Transport and Communications Science Journal

REAL-TIME MULTI-SENSOR FUSION FOR OBJECT DETECTION AND LOCALIZATION IN SELF-DRIVING CARS: A CARLA SIMULATION

Trung Thi Hoa Trang Nguyen[1,2], Thanh Toan Dao[2,*], Thanh Binh Ngo[2]

[1]Hanoi College of High Technology, Nhue Giang Street, Tay Mo Ward, Nam Tu Liem District, Hanoi, Vietnam

[2]University of Transport and Communications, No 3 Cau Giay Street, Hanoi, Vietnam

ARTICLE INFO

TYPE: Research Article
Received: 10/12/2024
Revised: 06/01/2025
Accepted: 10/01/2025
Published online: 15/01/2025
https://doi.org/10.47869/tcsj.76.1.6

[*] *Corresponding author*
Email: daotoan@utc.edu.vn; Tel: +84979379099

Abstract. Research on integrating camera and LiDAR in self-driving car systems has important scientific significance in the context of developing 4.0 technology and applying artificial intelligence. The research contributes to improving the accuracy in recognizing and locating objects in complex environments. This is an important foundation for further research on optimizing response time and improving the safety of self-driving systems. This study proposes a real-time multi-sensor data fusion method, termed "Multi-Layer Fusion," for object detection and localization in autonomous vehicles. The fusion process leverages pixel-level and feature-level integration, ensuring seamless data synchronization and robust performance under adverse conditions. Experiments conducted on the CARLA simulator. The results show that the method significantly improves environmental perception and object localization, achieving a mean detection accuracy of 95% and a mean distance error of 0.54 meters across diverse conditions, with real-time performance at 30 FPS. These results demonstrate its robustness in both ideal and adverse scenarios.

Keywords: Camera-LiDAR Fusion, Real-Time, Object Detection, Object Localization, Self-Driving Cars, CARLA.

1. INTRODUCTION

In recent years, the advancement of autonomous driving technology has been driven by the integration of sensor-based systems, with LiDAR emerging as a key player for environmental perception [1-3].

Prominent companies like Waymo (Google) and Tesla have been pioneers in sensor integration for autonomous systems. Waymo's system combines camera and LiDAR data to enhance object detection and real-time decision-making, while Tesla focuses on multi-camera setups complemented by LiDAR for precise object localization [4,5].

In our previous research [6], a single-beam LiDAR-based navigation system was developed, utilizing neural networks to perform obstacle avoidance and ensure vehicle navigation in controlled environments. This approach demonstrated notable effectiveness in detecting and avoiding obstacles using cost-efficient and computationally lightweight setups. However, it faced limitations when applied to real-world, complex environments, where dynamic obstacles, varied lighting conditions, and intricate spatial layouts demand more sophisticated perception capabilities.

To overcome these challenges, the integration of camera data with LiDAR offers a compelling solution. Cameras excel in capturing high-resolution images, enabling advanced object recognition and classification through visual processing. By fusing the spatial data from LiDAR with the detailed imagery from cameras, the system can leverage the complementary strengths of both sensors, significantly improving object detection, localization, and overall environmental understanding.

This paper proposes a multi-layer data fusion framework that combines multi-beam LiDAR and camera data for autonomous navigation in complex environments. The approach addresses the limitations of single-sensor systems and enhances real-time decision-making by incorporating:

- Layer 1 - Pixel-Level Fusion – Mapping LiDAR point clouds onto the camera’s image plane to achieve spatial alignment between depth and visual information.
- Layer 2 - Feature-Level Fusion – Extracting and merging features from both sensors to generate a unified dataset for robust decision-making processes.

The system integrates YOLOv8 for real-time object detection using camera data, while the LiDAR sensor provides precise distance and angle measurements. The proposed fusion method synchronizes data in both spatial and temporal domains, ensuring seamless integration and accurate environmental perception.

Simulations conducted in the CARLA simulator validate the effectiveness of the proposed method under varying environmental conditions, including complex traffic scenarios and dynamic lighting. This enhanced framework not only demonstrates improved accuracy and responsiveness but also holds significant potential for real-world applications, particularly in the domains of dynamic obstacle avoidance and autonomous driving safety.

This research contributes to the development of safer and more efficient autonomous systems capable of operating in real-world environments.

2. RELATED WORK

The integration of multi-sensor data, particularly from camera and LiDAR, has been a critical focus in autonomous vehicle research. Existing methods can be categorized into three primary approaches: early fusion, late fusion, and hybrid fusion. Each approach has its advantages and limitations, which have been extensively studied in the literature.

- Early Fusion: Raw sensor data is combined at the initial processing stages. For example, X.Chen et al. (2017) proposed MV3D, which projects LiDAR point clouds onto a bird's-eye view and integrates them with camera image features for improved perception accuracy [7]. Similarly, J.Ku et al. (2017) introduced AVOD, which fuses raw sensor feature maps for robust 3D object detection [8]. These methods leverage raw data but face challenges due to their high computational cost and stringent requirements for precise sensor synchronization. Moreover, early fusion can struggle in dynamic environments where real-time processing is critical.

- Late Fusion: Sensor data is processed independently and merged during decision-making. Geiger et al., (2012) demonstrated late fusion's efficiency in reducing computational overhead using the KITTI dataset [9]. However, this approach has limitations in unstructured environments where the independence of sensor processing can lead to loss of spatial and temporal alignment. Late fusion also suffers from difficulties in capturing inter-sensor dependencies, which are crucial for complex perception tasks.

- Hybrid Fusion: Recent research emphasizes real-time multi-sensor integration using hybrid approaches. For instance, Yin et al. (2020) integrated YOLOv4 with LiDAR for real-time obstacle detection, achieving fast and accurate results [10]. Hybrid methods aim to balance the strengths of early and late fusion but often require complex architectures and calibration.

In Vietnam, research remains in its early stages, with significant contributions from universities and companies like Phenikaa Group, which integrates LiDAR and camera data for urban autonomous vehicles. These efforts underscore the growing focus on sensor fusion for enhanced perception and localization.

While the aforementioned methods have advanced object detection and localization, they exhibit several limitations:

1) Environmental Conditions: Camera-based systems often fail in adverse conditions like poor lighting, rain, or fog, while LiDAR systems struggle with highly reflective or absorbent surfaces.
2) Real-Time Processing: Computational efficiency remains a significant challenge, especially for methods relying on early fusion due to the volume of raw data.
3) Robustness: Many methods assume ideal conditions for both sensors, which limits their effectiveness in real-world scenarios with dynamic obstacles and diverse environmental factors.

To address these limitations, this paper proposes a multi-layer fusion approach that integrates pixel-level and feature-level data from both sensors. By combining the high-resolution imagery of cameras with the accurate spatial data of LiDAR, the proposed method ensures robust perception across various lighting and weather conditions. Additionally, real-time synchronization techniques mitigate latency issues, making the system practical for dynamic environments.

3. PROBLEM FORMULATION

3.1. Proposed method and data fusion process

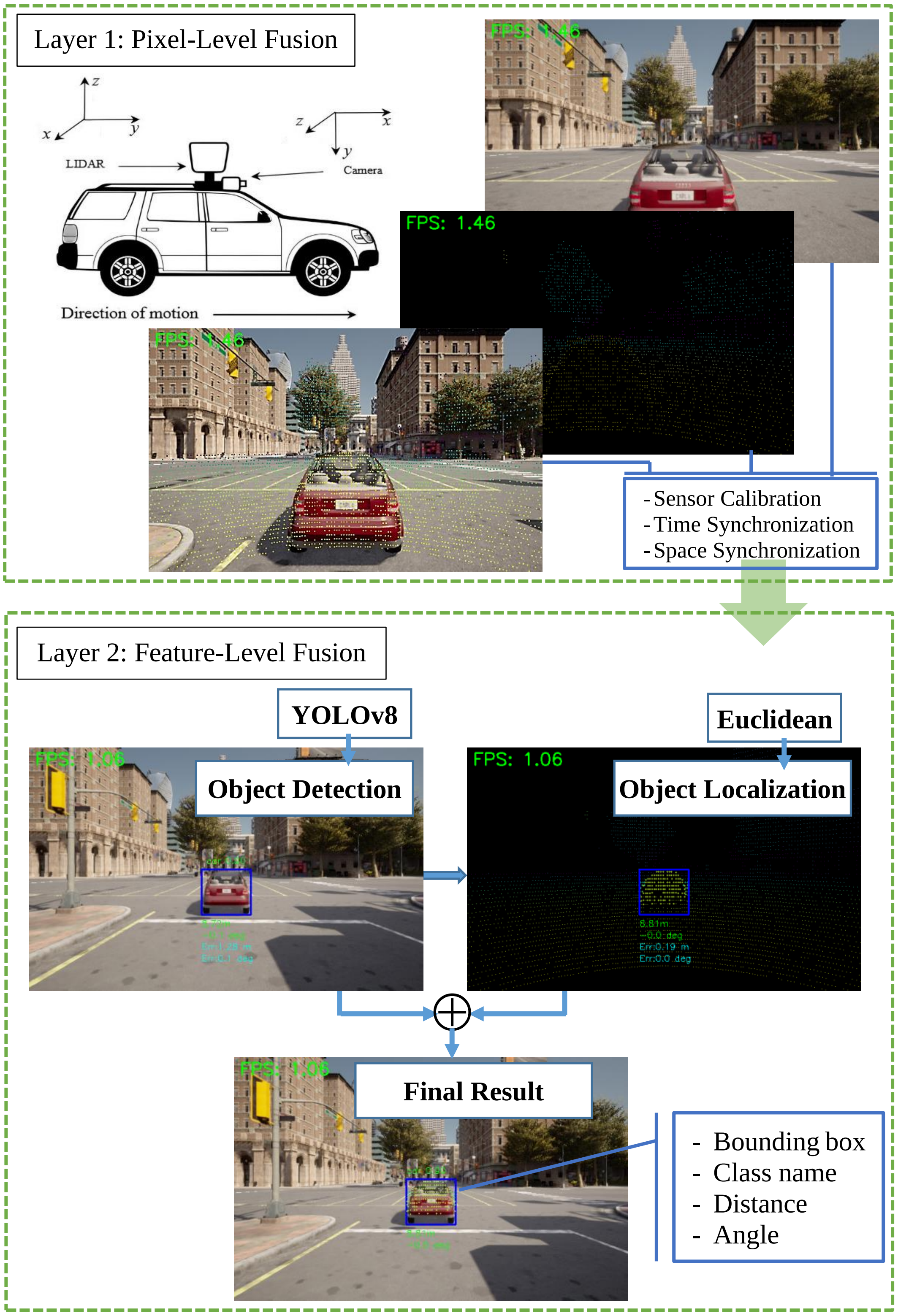

Figure 1. An illustration of the overall framework.

This section introduces a novel "Multi-layer fusion" method, which operates across two layers: Pixel-level fusion and Feature-level fusion. The proposed approach optimizes the advantages of each fusion method, enabling faster and more accurate system responses. An overview of the proposed model is depicted in Figure 1.

- Layer 1: Pixel-Level Fusion
  - Input: pixel data from camera and point cloud data from LiDAR
  - Output: pixel coordinates
- Layer 2: Feature-Level Fusion
  - Input: pixel coordinates
  - Output: location and distance of object in 3D space

Here is the explanation of the mathematical details for the Multi-Layer Fusion method in Pixel-Level Fusion and Feature-Level Fusion:

Algorithm Name: Multi-Layer Fusion for Camera-LiDAR Integration

Input:

- Camera data (I_{camera}): 2D image with pixel intensity values
- LiDAR data (P_{LiDAR}): 3D point clound $\{(x_i, y_i, z_i)\}_{i=1}^{N}$, where N is the number of LiDAR points.
- Camera Parameters: Intrinsic matrix K and extrinsic matrix [R t] for transforming and aligning LiDAR data to the camera's coordinate frame.

Output:

- Fused image I_{fused}: A pixel-level combined representation of color and depth.
- Object-level features F: A set of features F={F_1, F_2 ,…, F_n}, where F_i includes 2D detection and 3D localization for each detected object.

# Pixel-level Fusion

1. Project the LiDAR point cloud P_{LiDAR} onto the 2D camera plane using perspective projection:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \cdot \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (1)$$

Where:

- K: Camera intrinsic matrix (focal length, principal point).
- $\begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}$: Homogeneous Extrinsic matrix (rotation and translation) aligning LiDAR to the camera frame.

2. Assign depth z values from P_{LiDAR} to corresponding pixels in I_{camera}.
3. Generate a fused image:

$$I_{fused}(u,v) = \alpha \cdot \frac{I_{camera}(u,v)}{I_{max}} + \beta \cdot \frac{D(u,v) - D_{min}}{D_{max} - D_{min}} \quad (2)$$

Where:

$I_{camera}(u,v)$: Intensity (color) value from the camera, I_{max} is the maximum value of the color intensity (usually 255 in 8-bit images).

- $D(u,v)$: Depth value from LiDAR at pixel (u, v), where D_{min} and D_{max} are the smallest and largest depth values in the entire image, respectively.
- α, β: Weights to balance contributions from both sources, with α + β=1. Choosing α and β is important to ensure that the data from the camera and LiDAR are properly combined. It is possible to determine the α and β values automatically based on environmental conditions by building a system that classifies environmental conditions (e.g., light, fog, rain)

based on sensors. Then map the environmental conditions to the appropriate α and β values. Finally, evaluate the results on real-world data sets to fine-tune the parameters.

# **Feature-level Fusion**

4. Use YOLOv8 on I_{fused} to detect objects $\{O_1, O_2, \dots, O_n\}$ in the image extract 2D bounding boxes, each detected object O_i has:

$$B_i = (x_{min}, y_{min}, x_{max}, y_{max}, c) \tag{3}$$

Where:

- $(x_{min}, y_{min}, x_{max}, y_{max})$: Bounding box coordinates in the 2D image.
- c: Object class.

5. Localize each object in 3D space using LiDAR points within the detected bounding box, LiDAR points are used to compute the Euclidean distance and localize objects in 3D space. For each object O_i, the center point is computed as:

$$C_i = \frac{1}{N}\sum_{k=1}^{N} P_k \tag{4}$$

Where:

- $P_k=(x_k, y_k, z_k)$: LiDAR points within the 2D bounding box of O_i.
- N: Number of LiDAR points associated with the object.

6. Fuse 2D detection (B_i) and 3D localization (C_i) into object-level features:

$$F_i = [B_i, C_i] \tag{5}$$

Combine features from multiple objects:

$$F = \sum_{i=1}^{n} w_i \cdot F_i \tag{6}$$

Where: w_i is the weighting factor, which could depend on the reliability of the camera or LiDAR data.

In the context of this method, spatial and temporal calibration (Calibration and Synchronization) are considered the factors that need to be addressed first, because they lay the foundation for all subsequent merging steps. If the calibration process is not accurate, the entire merging result will be affected.

3.2. Problem

3.2.1. Sensor Calibration, Time Synchronization, and Spatial Alignment

Calibration and synchronization of the camera and LiDAR are essential steps in integrating these two sensors to ensure accurate perception and localization in autonomous systems. This process involves internal calibration, external calibration, and both spatial and temporal synchronization. Among these, temporal synchronization poses significant challenges, particularly in real-time applications, due to differences in data acquisition rates and processing times. One of the primary challenges in integrating camera and LiDAR data is the mismatch between the frame rate (FPS) of the camera and the scanning speed of the LiDAR:

- Cameras typically operate at fixed frame rates, such as 30 or 60 FPS.
- LiDAR sensors, on the other hand, generate point clouds at rates determined by their rotation speed (e.g., 10 Hz to 20 Hz).

This difference creates a temporal misalignment, where camera frames and LiDAR scans do not correspond to the same moment in time, particularly in dynamic environments with fastmoving objects. Real-time systems exacerbate this challenge due to the need for immediate data fusion and decision-making. There are 2 methods to do this:

- Hardware synchronization:
  - Use a common clock signal (e.g., GPS PPS).
  - Uniform timestamps.
- Software synchronization:
  - Apply interpolation to match data between time frames.
  - Use the Kalman Filter algorithm to predict the next state of the object and match the time.

Autonomous vehicles operating at high speeds or drones navigating in dynamic environments, hardware synchronization is generally the better option due to its accuracy and reliability.

1) Internal calibration: Adjust the internal parameters of each camera and LiDAR sensor, including focal length, principal point, and distortion. The result is the internal matrix K.

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{7}$$

In which:

- f_x and f_y are the focal lengths of the camera (in pixels) respectively on the x-axis and y-axis, usually given:

$$f_x = f_y = focal = \frac{\text{image\_w}}{2 \times tan\left(\frac{fov \times \pi}{360}\right)} \tag{8}$$

Where:

+ focal is the focal length of the camera.
+ image_w is the width of the image.
+ fov is the field of view of the camera measured in degrees.

- c_x and c_y are the optical centers (principal points), usually given: c_x= image width /2 and c_y= image height /2
- s is the skew factor, s=0 if the image axis is perpendicular
- The values 0 and 1 in the remaining positions are to ensure the matrix has the correct geometry.

2) External Correction: Determines the position and angle of the camera and LiDAR relative to each other in 3D space, using a rotation matrix and a displacement vector. The resulting matrix is:

$$T_{LiDAR\to Camera} = \begin{bmatrix} R_{LiDAR\to Camera} & t_{LiDAR\to Camera} \\ 0 & 1 \end{bmatrix} \tag{9}$$

3) Synchronize time using the Uniform timestamps method: The goal is to calculate the time interval between consecutive frames and use it to estimate the system's frame rate (Frames Per Second, FPS). This approach is fundamental for synchronizing time-sensitive operations like sensor data fusion or real-time processing.

Algorithm Name: Uniform timestamps

```
1. Initialize the System Time:
    • Record the initial time $T_{last}$ when the program starts or the loop begins.
    • This serves as a reference for measuring elapsed time during subsequent frames.
                  $T_{last}$ = current_time()                                (10)
```

2. Loop Through Frames: For each frame i (where i = 1, 2, … , N): Record the current time $T_{current}$ at the start of the frame.

3. Calculate the time difference (delta time) ΔT between the current and last recorded time:

$$\Delta T = T_{current} - T_{last} \tag{11}$$

4. Estimate FPS:

- If ΔT>0, calculate the FPS as:

$$FPS = \frac{1}{\Delta T} \tag{12}$$

- If ΔT≤0, set FPS=0 to avoid division by zero.

5. Update Time Reference: Update T_{last} for the next iteration: $T_{last} = T_{current}$

6. Optional Frame Rate Control:

- To maintain a consistent frame rate, introduce a delay:

$$delay\_time = max\left(0, \frac{1}{target\_FPS} - \Delta T\right) \tag{13}$$

- This ensures that each frame processing time aligns with the desired target FPS.

4) Once the data is synchronized in time, the next step is spatial synchronization. Each sensor operates on its own coordinate system, so a transformation process is needed to bring all the data to the same common reference coordinate system. Specifically:

- Camera: Operates in a 2D image coordinate system.
- LiDAR: Operates in a 3D or 2D coordinate system (depending on the type of LiDAR used).

The transformation between coordinate systems involves projecting the point clouds from the LiDAR onto the camera image space through a transformation matrix. This ensures that the points in the LiDAR space are accurately mapped onto the objects detected in the camera image (see Figure 1).

Algorithm Name: Spatial synchronization

1. Collect LiDAR Point Cloud:

- Assume the LiDAR point cloud P_{LiDAR} is a matrix of size N×3, where N is the number of LiDAR points, and each row represents a point with coordinates (x,y,z).
- Convert the points into homogeneous coordinates by adding a fourth dimension with value 1:

$$P_{LiDAR}^{\text{homogeneous}} = \begin{bmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ x_N & y_N & z_N & 1 \end{bmatrix} \tag{14}$$

2. Transform LiDAR Points to World Coordinates

- Apply a transformation matrix $T_{LiDAR \rightarrow World}$ to convert LiDAR points from the LiDAR coordinate system to the world coordinate system:

$$P_{World}^{\text{homogeneous}} = T_{LiDAR \rightarrow World} \times P_{LiDAR}^{\text{homogeneous}} \tag{15}$$

3. Transform World Coordinates to Camera Coordinates

- Use the transformation matrix $T_{World \rightarrow Camera}$ to map the points from the world coordinate system to the camera coordinate system:

$$P_{Camera}^{\text{homogeneous}} = T_{World \to Camera} \times P_{World}^{\text{homogeneous}} \tag{16}$$

- Extract the first three rows (ignoring the homogeneous coordinate) to obtain the 3D points in the camera coordinate system:

$$P_{Camera} = \begin{bmatrix} y_{Camera} \\ -z_{Camera} \\ x_{Camera} \end{bmatrix} = \begin{bmatrix} y_{Camera,1} & y_{Camera,2} & \cdots & y_{Camera,N} \\ -z_{Camera,1} & -z_{Camera,2} & \cdots & -z_{Camera,N} \\ x_{Camera,1} & x_{Camera,2} & \cdots & x_{Camera,N} \end{bmatrix} \tag{17}$$

Because:

- The x coordinate of LiDAR (outward direction) is the z coordinate of the camera.
- The y coordinate of LiDAR (left to right direction) is the x coordinate of the camera.
- The z coordinate of LiDAR (bottom to top direction) is opposite to the y coordinate of the camera so it will be −z.

4. Convert Camera Coordinates to Pixel Coordinates

- Map the camera coordinates to the 2D image plane using the intrinsic matrix K:

$$P_{Pixel} = KP_{Camera} \tag{18}$$

Extract the pixel coordinates (u, v) as:

$$u = \frac{f_x \cdot y_{Camera}}{x_{Camera}} + c_x \ ; \ \ v = \frac{-f_y \cdot z_{Camera}}{x_{Camera}} + c_y \tag{19}$$

3.2.2. Object detection and localization

1) Object detection and recognition in images is an important step in identifying important objects around the vehicle. Advanced algorithms include YOLO, SSD, and Faster R-CNN. In our experiment, the YOLO model was chosen because of its combination of fast processing speed, high accuracy, ability to detect multiple objects in a single pass, and ability to perform well in real-world conditions. Especially in systems that require object recognition with fast and accurate response such as autopilot systems, YOLO is an ideal choice.
2) Determining the distance and angle between a detected object and the vehicle using LiDAR involves leveraging geometric properties and sensor data. Assuming the object is defined by coordinates (x_{obj}, y_{obj}, z_{obj}) and the LiDAR cloud point has coordinates at (x_{veh}, y_{veh}, z_{vel}), the Euclidean distance d from the LiDAR to the object will be calculated by the formula:

$$d = \sqrt{\left(x_{\text{obj}} - x_{\text{veh}}\right)^2 + \left(y_{\text{obj}} - y_{\text{veh}}\right)^2 + \left(z_{\text{obj}} - z_{veh}\right)^2} \tag{20}$$

For 360° LiDAR (full vehicle scan), the azimuth angle can be calculated by considering the LiDAR beams in the xy plane (horizontal plane):

$$\theta = arctan\left(\frac{y}{x}\right)\left(\frac{180}{\pi}\right) \tag{21}$$

Where: x and y are the coordinates of the LiDAR point in the xy plane.

4. EXPERIMENTAL RESULT AND DISCUSSION

Simulation tools like CARLA and Apollo are widely used for testing multi-sensor methods. Dosovitskiy et al. (2017) introduced CARLA as an open-source platform simulating realistic lighting, weather, and traffic scenarios [11,12].

In this section, we detail the experimental results conducted in the CARLA simulation environment. CARLA was chosen for its robust capabilities in replicating real-world urban and highway scenarios, allowing for comprehensive testing under diverse environmental and traffic conditions. The experiments were designed to evaluate the performance of the integrated camera-LiDAR system in detecting and localizing objects under varying lighting and weather conditions.

4.1. Introduction to CARLA

CARLA (Car Learning to Act) is an open-source simulation platform specifically designed for autonomous vehicle research. It provides:

- Realistic Urban Models: Predefined towns with configurable road layouts, buildings, and vegetation.
- Dynamic Traffic Management: Integration of vehicles and pedestrians with realistic physics and behaviors.
- Weather and Lighting Control: Flexible configuration of lighting conditions, time of day, and weather (e.g., clear skies, rain, fog).
- Sensor Simulation: Support for various sensors, including RGB cameras, LiDAR, RADAR, and GNSS, with customizable parameters.

The platform's flexibility and realism make it a powerful tool for validating autonomous driving systems before deployment in real-world scenarios. Figure 2 is a map and scene of town number 10 in CARLA.

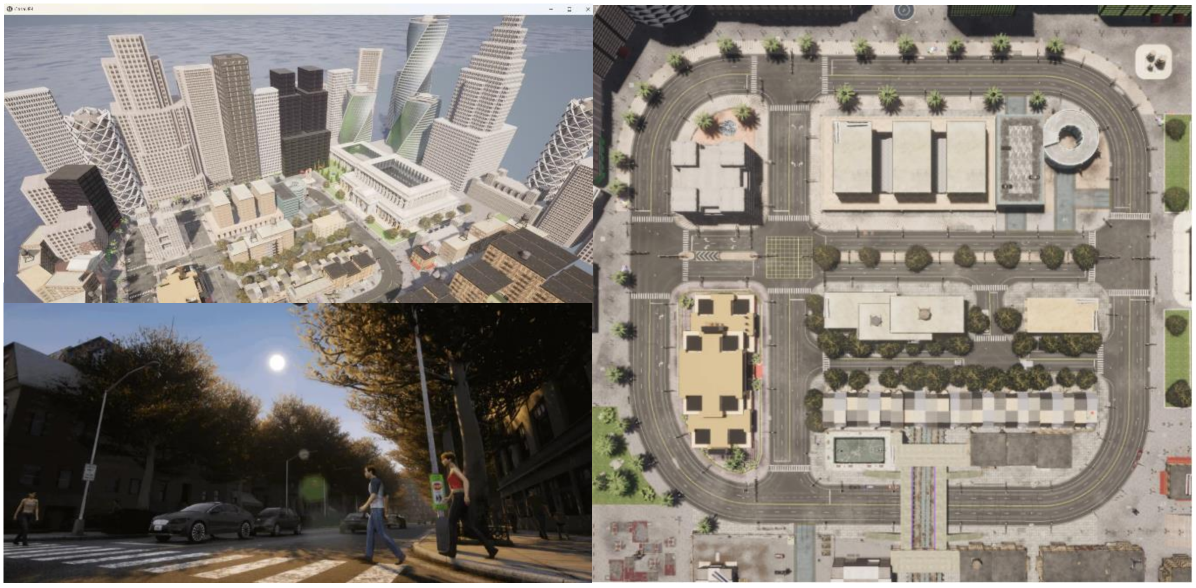

Figure 2. Some scenes (left two) and town map 10 (right) in CARLA.

4.2. Scenario Building

To have more complex scenarios for testing, we choose and set some parameters for the environment as follows:

- Town selection: CARLA provides 12 towns with different terrain and weather characteristics.
  - world = client.load_world('Town10') #change town with this command
- Customize time and weather:
  - weather.sun_altitude_angle = -90.0 # -90: night, 0: sunrise/sunset, 90: noon
  - weather.cloudiness = 0.0 # Cloudiness level (0.0 is no clouds)

- weather.precipitation = 0.0 # Rainfall (0.0: no rain, 50.0: moderate rain, 100.0: heavy rain)
- weather.fog_density = 0.0 # Fog density (0.0 is no fog)

- Create vehicles and traffic participants to diversify the situation. Change the number of people and vehicles participating in traffic in function def parse_args()

4.3. Sensor Configuration

The vehicle was equipped with a camera and a multi-beam LiDAR, configured as follows:

1) *Camera Parameters*:
   - Image Resolution: 680×420 pixels.
   - Field of View (fov): 90^o, covering a wide horizontal field for comprehensive scene capture.
   - Placement: Mounted at the front of the vehicle for forward-looking perception.
2) *LiDAR Parameters*:
   - Field of View:
     - Upper Field of View (upper_fov): 35^o
     - Lower Field of View (lower_fov): −25^o
   - Number of Channels: 64, simulating a high-resolution 3D LiDAR.
   - Maximum Range: 100 meters, allowing for detection of distant objects.
   - Placement: Positioned on the roof of the vehicle, providing a 360° view around the vehicle.

$$\text{Angle between channels} = \frac{35^0 - (-25^o)}{64} \approx 0.94^o \qquad (22)$$

This means that each channel will represent an angle of approximately 0.94° in the scan space.

4.4. Building a simulation program

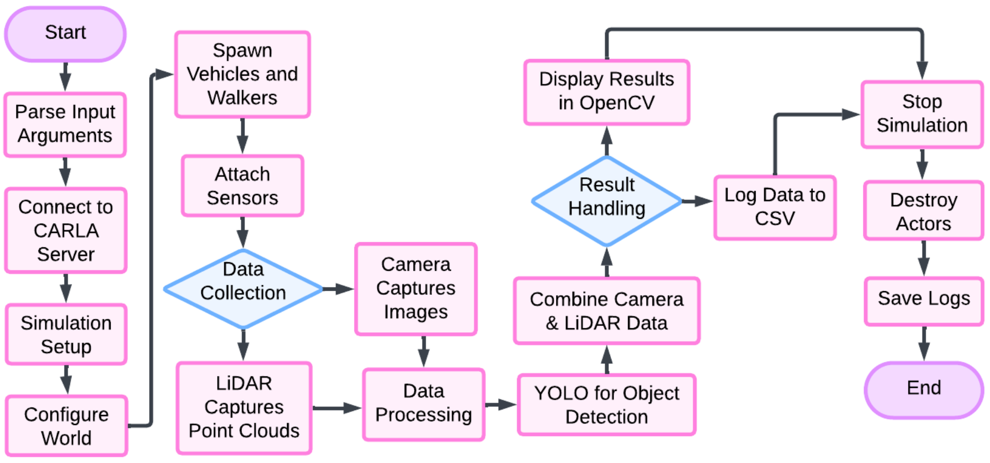

Figure 3. Functional block diagram of the programs.

Figure 3 shows illustrate the workflow of a CARLA simulation system that integrates LiDAR and camera sensors for data collection, processing, and analysis. The process consists of several stages:

1) Initialization: The simulation begins with parsing input arguments, connecting to the CARLA server, setting up the simulation, and configuring the environment (e.g., weather, map).
2) Vehicle and Sensor Deployment: Vehicles and pedestrians are spawned in the simulation, and sensors such as LiDAR and cameras are attached to the vehicles.
3) Data Collection:
   - LiDAR sensors capture point cloud data.
   - Cameras capture images of the environment.
   - The collected data is processed, and YOLO is used for object detection.
4) Data Visualization and Logging: The processed LiDAR and camera data are combined and displayed using OpenCV. The results are logged into CSV files for further analysis.
5) Simulation Termination: After data logging, the simulation stops. All actors (vehicles and pedestrians) are destroyed, and log files are saved for post-simulation evaluation.

This workflow demonstrates an efficient approach to leveraging sensor data in autonomous vehicle simulation using CARLA.

4.5. Simulation results

First, we set up a vehicle in a fixed position to calibrate the sensors and check the spatial and temporal data synchronization. We also checked the distance and angle of the object measurement using the camera and LiDAR (see Figures 4 and 5).

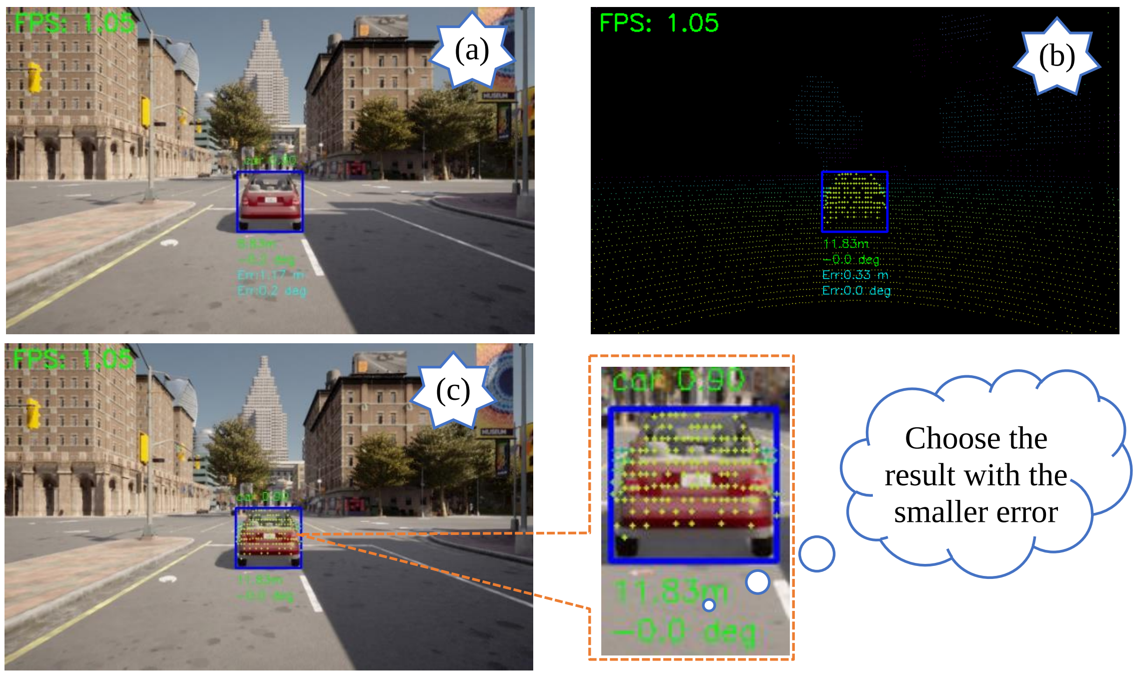

Figure 4. Calibration, time, and space synchronization results: (a) object detection using camera, (b) distance and angle measurement using LiDAR, (c) Integration of data from camera and LiDAR.

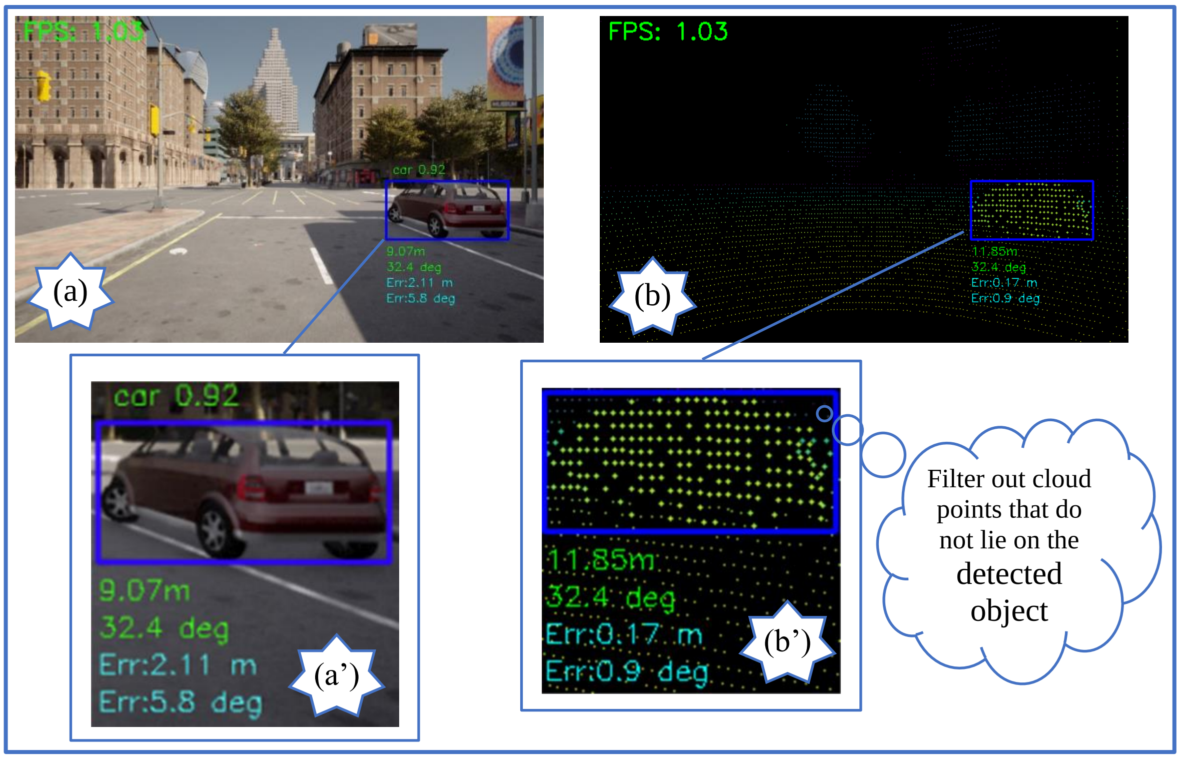

Figure 5. Comparison of distance and angle measurements using camera (left image) and LiDAR (right image).

The results show that the distance measurements obtained from the camera have a larger error and the error is affected by the width and height ratio of the bounding box see (Figures 4a and 5a), while LiDAR provides highly accurate distance measurements regardless of the location of the object. However, the camera has the advantage of detecting object types and providing information about the color and type of the object, which LiDAR cannot do. Therefore, combining data from both sensors will enhance the vehicle's perception of the surrounding environment, thereby improving the accuracy of navigation decision-making. For multi-beam LiDAR, noise handling, and accurate object localization are quite complicated and consume a lot of computer resources. As observed in Figure 5b', we used the method of filtering cloud points according to the minimum height threshold to remove cloud points on the ground and filtering according to the intensity of the response to remove points that are too far away and not on the object (these points are usually located in the two upper corners of the bounding box). The application of the integrated data is as follows:

1) Object Tracking: Use angular data to classify objects within the vehicle's field of view into left, right, or forward zones.

2) Path Planning: Combine distance and angle data to adjust the vehicle's trajectory and avoid obstacles.

3) Targeted Object Interaction: Allocate processing resources to objects within a critical angular range (e.g., the forward-driving cone).

We evaluated the proposed method for detecting and locating objects (humans and vehicles), under various traffic and lighting conditions. The results are presented in Figure 6 and Table 1. The results show that the proposed method achieves high performance in the test scenarios.

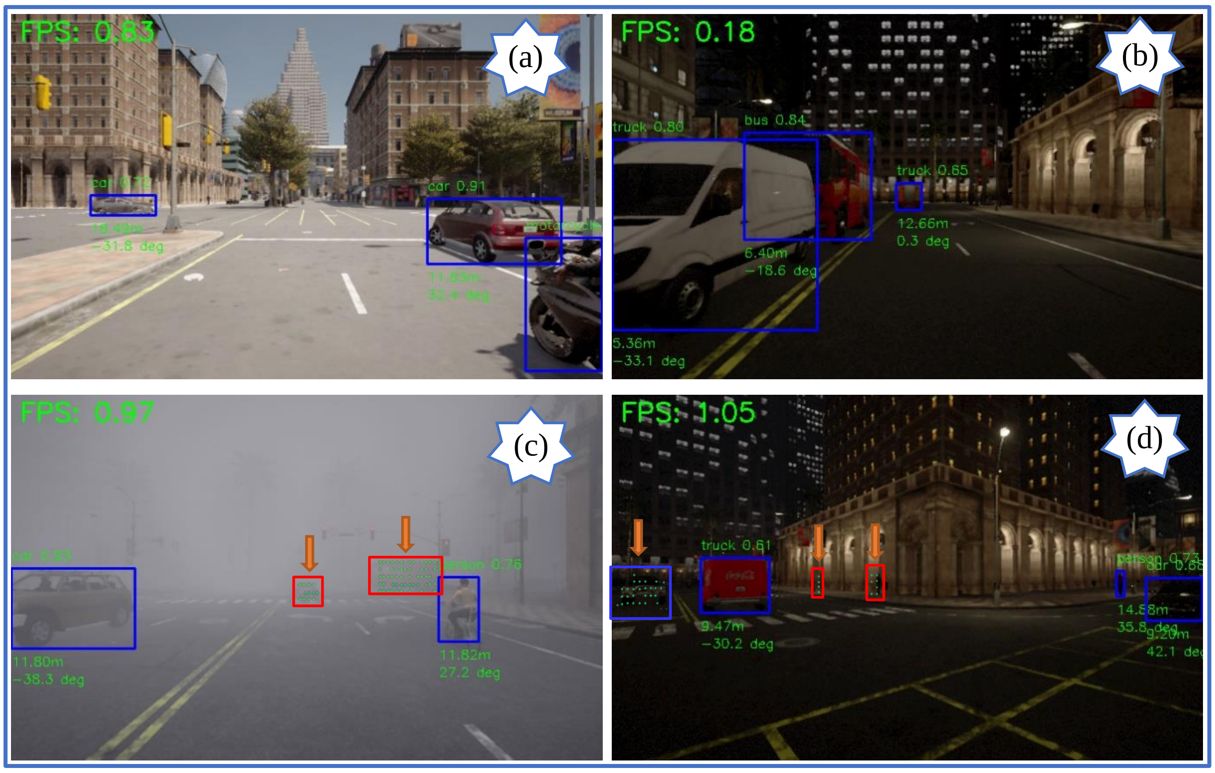

Figure 6. Simulation results under different weather and lighting conditions.

Table 1. Number of detected and located objects in scenarios (confidence>0.6).

Number of detected objects/Number of real objects	Daytime, good light and weather	At night, no rain, no fog	During the day, no rain, dense fog	At night, heavy rain, low fog density
Camera	3/3	3/3	2/4	3/5
Camera-LiDAR With α and β	3/3 α=0.8, β=0.2	3/3 α=0.5, β=0.5	4/4 α=0.3, β=0.7	5/5 α=0.4, β=0.6
Mean error (distance)	0.53516553	0.536226235	0.546190559	0.558860899

With the values of α and β optimized for each condition, we will always be able to take advantage of the strengths of both data sources. This allows the system to maintain performance even if one of the two sources is degraded or lacking data. Additionally, if we want to improve further, we can apply methods to dynamically adjust α and β over time, based on the detection of environmental conditions from the sensors.

The results show that the proposed method significantly improves the ability of autonomous vehicles to perceive their surroundings in low light and adverse weather conditions.

5. CONCLUSION

This study demonstrates the potential of integrating camera-LiDAR data to enhance environmental perception for autonomous vehicles. The integrated system, evaluated on the CARLA simulation platform, demonstrated robust performance across various scenarios:

1) In ideal conditions (daytime, clear weather), the system detected 100% of objects (3/3) with a mean distance error of 0.53 meters.
2) Under challenging conditions (nighttime, heavy rain, low fog density), the system detected 5/5 objects with a mean error of 0.56 meters, outperforming camera-only systems.
3) Across all scenarios, the system maintained real-time performance at 30 FPS, showcasing its practical potential for real-world applications."

Future work will focus on adapting this method to cost-efficient setups, such as replacing multi-beam LiDAR with single-beam alternatives, while preserving detection and localization accuracy.

ACKNOWLEDGEMENT

This research was funded by MOET under grant number B2023-GHA-02.

REFERENCES

[1]. Allied Market Research, Autonomous Vehicle Market by Level of Automation, Component, Application, and Region: Global Opportunity Analysis and Industry Forecast, (2021) 2025–2035. https://www.alliedmarketresearch.com/autonomous-vehicle-market.

[2]. S. Grigorescu, B. Trasnea, T. Cocias, G. Macesanu, A survey of deep learning techniques for autonomous driving, IEEE Transactions on Intelligent Transportation Systems, 21 (2020) 909-922. https://ieeexplore.ieee.org/document/8793489

[3]. Y. Kim, J. Kim, Multi-Object Tracking with Camera-LiDAR Fusion for Autonomous Vehicles: A Survey, IEEE Access, 10 (2022) 38819-38835. https://ieeexplore.ieee.org/document/10591139

[4]. Waymo, Our Journey, Waymo Official Website. https://waymo.com

[5]. Tesla, Inc., Autopilot, Tesla Official Website. https://www.tesla.com/autopilot

[6]. Trung Thi Hoa Trang Nguyen, Thanh Toan Dao, Thanh Binh Ngo, Vu Anh Phi, Self-Driving Car Navigation with Single-Beam LiDAR and Neural Networks Using JavaScript, IEEE Access, (2024). https://doi.org/10.1109/ACCESS.2024.3511572

[7]. X. Chen et al., Multi-view 3D Object Detection Network for Autonomous Driving, IEEE CVPR, (2017). https://doi.org/10.1109/CVPR.2017.691

[8]. J. Ku et al., Joint 3D proposal generation and object detection from view aggregation, IEEE IROS, (2017). https://doi.org/10.48550/arXiv.1712.02294

[9]. A. Geiger, P. Lenz, R. Urtasun, Are we ready for autonomous driving? The KITTI vision benchmark suite, IEEE CVPR, (2012). https://doi.org/10.1109/CVPR.2012.6248074

[10]. A. Bochkovskiy et al., YOLOv4: Optimal Speed and Accuracy of Object Detection, (2020).https://doi.org/10.48550/arXiv.2004.10934

[11]. A. Dosovitskiy et al., CARLA: An Open Urban Driving Simulator, arXiv, (2017). https://arxiv.org/abs/1711.03938

[12]. Carla, The CARLA Autonomous Driving Challenge, (2023). https://leaderboard.carla.org/challenge/