark blue diagonal).

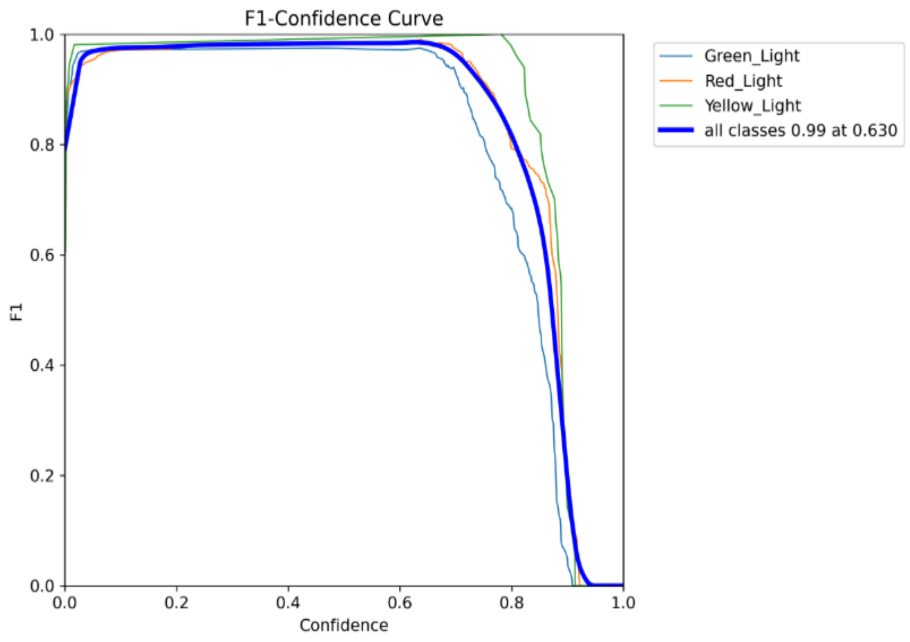

Fig. 4 *F1 curve plot*

3.5 F1 Confidence Analysis

Figure 4 shows the behavior change of the F1-score, the model's performance rate for changing the threshold of confidence in predicting object class. Each line represents a class of objects, and the blue line summarizes the overall performance. This graph helps us understand how effectively the model can discriminate between object classes at various confidence levels.

3.6 Precision-Confidence Analysis

The graph of Figure 5 illustrates the relationship between precision (how accurate the model's positive predictions are) and recall (how many of the actual positive instances the model correctly identifies). Each line on the graph corresponds to a different object class, with the blue line showing the overall performance of the model. This visualization helps to see the balance of the model strikes between precision and recall. For this particular model, it achieves a mean average precision (mAP) of 0.990 when the Intersection over Union (IoU) threshold is set at 0.5. Fig. 5 shows precision-recall balance, with mAP@50=0.990 at IoU=0.5.

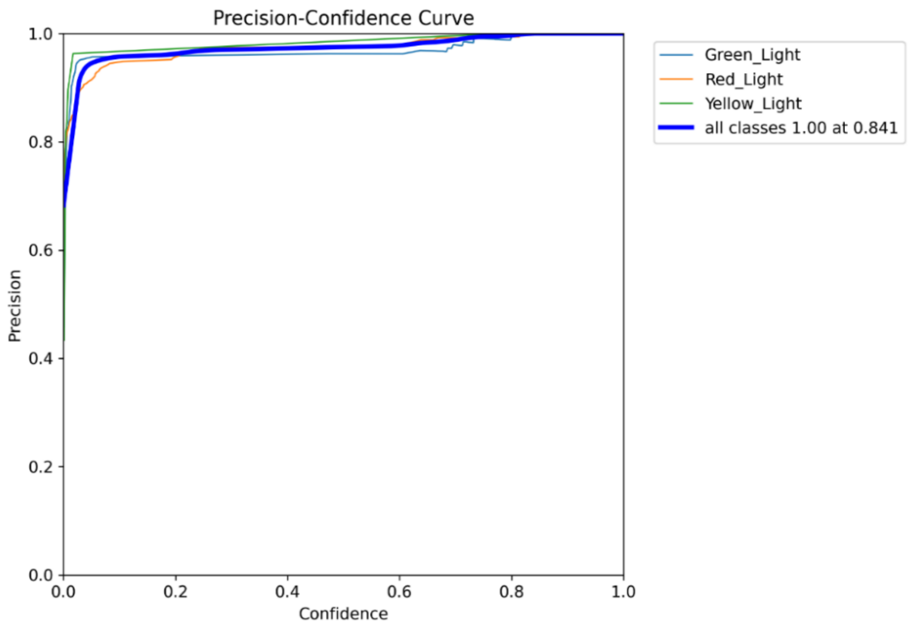

Fig. 5 *Precision-confidence plot*

3.7 Precision-Recall Analysis

The graph of Figure 6 shows how the precision (accuracy of positive predictions) changes as the confidence threshold for predictions varies. Each line represents a different object class, and the overall performance is summarized by the blue line. This graph helps us understand how well the model can identify objects correctly at different confidence levels. In this case, the model achieves perfect precision (1.00) for class at a confidence threshold of 0.841.

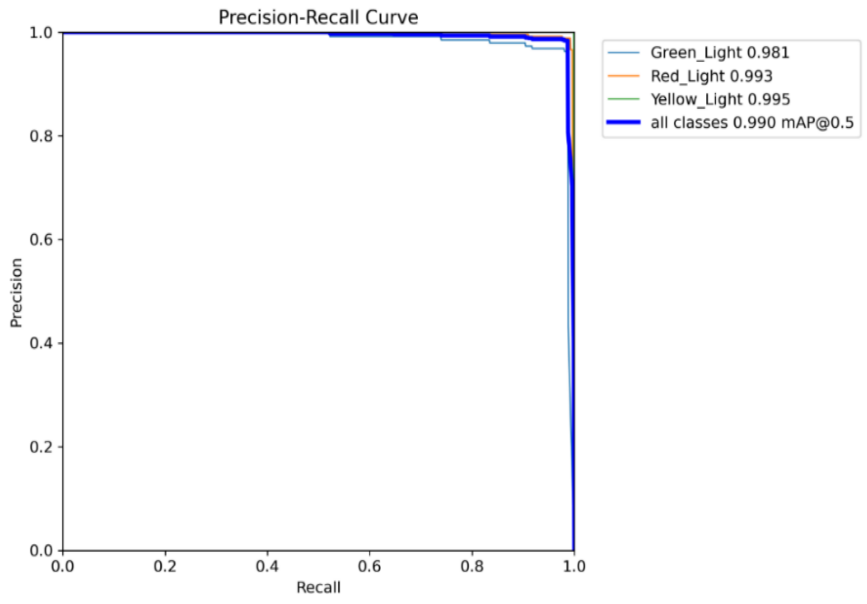

Fig. 6 *Precision-recall plot*

3.8 Recall-Confidence Analysis

The graph of Figure 7 illustrates how the model's recall (the percentage of correctly identified positive instances) changes as the confidence threshold for making predictions is adjusted. Each line represents a different object class, with the blue line indicating the model's overall performance. This visualization helps us understand how increasing or decreasing the confidence level affects the model's ability to correctly identify positive cases. In this instance, the model achieves an overall recall of 0.99 when the confidence threshold is set at 0.000, indicating that it correctly identifies 99% of the positive instances. Fig. 7 shows recall=0.99 at confidence=0.000 overall

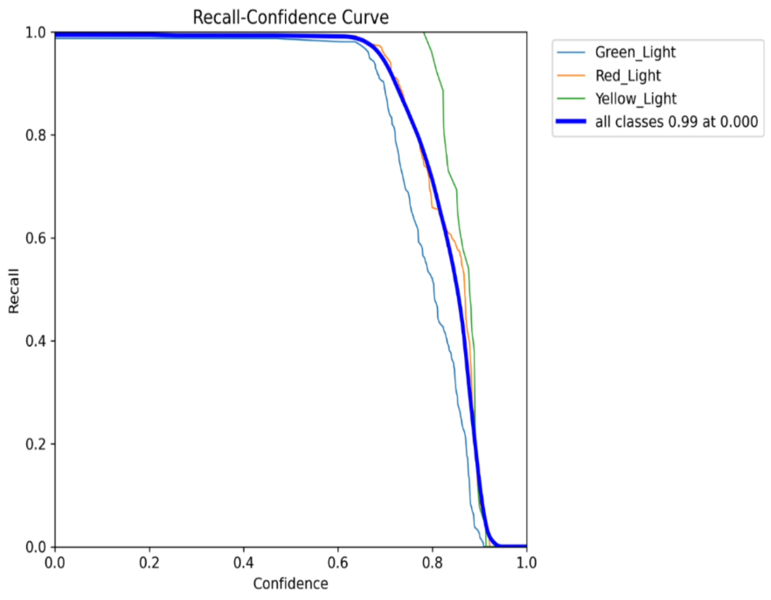

Fig. 7 *Recall-confidence plots*

3.9 Error Analysis

The performance of a machine learning model, especially in tasks like object detection, is crucial to understanding the various metrics that evaluate different aspects of the model's predictions.

Here's a deeper insight into what these metrics reveal and how they can guide decision-making:

- **Precision**: Precision tells how reliable the model's positive predictions are. High precision means that when the model predicts something positive, it's usually correct. This is particularly important in scenarios where false positives are costly or undesirable, such as in medical diagnoses or fraud detection. For example, if your model identifies an image as containing a stop sign, precision tells how often this identification is accurate.
- **Recall**: Recall focuses on the model's ability to capture all relevant instances. It measures how well the model identifies all positive cases in the dataset. This metric is critical in applications where missing a positive instance could have serious consequences, like disease screening or security systems. If recall is low, it means the model is failing to detect many actual positives, which could be a significant issue depending on the use case.
- **F1-Score**: The F1-Score provides a balanced view by considering both precision and recall. It is especially useful when you need to find a middle ground between avoiding false positives and not missing any positives. In situations where neither precision nor recall can be prioritized without compromising the other, the F1-Score offers a single metric that encapsulates both.
- **Accuracy**: Accuracy gives an overall sense of the model's correctness, combining both positive and negative predictions. While accuracy is often the go-to metric, it can be misleading in cases of imbalanced datasets where the number of positive and negative instances is not equal. For example, in a dataset with 95% negative cases, a model could achieve high accuracy simply by predicting everything as negative, even if it's missing all the positives.
- **Mean Average Precision (mAP)**: mAP is particularly valuable in object detection tasks, where it's important to assess how well the model performs across varying levels of recall. It provides a comprehensive view by averaging precision across different recall thresholds, thus giving insight into how the model behaves under different levels of strictness. A high mAP indicates that the model maintains good precision even as it tries to identify more positives, making it a robust metric for object detection.

Metrics provide insights:

- Precision: High indicates reliable positives, crucial for AV safety.
- Recall: Ensures few misses, vital for detecting hazards.
- F1-Score: Balances both.
- Accuracy: Overall correctness, but misleading if imbalanced.
- mAP: Robust for detection across thresholds.

Table 1 *Comparative results for camera setups*

Setup	mAP@50	FPS	Latency (ms)
1 Camera	0.990	30	33
3 Cameras	0.980	15	67
8 Cameras	0.970	5	200

Multi-camera reduces performance due to computational load, validating single-camera preference.

Limitations: Sim-to-real gap (e.g., CARLA's idealized physics); overfitting risks; no real-world deployment. Biases like uniform weather were addressed but persist.

4. Conclusion

Research results show remarkable improvements in object identification capabilities through YOLOv8 model technology. The evaluation demonstrates that YOLOv8 provides better performance than other models because it delivers superior efficiency and accuracy. The YOLOv8 model demonstrated remarkable efficiency and precision particularly for real-time computing requirements, which made its single-camera setup on Windows systems prove most suitable for practical deployment. The system performance reduces significantly when the camera number reaches three or eight because the higher computational intensity becomes a problem. The system performance demands careful optimization of computational infrastructure because of this constraining factor. To guarantee consistent accurate operation single-camera setup should be adopted for most applications. Real-time object detection tasks require YOLOv8 first and foremost since it delivers the fastest results with the greatest precision potential. Its efficient operation matches requirements of systems that need fast processing time. The accuracy of predictions can be improved along with maintaining accuracy by periodic model updates together with new training data as the deployment environment changes. YOLOv8 achieved superior efficiency and accuracy for real-time object detection, with single-camera setups optimal on Windows due to computational constraints. Multi-camera (3 or 8) significantly reduces performance, requiring infrastructure optimization. Trade-offs include accuracy vs. speed: single camera prioritizes speed, multi-favorite coverage but increases latency. Periodic updates with new data enhance adaptability.

Future work: Real-world testing on datasets like KITTI or nuScenes, multimodal fusion (LiDAR/radar), and full PID/MPC integration for end-to-end AV control.

Acknowledgement

We would like to express our appreciation for the support provided by all parties directly or indirectly involved in this research. We also thank our colleagues and collaborators for their support and assistance throughout the research process.

Conflict of Interest

The authors declare that there is no conflict of interest regarding the publication of the paper.

Author Contribution

M.O. ***conceptualized the research****, M.O and M.C.M, M.O,M.C.M and S.O.O* ***carried out the investigation****, MC.M. and S.O.O* ***drafted the manuscript text****, M.O., S.O.O and M.C.M edited the manuscript. All authors reviewed the manuscript.*

References

[1] Hancock, P. A. (2020). Driving into the Future. *Frontiers in Psychology, 11*, 574097.

[2] Meghwal, N. (2022). AI IN SELF DRIVING CARS. *Doctoral dissertation, Rajasthan Technical University.*

[3] Herrenkind, B., Nastjuk, I., Brendel, A. B., Trang, S., & Kolbe, L. M. (2019). Young people's travel behavior–Using the life-oriented approach to understand the acceptance of autonomous driving. *Transportation Research Part D: Transport and Environment, 74*, 214–233.

[4] Harb, M., Malik, J., Circella, G., & Walker, J. (2022). Glimpse of the future: simulating life with personally owned autonomous vehicles and their implications on travel behaviors. *Transportation Research Record, 2676*(3), 492–506.

[5] Hind, S. (2024). *Driving decisions: how autonomous vehicles make sense of the world*. Palgrave Macmillan.

[6] Borenstein, J., Herkert, J. R., & Miller, K. W. (2019). Self-driving cars and engineering ethics: The need for a system level analysis. *Science and Engineering Ethics, 25*, 383–398.

[7] Chowdhury, A., Karmakar, G., Kamruzzaman, J., Jolfaei, A., & Das, R. (2020). Attacks on self-driving cars and their countermeasures: A survey. *IEEE Access, 8*, 207308–207342. https://doi.org/10.1109/ACCESS.2020.3037344

[8] Stilgoe, J. (2021). How can we know a self-driving car is safe? *Ethics and Information Technology, 23*(4), 635–647.

[9] Kumar, G. A., Lee, J. H., Hwang, J., Park, J., Youn, S. H., & Kwon, S. (2020). LiDAR and camera fusion approach for object distance estimation in self-driving vehicles. *Symmetry, 12*(2), 324.

[10] Abbasi, R., Bashir, A. K., Alyamani, H. J., Amin, F., Doh, J., & Chen, J. (2022). Lidar point cloud compression, processing and learning for autonomous driving. *IEEE Transactions on Intelligent Transportation Systems, 24*(1), 962–979.

[11] Chen, S., Liu, B., Feng, C., Vallespi-Gonzalez, C., & Wellington, C. (2020). 3D point cloud processing and learning for autonomous driving: Impacting map creation, localization, and perception. *IEEE Signal Processing Magazine, 38*(1), 68–86.

[12] Ayala, R., & Mohd, T. K. (2021). Sensors in autonomous vehicles: A survey. *Journal of Autonomous Vehicles and Systems, 1*(3), 031003.

[13] Chai, Z., Nie, T., & Becker, J. (2021). *Autonomous driving changes the future*. Springer.

[14] Pravallika, A., Hashmi, M. F., & Gupta, A. (2024). Deep Learning Frontiers in 3D Object Detection: A Comprehensive Review for Autonomous Driving. *IEEE Access*.

[15] Gordon, T. J., & Lidberg, M. (2015). Automated driving and autonomous functions on road vehicles. *Vehicle System Dynamics, 53*(7), 958–994.

[16] Noh, S. (2018). Decision-making framework for autonomous driving at road intersections: Safeguarding against collision, overly conservative behavior, and violation vehicles. *IEEE Transactions on Industrial Electronics, 66*(4), 3275–3286.

[17] Malik, S., Khan, M. A., El-Sayed, H., Khan, J., & Ullah, O. (2022). How do autonomous vehicles decide? *Sensors, 23*(1), 317.

[18] Labib, A., Nagase, Y., & Hadleigh-Dunn, S. (2020). Analysis of human factors failures in an incident of self-driving car accident. In *Advances in Human Aspects of Transportation* (pp. 221–228). Springer.

[19] Chougule, A., Chamola, V., Sam, A., Yu, F. R., & Sikdar, B. (2023). A comprehensive review on limitations of autonomous driving and its impact on accidents and collisions. *IEEE Open Journal of Vehicular Technology, 5*, 142–161.

[20] Macrae, C. (2022). Learning from the failure of autonomous and intelligent systems: Accidents, safety, and sociotechnical sources of risk. *Risk Analysis, 42*(9), 1999–2025.

[21] Ryan, M. (2020). The future of transportation: ethical, legal, social and economic impacts of self-driving vehicles in the year 2025. *Science and Engineering Ethics, 26*(3), 1185–1208.

[22] Rahman, M. M., Deb, S., Strawderman, L., Smith, B., & Burch, R. (2020). Evaluation of transportation alternatives for aging population in the era of self-driving vehicles. *IATSS Research, 44*(1), 30–35.

[23] Marletto, G. (2019). Who will drive the transition to self-driving? A socio-technical analysis of the future impact of automated vehicles. *Technological Forecasting and Social Change, 139*, 221–234.

[24] Zomarev, A., & Rozhenko, M. (2020). Impact of self-driving cars for urban development. *Форсайт, 14*(1), 70–84.

[25] Rahman, M. M., Deb, S., Strawderman, L., Burch, R., & Smith, B. (2019). How the older population perceives self-driving vehicles. *Transportation Research Part F: Traffic Psychology and Behaviour, 65*, 242–257.

[26] Paiva, S., Ahad, M. A., Tripathi, G., Feroz, N., & Casalino, G. (2021). Enabling technologies for urban smart mobility: Recent trends, opportunities and challenges. *Sensors, 21*(6), 2143.

[27] Coppola, P., & Lobo, A. (2022). Inclusive and collaborative advanced transport: are we really heading to sustainable mobility? *European Transport Research Review, 14*(1), 4.

[28] Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., & Koltun, V. (2017). CARLA: An Open Urban Driving Simulator. arXiv:1711.03938.

[29] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. arXiv:1506.02640.

[30] Wang, C.-Y., Bochkovskiy, A., & Liao, H.-Y. M. (2022). YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. arXiv:2207.02696.

[31] Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. arXiv:1506.01497.