Politecnico Di Torino

Master’s degree in Mechanical Engineering

Master’s Degree Thesis

Safety evaluation of critical maneuvers in simulated road traffic scenarios.

Supervisor

Prof. Francesco Paolo Deflorio

Co-Supervisors

Matteo Ferraro

Giuseppe Calcagno

Candidate

Muhammad Abdullah

Academic Year 2025 – 2026

April 2026

Abstract

The safety of critical maneuvers of autonomous vehicles is a principal aspect of autonomous vehicle validation, assessed within the simulation environment of CARLA (Car Learning to Act). The simulation environment features a cluster configuration of twelve sensors installed on the ego car achieving Level 3 conditional autonomous control on the figure eight highway of the Town 04 environment: along with 20 background cars (controlled traffic).

The testing environment covers 30 different test cases by testing the sensors output with different weather/time of day and different levels of progressive ADAS configurations, from only emergency braking to complete Level 3 ADAS package enforcement. All of the ADAS functions rely only on the information as perceived by the virtual sensors installed on the simulated ego vehicle. Longitudinal control is managed using a PID controller with anti-windup, while a time-based sensor fusion persistence mechanism ensures perception continuity during weather-induced sensor degradations. A post-stop resume protocol limits cruise speed following red-light stops, and a map-based radar lane association algorithm provides ground-truth lane filtering independent of road curvature estimation.

The outputs are assessed using several important parameters such as Time-to-Collision (TTC), radar false positive (FPR) and false negative rates (FNR), variance of headway distances (σ), dropout rates of sensors, Real-Time Factor (RTF), Lateral Deviation and the number of fatal collisions that occur during the simulated cases. The results demonstrate a progressive safety narrative: partial ADAS configurations (ACC with Lane Centering Assist) introduced collision risk under degraded conditions by enabling speeds that exceeded the sensor envelope, while the complete Level 3 package — incorporating a speed limiter capped at 80 km/h — achieved zero collisions across all six weather and lighting conditions.

Contents

List of Figures i

List of Tables iii

List of Acronyms iv

Chapter 1: Introduction 1

1.1 Overview 1

1.2 Levels of Driving Automation 1

1.3 Simulation in ADAS Development 2

1.3.1 CARLA Simulation Environment 2

1.4 ADAS Functional Architecture 3

1.5 Safety Performance Metrics 4

Chapter 2: Literature Review 1

2.1 Vehicle Control Techniques in CARLA Simulator 1

2.1.1 PID Controller Implementation 1

2.1.2 Rule-Based and Threshold Control Systems 2

2.1.3 Comparative Analysis of Control Methodologies 2

2.2 Traffic Management and Sensor Fusion 3

2.2.1 CARLA Traffic Manager Architecture 3

2.2.2 Multi-Sensor Fusion Approaches 3

2.2.3 Radar Processing and False Positive Rejection 3

2.3 Sensor Degradation in Autonomous Vehicles 3

2.3.1 Fundamental Sensor Vulnerabilities 4

2.3.2 Environmental Factors Affecting Sensor Performance 4

2.4 Weather Effects on Sensor Performance 5

2.4.1 Precipitation Impact Analysis 5

2.4.2 Combined Weather Effects 6

2.5 Time-of-Day Effects on Sensor Performance 6

2.5.1 Sunrise Conditions 6
2.5.2 Noon Illumination Characteristics 6
2.5.3 Night-Time Operations 6
2.6 CARLA Simulation Framework Implementation 7
2.6.1 CARLA Architecture and Sensor Configuration 7
2.6.2 Traffic Light Detection and Lane Keeping 8
2.7 Research Gap 8
Chapter 3: Simulation Methodology 1
3.1 Features of the Simulation Environment 1
3.1.1 CARLA Simulator Description 1
3.1.2 Simulating Vehicle Behavior 1
3.1.3 Differences in Vehicle Simulation Models 2
3.2 System Architecture and Schematic Diagrams 2
3.3 Simulating Perception and Virtual Sensor Configuration 4
3.3.1 RGB Camera Array 4
3.3.2 Perception Sensors 5
3.3.3 Radar Sensor 5
3.3.4 Safety and Event Sensors 6
3.3.5 Sensor Suite Design Rationale 6
3.4 Collision Recovery and Wrong-Way Correction 7
3.4.1 Collision Recovery Mechanism 8
3.4.2 Wrong-Way Detection and Correction 8
Chapter 4: ADAS implemented in simulation 10
4.1 Adaptive Cruise Control (ACC) Logic 10
4.1.1 PID Controller Architecture 10
4.1.2 Sensor Fusion for Distance Measurement 10
4.1.3 Time-Based Fusion Persistence 13
4.1.4 Three-Zone ACC Control Strategy 14
4.2 Lane Centering Assist (LCA) 17

4.3 Traffic Light Detection and Automated Braking 19
4.3.1 The Logic Adopted 20
4.3.2 Post-Stop Resume Protocol 21
Chapter 5: Experimental Conditions 23
5.1 Weather Degradation Model 23
5.2 Experimental Test Matrix and Scenarios 23
5.2.1 Weather and Time of Day 23
5.2.2 Complete ADAS Configuration Matrix 24
5.3 Test Environment in Town 04 25
5.4 Traffic Configuration 25
5.5 Input Parameters for Simulation 27
5.5.1 Real-Time Performance Optimization 27
5.6 Data Extraction and Performance Metrics 28
Chapter 6: Experiment Simulations and Results 32
6.1 Simulation Results of the various ADAS Features 32
6.1.1 Baseline Performance - Configuration #0 32
6.1.2 Adaptive Cruise Control Performance - Configuration #1 34
6.1.3 ACC with Lane Centering Assist – Configuration #2 35
6.1.4 ACC, LCA and Traffic Light Assist – Configuration #3 36
6.1.5 Complete Level 3 ADAS Package – Configuration #4 38
6.2 Cross-Configuration Comparative Analysis 40
6.2.1 Collision Distribution 40
6.2.2 Sensor Dropout Rate Analysis 41
6.2.3 Lane Invasion Analysis 42
6.2.4 Speed-Envelope Safety Mechanism 42
6.3 Iterative Development: Comparison with Preliminary Script Results 43
6.3.1 Real-Time Factor (RTF) Improvement 43
6.3.2 Lane Invasion Reduction 44
6.3.3 Radar Performance: FPR and FNR Trade-Off 44

6.3.4 Distance Accuracy and Headway Variance 45
6.3.5 Dropout Rate Comparison 45
6.3.6 Collision Count Consistency 46
Chapter 7: Discussion and Conclusions 47
7.1 Summary of Key Findings 47
7.2 The Partial Automation Paradox 48
7.3 Environmental Degradation Pathways 49
7.3.1 Noon Solar Angle: Depth Camera Degradation 49
7.3.2 Night-Time: Depth Estimation Accuracy Degradation 50
7.3.3 Heavy Rain: Compound Multi-Sensor Degradation 51
7.4 Speed-Envelope Matching as the Primary Safety Mechanism 52
7.5 Iterative Development Validation 53
7.6 Principal Conclusions 54
Chapter 8: Limitations and Future Work 57
Appendix A: Test/Ego Vehicle Sensor and Collision configuration 59
A-1: Test/Ego Vehicle Sensor Configuration 59
A-2: Radar Processor and Lane Filtering 61
A-3: Collision Recovery and Respawning mechanism 62
A-4: Wrong Way Detection and Respawning mechanism 63
Appendix B: ADAS Implementation 67
B-1: PID Controller Class 67
B-2: Visual Fusion Processor (Dual-Tier) 68
B-3: Timing + Fusion Persistence Configuration 70
B-4: Three Zone ACC + Emergency hierarchy (7 Priority Tiers) 71
B-5: LCA Module 75
B-6: Traffic Light Braking (Map-Based Detection & Heuristic) 76
Appendix C: Simulation Criteria and Output Extraction 79
C-1: Camera Degradation due to weather 79

C-2: Weather Definition 80

C-3: Road ID Tracing 81

C-4: Fixed Traffic 82

C-5: Degradation Metrics Tracker 84

C-6: Vehicle State Tracker Initialization 86

Bibliography 88

List of Figures

Figure 1: SAE J3016 Levels of Driving Automation (SAE International, 2021) 1
Figure 2: The V-Model development process highlighting the role of simulation in verification and validation (Vector, 2024) 2
Figure 3: Map layout of CARLA Town 04 featuring the figure-eight highway (CARLA Documentation, 2024) 3
Figure 4: Sensor types and their positioning in an autonomous vehicle (Yeong et al., 2021) 4
Figure 5: Schematic Diagram of a hierarchical PID control loop for autonomous vehicle guidance (Michael Wu, 2020) 1
Figure 6: Finite State Machine (FSM) for autonomous driving mode transitions between Cruise, Follow, and Emergency states (Olsson Kth et al., 2016) 2
Figure 7: Comparative overview of sensor modality strengths and vulnerabilities across environmental conditions for autonomous vehicles (Vargas et al., 2021) 4
Figure 8: Impact of heavy rainfall on LiDAR point cloud density and camera image quality, illustrating signal attenuation and visibility reduction (Kim et al., 2023) 5
Figure 9: Comparison of camera-based object detection performance across sunrise, noon, and night-time conditions, demonstrating effective detection range reduction (Kenk & Hassaballah, 2020) 7
Figure 10: Synchronous communication between CARLA and the Python API control interface (CARLA Documentation, 2024) 7
Figure 11: Methodology and System Architecture Flowchart 3
Figure 12: Ego vehicle sensor suite configuration 4
Figure 13: Bird's-eye view + multi-camera display showing all feeds 4
Figure 14: Semantic segmentation + depth camera side-by-side with vehicle detected 5
Figure 15: Sensor Fusion Pipeline 12
Figure 16: ACC Following Schematic Diagram 15
Figure 17: ACC and Lane Centering Assist Logic 18
Figure 18: Traffic Light Detection Schematic Diagram 19

FIGURE 19: FRONT CAMERA APPROACHING A RED LIGHT, EGO VEHICLE BRAKING 21
FIGURE 20: COMPARING DIFFERENT WEATHER CONDITIONS FROM THE FRONT CAMERA............. 24
FIGURE 21: SAFETY METRICS COLLECTION AND PER-WEATHER AGGREGATION PIPELINE......... 29
FIGURE 22: SELECTED CRASH SNAPSHOT FROM CONFIGURATION 0 (NO ADAS) 33
FIGURE 23: CRASH SNAPSHOTS FROM ALL SIX CONFIGURATION 2 COLLISIONS 36
FIGURE 24: SELECTED CRASH SNAPSHOTS FROM CONFIGURATION 3 38
FIGURE 25: NORMAL DRIVING UNDER CONFIGURATION 4 – CARLA NATIVE FOLLOW CAMERA VIEW... 39
FIGURE 26: COLLISION COUNT UNDER DIFFERENT WEATHER CONDITIONS 41
FIGURE 27: COLLISION DISTRIBUTION ACROSS FIVE ADAS CONFIGURATIONS 47
FIGURE 28: SENSOR FUSION DROPOUT RATE ACROSS FIVE ADAS CONFIGURATIONS 48
FIGURE 29: COMPARISON OF CONFIGURATION #1 & #2... 49
FIGURE 30: HEADWAY VARIANCE DISTRIBUTION FOR CONFIGURATION #4 51
FIGURE 31: MULTI-METRIC DEGRADATION COMPARISON FOR CONFIGURATION 3....................... 52
FIGURE 32: DROPOUT RATE VS COLLISION COUNT 53
FIGURE 33: PERFORMANCE COMPARISON BETWEEN PRELIMINARY AND OPTIMIZED SCRIPTS 54
FIGURE 34: ADAS SAFETY NARRATIVE ACROSS FIVE INCREMENTAL CONFIGURATIONS 55

List of Tables

TABLE 1: TESTING CONFIGURATIONS 24

TABLE 2:SPAWN POINT COORDINATES FOR FIXED TRAFFIC CONFIGURATION – AHEAD OF THE TEST VEHICLE 26

TABLE 3:SPAWN POINT COORDINATES FOR FIXED TRAFFIC CONFIGURATION – BEHIND THE TEST VEHICLE 26

TABLE 4: INPUT PARAMETERS FOR EACH SIMULATION RUN 27

TABLE 5: CONFIGURATION 0 RESULTS PER WEATHER CONDITION 32

TABLE 6: CONFIGURATION 1 RESULTS PER WEATHER CONDITION 34

TABLE 7: CONFIGURATION 2 RESULTS PER WEATHER CONDITION 35

TABLE 8: CONFIGURATION 3 RESULTS PER WEATHER CONDITION 37

TABLE 9: CONFIGURATION 4 RESULTS PER WEATHER CONDITION 39

TABLE 10: COLLISION MATRIX 40

TABLE 11: RADAR % DROPOUT MATRIX 41

TABLE 12: LANE INVASION MATRIX 42

TABLE 13: DIRECT COMPARISON BETWEEN CONFIGURATION 2 AND 4 42

TABLE 14: RESULTS OF EARLIER CARLA SCRIPT 43

TABLE 15: DIRECT COMPARISON BETWEEN CONFIGURATION 1 AND 2 48

List of Acronyms

1. **ACC** – Adaptive Cruise Control
2. **ADAS** – Advanced Driver Assistance Systems
3. **API** – Application Programming Interface
4. **CARLA** – Car Learning to Act
5. **FNR** – False Negative Rate
6. **FOV** – Field of View
7. **FPR** – False Positive Rate
8. **FPS** – Frame Per Seconds
9. **HSV** – Hue, Saturation, Value
10. **LCA** – Lane Centering Assist
11. **LiDAR** – Light Detection and Ranging
12. **PID** – Proportional-Integral-Derivative
13. **RMSE** – Root Mean Square Error
14. **RTF** – Real-Time Factor
15. **TLA** – Traffic Light Assist
16. **TM** – Traffic Manager
17. **TTC** – Time-to-Collision
18. **SSM** – Surrogate Safety Measure

Chapter 1: Introduction

1.1 Overview

Advanced driver assistance systems (ADAS) emerged as a transformative force in modern transportation, revolutionizing road safety and traffic efficiency (SAE International, 2021). As technology progresses from conceptual design to its real-world implementation, the most important challenge it incurs is ensuring safety and reliability using comprehensive testing methodologies. The focus of this thesis is to evaluate the safety of critical maneuvers in simulated road traffic scenarios, moving from Level 0 up to Level 3 ADAS capabilities, and to investigate the resilience of an autonomous system within the CARLA (Car Learning to Act) simulation framework. By utilizing a cluster of sensor perception suite, this research evaluates vehicle performance across diverse weather conditions and functional configurations to identify the boundaries and limitations of current automation logic.

1.2 Levels of Driving Automation

The Society of Automotive Engineers (SAE) International has established a taxonomy for driving automation, formally known as SAE J3016, categorizing the automation into six levels ranging from Level 0 (no automation) to Level 5 (full automation) (SAE International, 2021). Where Level 3 represents a significant milestone, representing conditional driving automation where the vehicle handles all aspects of the Dynamic Driving Task (DDT) under specific conditions but still requires the human driver to be ready to intervene when the system prompts for control (Autocrypt, 2023).

Figure 1: SAE J3016 Levels of Driving Automation (SAE International, 2021)

1.3 Simulation in ADAS Development

Simulation is an engineer's best friend in this technological era. It offers a safe, cost-effective, and repeatable environment well suited for testing complex scenarios for the development and validation of ADAS (Zhou et al., 2016). Virtual testing environments enable engineers to evaluate system performance across millions of test cases, including critical cases such as sensor degradation and output fluctuations due to adverse weather and high-density traffic. Modern simulation approaches follow the V-model development process, supporting Model-in-the-Loop (MIL), Software-in-the-Loop (SIL), and Hardware-in-the-Loop (HIL) testing stages (Vector, 2024).

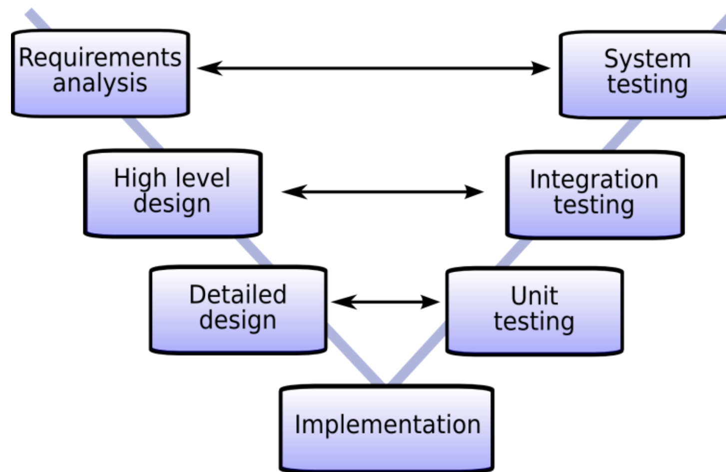

Figure 2: The V-Model development process highlighting the role of simulation in verification and validation (Vector, 2024)

1.3.1 CARLA Simulation Environment

The primary focus of this thesis is to conduct the testing of ADAS system in Town 04 of CARLA simulator. This town is characterized by its mountainous backdrop and a unique "figure-eight" highway purposefully designed for infinite loop testing (CARLA Documentation, 2024). This environment offers ideal conditions to evaluate critical maneuvers like Adaptive Cruise Control (ACC), Lane Centering Assist (LCA), Intelligent Speed Assist (ISA) and Traffic Light Assist (TLA). CARLA's API facilitates the systematic manipulation of weather parameters, allowing us to study the sensor performance and degradation across various distinct atmospheric conditions.

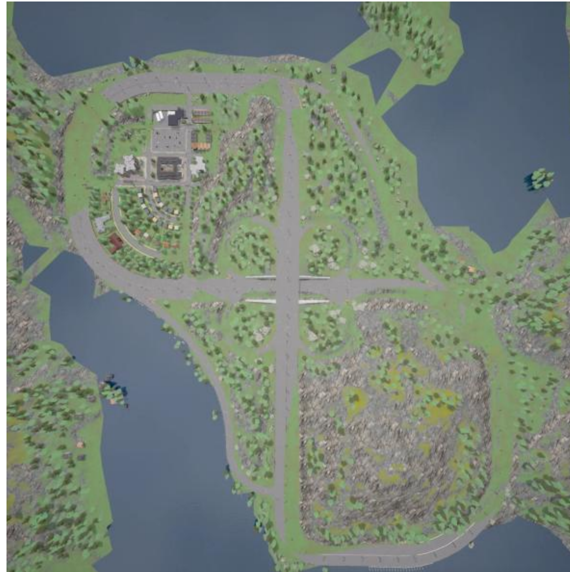

Figure 3: Map layout of CARLA Town 04 featuring the figure-eight highway (CARLA Documentation, 2024)

1.4 ADAS Functional Architecture

Modern ADAS functions are based on a hierarchical approach consisting of perception, planning, and control layers (Yeong et al., 2021). The first layer is perception, aggregates data from multiple sensor modalities such as cameras, radar, and LiDAR and converts the collected data into a unified environmental construct. The second layer is planning, which uses this construct to make crucial decisions regarding but not limited to maintaining a safe following distance, lane centering, responding to traffic signals. The third layer is control, which translates these decisions into precise actuator commands which are then implemented on throttle, braking, and steering (Shrivastava et al., 2023).

For Level 3 conditional autonomy, the perception system must provide 360° coverage so that a single sensor failure does not affect the integrity of the whole system and in turn vehicle safety. Typically, this requires a combination of forward-facing radar to effectively calculate direct velocity and distance, multiple camera types like RGB, semantic, or depth are used for object classification and spatial mapping. Moreover, dedicated sensors are installed for specific functions such as traffic light recognition and lane marking detection (Mercedes-Benz Group, 2022).

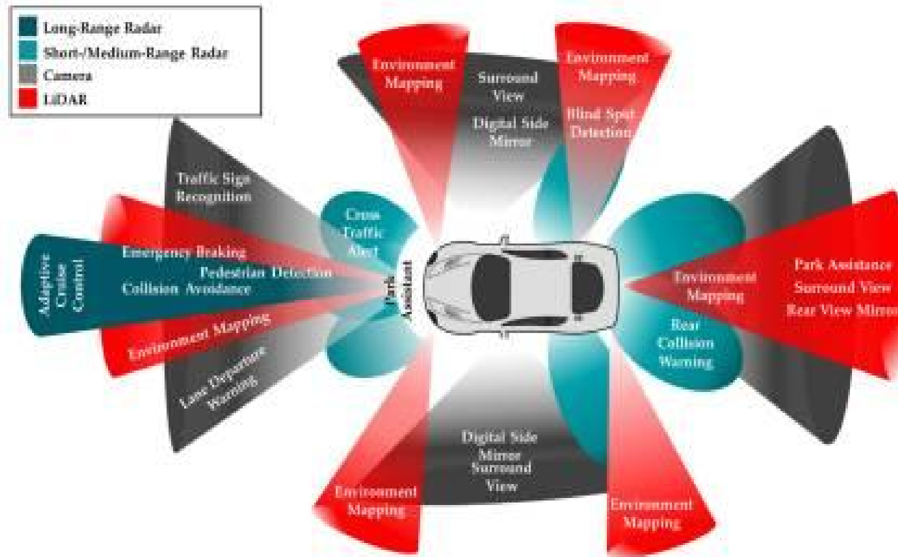

Figure 4: Sensor types and their positioning in an autonomous vehicle (Yeong et al., 2021)

1.5 Safety Performance Metrics

Safety evaluation of autonomous vehicles relies heavily on Surrogate Safety Measures (SSMs). SSMs are indicators that detect and quantify conflict's severity without requiring actual collisions to occur (Yan et al., 2023). Among these indicators are Time-to-Collision (TTC), which is the widely accepted metric for longitudinal conflict analysis, it measures the time remaining before a potential rear-end collision given that current speeds and trajectories remain constant. TTC has been extensively validated in both real-world and simulated highway settings as a reliable indicator of collision imminence (Wang et al., 2023).

Apart from proximity-based measures, the reliability of the perception system itself is a fundamental safety measure, where false positive and false negative detection rates quantify how often sensors incorrectly report or fail to detect objects in the environment, affecting the quality of decisions made by the planning and control layers (Bijelic et al., 2020). Headway distance variance offers an additional metric on control stability - high variance indicates oscillatory or inconsistent gap management that signals underlying perception or controller issues.

Collectively, these metrics form a multi-dimensional evaluation framework that captures both the safety outcomes of autonomous behavior and the reliability of the perception systems driving those decisions.

Chapter 2: Literature Review

The integration of Level 3 conditional autonomous vehicles into modern traffic infrastructure requires a complex merger and integration of control theory, perception resilience, and environmental modeling. This chapter reviews the academic research conducted regarding vehicle control within the CARLA simulation environment, multi-modal sensor fusion strategies, and the quantifiable impact of environmental degradation on autonomous perception suites.

2.1 Vehicle Control Techniques in CARLA Simulator

Control methodologies in high-fidelity simulators are tasked with translating high-level mission objectives into stable actuator commands while maintaining safety boundaries (Jaimin K, 2021).

2.1.1 PID Controller Implementation

Proportional-Integral-Derivative (PID) controllers are the foundational methodology for longitudinal speed and distance management in autonomous driving research due to their computational efficiency and deterministic behavior (Borase et al., 2020; Michael Wu, 2020). The PID algorithm minimizes the error "e(t)" between the desired set-point and the actual state:

$$\boldsymbol{u(t) = Kp.\,e(t) + Ki.\int e(t)dt + Kd.\frac{d}{dt}e(t)} \qquad \boldsymbol{eq.1}$$

In autonomous contexts, hierarchical PID structures are common, where an upper-level controller computes required acceleration based on inter-vehicle distance, and a lower-level controller manages throttle and brake pressure (Shrivastava et al., 2023). To prevent integral windup during prolonged braking such as at signalized intersections, modern implementations utilize anti-windup mechanisms that clip the integral term within specific bounds (e.g., ±50.0) to ensure rapid recovery and prevent velocity overshoot (Michael Wu, 2020; Shrivastava et al., 2023).

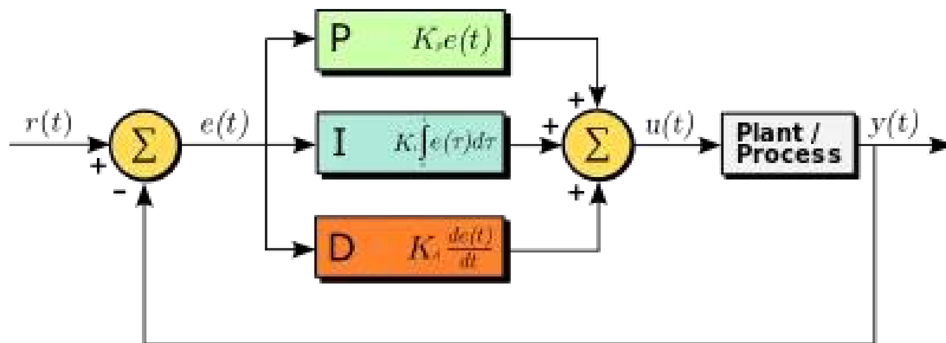

Figure 5: Schematic Diagram of a hierarchical PID control loop for autonomous vehicle guidance (Michael Wu, 2020)

2.1.2 Rule-Based and Threshold Control Systems

Rule-based systems provide a predictable framework for decision-making using predefined behavioral logic and threshold-driven transitions (CARLA Documentation, 2024). These often utilize Finite State Machines (FSMs) or Behavior Trees (BT) to manage complex maneuvers like lane keeping or speed regulation (Maciek Dziubinski, 2019a, 2019b; Olsson Kth et al., 2016). Behavior Trees, in particular, offer superior modularity by evaluating conditions (e.g., "Is the time gap below 1.2 seconds?") and actions (e.g., "Apply emergency braking") in a hierarchical structure, allowing the vehicle to switch reactively between Cruise, Follow, and Emergency modes based on sensor thresholds (Anand & Ohol, 2023; Olsson Kth et al., 2016).

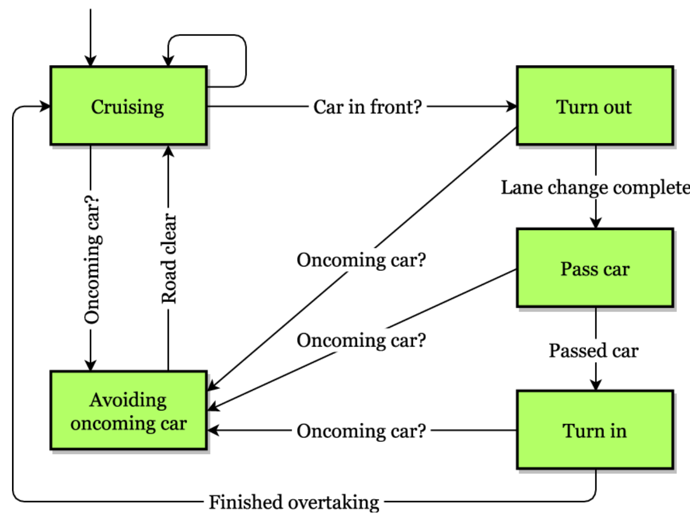

Figure 6: Finite State Machine (FSM) for autonomous driving mode transitions between Cruise, Follow, and Emergency states (Olsson Kth et al., 2016)

2.1.3 Comparative Analysis of Control Methodologies

Literature reveals distinct trade-offs between classical and learning-based control. PID controllers demonstrate high trajectory tracking precision, with Root Mean Square Errors (RMSE) typically below 0.5 meters in lateral deviation (Luca Venturi & Krishtof Korda, 2020). Conversely, Deep Reinforcement Learning (DRL) approaches, such as Deep Deterministic Policy Gradient (DDPG), offer improved fuel efficiency (up to 15%) and better adaptation to unpredictable environments (D. Li & Okhrin, 2023; Pérez-Gil et al., 2022). However, PID systems are preferred for large-scale multi-agent simulations due to their low computational footprint (approximately 1–2% CPU) compared to the heavy GPU demands of DRL models (Idrees Shaikh, 2024). For this thesis, PID control is selected specifically because the multi-vehicle traffic density and twelve concurrent sensor streams necessitate computational efficiency that learning-based methods cannot provide without dedicated GPU inference pipelines.

2.2 Traffic Management and Sensor Fusion

Ensuring safety in dense traffic requires a unified environmental model derived from multiple perception streams and intelligent traffic orchestration.

2.2.1 CARLA Traffic Manager Architecture

The CARLA Traffic Manager (TM) manages background actors using server-client architecture. It utilizes "floating bubble" logic to dynamically spawn and destroy vehicles around the ego vehicle, maintaining a relevant traffic density without exceeding computational limits (CARLA Documentation, 2024). TM supports Gaussian-distributed speed variations and rule-based collision avoidance, which provides a naturalistic background for testing Level 3 autonomous features (CARLA Documentation, 2024).

2.2.2 Multi-Sensor Fusion Approaches

Sensor fusion mitigates the limitations of individual sensors by combining data at various levels: early (raw), mid (feature), or late (decision) (MathWorks, 2024; Yeong et al., 2021). For Level 3 systems, fusion between semantic segmentation cameras and radar is critical; cameras provide high-resolution classification of objects like traffic signs and lane markings, while radar provides direct, high-precision velocity and range measurements (Bijelic et al., 2020; Yeong et al., 2021). This synergistic approach allows for robust object detection even in scenarios where one sensor modality is compromised (Yeong et al., 2021).

2.2.3 Radar Processing and False Positive Rejection

Radar systems frequently encounter false positives from stationary roadside infrastructure like guardrails and bridges (Bijelic et al., 2020), to address this issue lateral lane filtering and dynamic velocity thresholding (e.g., rejecting detections with a relative velocity signature matching the ego-speed) are implemented to distinguish irrelevant static objects from genuine threats (Bijelic et al., 2020). Advanced geometric filtering using azimuth and altitude data further reduces clutter in complex urban maps (Bijelic et al., 2020).

2.3 Sensor Degradation in Autonomous Vehicles

The long-term reliability of autonomous systems depends on understanding both intrinsic vulnerabilities and external performance decay.

2.3.1 Fundamental Sensor Vulnerabilities

Perception hardware is susceptible to physical and technical limitations. LiDAR systems can experience a 10–15% reduction in point cloud density over extended lifecycles due to laser diode aging (Royo & Ballesta-Garcia, 2019). Camera sensors are highly temperature-sensitive, with image quality degrading significantly outside the −10°C to 50°C range, while radar remains the most resilient to physical contamination, maintaining up to 95% detection accuracy even in soiled conditions (Bijelic et al., 2020; Royo & Ballesta-Garcia, 2019).

2.3.2 Environmental Factors Affecting Sensor Performance

External noise floors in urban environments, such as electromagnetic interference from power lines, can increase the radar noise floor by 3–5 dB, potentially masking small targets like pedestrians (Ali Alheeti & Mc Donald-Maier, 2018). Furthermore, vibration-induced wear in mechanical LiDAR systems can increase measurement uncertainty from ±2 cm to ±5 cm RMS over time, a challenge largely mitigated by the adoption of solid-state LiDAR architectures (Royo & Ballesta-Garcia, 2019; Vargas et al., 2021).

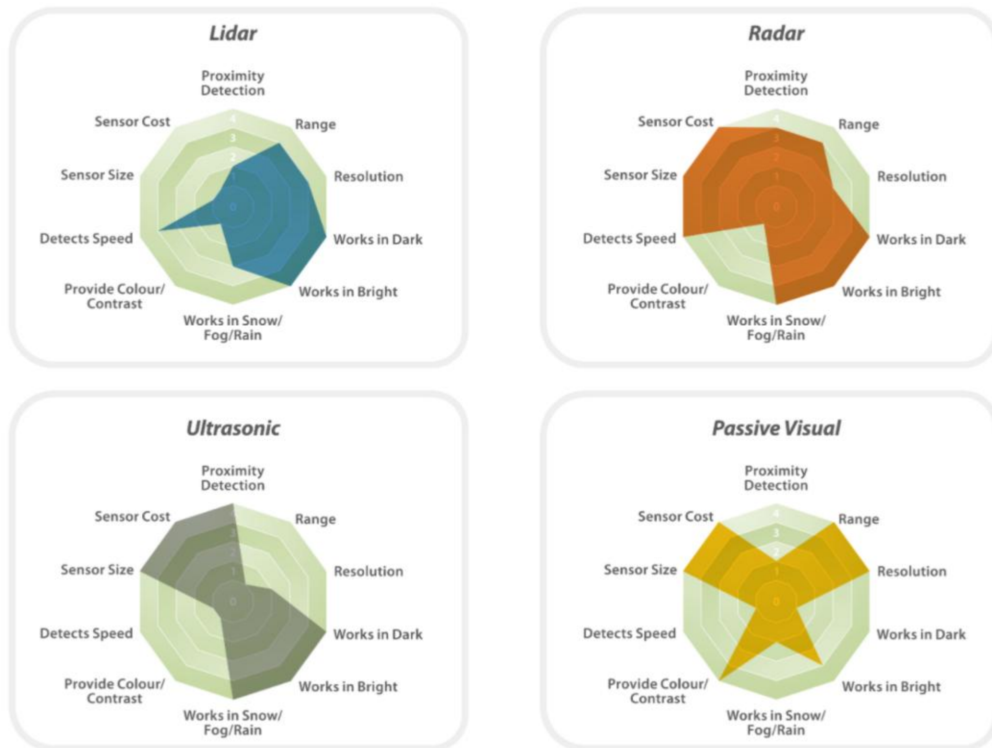

Figure 7: Comparative overview of sensor modality strengths and vulnerabilities across environmental conditions for autonomous vehicles (Vargas et al., 2021)

2.4 Weather Effects on Sensor Performance

Atmospheric particulates significantly impair optical sensors, requiring perception suites to adapt their confidence weightings dynamically.

2.4.1 Precipitation Impact Analysis

Heavy rainfall (>25 mm/h) introduces water droplet scattering, reducing camera visibility by 40–60% and significantly obscuring the edges of distanced objects (Bijelic et al., 2020). LiDAR sensors suffer from exponential signal attenuation following the Beer-Lambert law, reducing effective detection ranges from 100+ m to 30–50 m in torrential rain(Kim et al., 2023). Radar remains functionally operational in rain but can experience a 10–15% range reduction due to atmospheric absorption at 77 GHz (Yoneda et al., 2019).

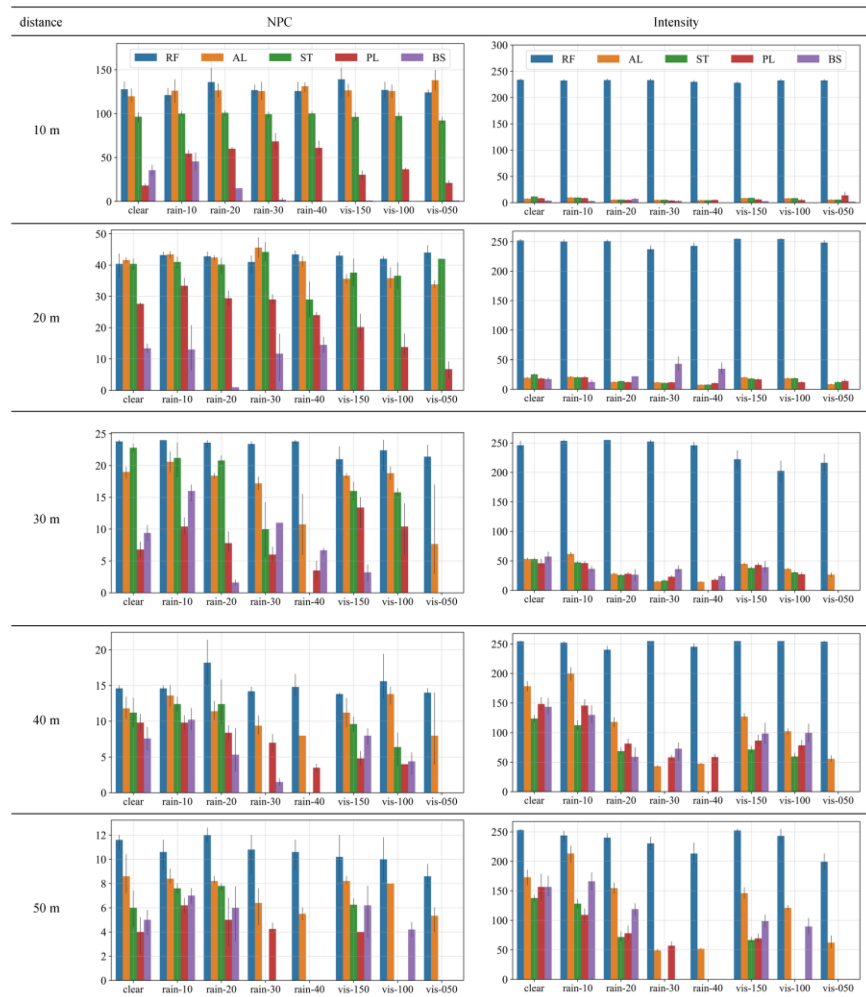

Figure 8: Impact of heavy rainfall on LiDAR point cloud density and camera image quality, illustrating signal attenuation and visibility reduction (Kim et al., 2023)

2.4.2 Combined Weather Effects

Synergistic environmental conditions create compounding degradation. Research indicates that the combination of rain and other atmospheric stressors can reduce the overall effectiveness of a sensor suite by 50–70%, a level significantly higher than the impact of any single weather phenomenon alone (Kumar & Muhammad, 2023). This highlights the need for redundant multi-modal suites where radar and thermal imaging can compensate for the failure of optical cameras and LiDAR (Kumar & Muhammad, 2023).

2.5 Time-of-Day Effects on Sensor Performance

Diurnal variations in illumination create unique challenges for visual-based perception algorithms.

2.5.1 Sunrise Conditions

Low sun angles (<20°) at sunrise create intense glare and high dynamic range (HDR) scenes exceeding 120 dB, which often saturates standard camera sensors capable of only 60–70 dB (Maddern et al., 2020). This illumination imbalance severely impacts object detection accuracy, with average precision (AP) for standard models potentially dropping from 85% to 55% (Maddern et al., 2020).

2.5.2 Noon Illumination Characteristics

Intense noon sunlight (>100,000 lux) can cause blooming effects and sensor saturation, particularly when reflecting off wet or metallic surfaces (Dai & Gool, 2018). Furthermore, high ambient temperatures during midday can induce thermal noise in image sensors, reducing the signal-to-noise ratio by 2–4 dB and requiring adaptive exposure control algorithms (Sun et al., 2019).

2.5.3 Night-Time Operations

Night driving reduces the effective range of standard RGB cameras from 100+ m to approximately 30–50 m under typical headlight illumination (Kenk & Hassaballah, 2020). To maintain Level 3 conditional safety, systems must integrate active illumination or thermal imaging modalities, which can detect heat signatures of road users at ranges exceeding 150 m regardless of ambient light conditions (Kenk & Hassaballah, 2020).

Figure 9: Comparison of camera-based object detection performance across sunrise, noon, and night-time conditions, demonstrating effective detection range reduction (Kenk & Hassaballah, 2020)

2.6 CARLA Simulation Framework Implementation

CARLA provides a deterministic environment essential for the reproducible evaluation of safety-critical maneuvers (CARLA Documentation, 2024).

2.6.1 CARLA Architecture and Sensor Configuration

CARLA operates using a client-server architecture in synchronous mode, ensuring fixed time steps for physically accurate dynamics (CARLA Documentation, 2024; P. Li et al., 2021). The platform supports bespoke sensor blueprints where users can customize intrinsic parameters (e.g., focal length, sensor noise) and extrinsic placement to replicate real-world Level 3 architectures (CARLA Documentation, 2024; P. Li et al., 2021).

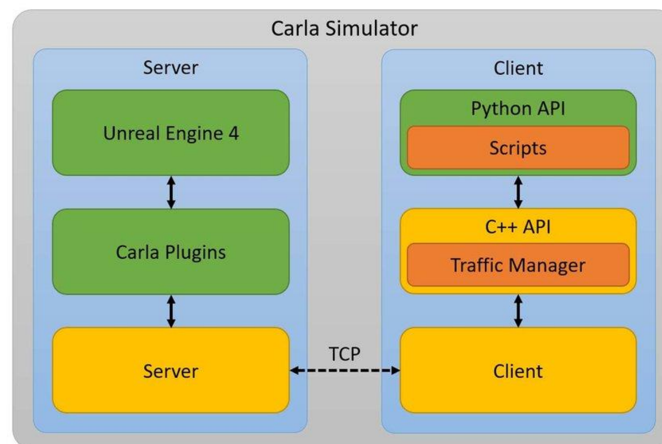

Figure 10: Synchronous communication between CARLA and the Python API control interface (CARLA Documentation, 2024)

2.6.2 Traffic Light Detection and Lane Keeping

Traffic light recognition typically employs vision-based HSV color space analysis for real-time state identification, often cross-referenced with map-based ground truth data for stop-line distance calculation (Shrivastava et al., 2023). Lane Centering Assist (LCA) systems utilize waypoint-based steering with dynamic lookahead distances that scale with vehicle velocity (typically 10–20 meters) to maintain stability at high speeds (Coulter & R.C., 1992).

2.7 Research Gap

Despite advances in vehicle control, sensor fusion, and environmental modeling, the existing literature examines these components largely in isolation. PID-based control is evaluated under nominal sensor conditions, sensor fusion is tested against individual weather perturbations, and environmental degradation is quantified for single sensor modalities without assessing system-level safety outcomes. Three specific gaps emerge from this review.

First, no prior study evaluates whether progressively adding ADAS capabilities to a vehicle improves or degrades safety. Existing work tests individual functions (ACC alone, LCA alone) but does not examine the interactions that arise when these features are combined incrementally on the same platform under the same conditions.

Second, the combined impact of weather and time-of-day variations on a multi-sensor perception pipeline has not been systematically mapped. Studies quantify rain degradation or night-time degradation individually, but do not evaluate whether these effects compound, cancel, or operate independently when applied to a complete sensor fusion architecture.

Third, the relationship between vehicle speed and perception capability as a safety mechanism has not been isolated experimentally. While speed limitation is a common engineering practice, no study has demonstrated whether matching vehicle speed to the sensor detection envelope is more effective than improving sensor performance itself.

This thesis addresses these three gaps through a 30-run evaluation framework comprising five incremental ADAS configurations tested across six weather and lighting combinations in the CARLA simulation environment, using a twelve-sensor fusion architecture and eight quantitative safety metrics.

Chapter 3: Simulation Methodology

3.1 Features of the Simulation Environment

The evaluation of critical maneuvers in complex traffic environments necessitates a high-fidelity simulation tool chain that ensures physical accuracy and deterministic reproducibility. Virtual environments allow for the assessment of automated driving systems under conditions that are hazardous or logistically impractical for real-world testing. For this thesis CARLA v0.9.15 was employed to simulate test vehicles, implement vehicle features and scenario conditions, including traffic.

3.1.1 CARLA Simulator Description

CARLA is the primary open-source simulator used in this research. Built upon Unreal Engine 4, it utilizes a scalable client-server architecture where the server manages physics calculations (PhysX), environment rendering, and the state of all actors, while the client-side interfaces via a Python API to control agent logic (CARLA Documentation, 2024). CARLA is specifically chosen for its support of bespoke sensor blueprints and its ability to simulate precise environmental degradation. The simulation operates in synchronous mode with a fixed time step of 0.05 seconds (20 FPS) to ensure deterministic, reproducible physics and sensor timing across all test runs. The 20 FPS tick rate was selected to balance computational overhead from the twelve-sensor suite with sufficient temporal resolution for the PID controllers and persistence timing mechanisms.

3.1.2 Simulating Vehicle Behavior

The ego vehicle used throughout all the test scenarios is the Mini Cooper S (2021), selected from CARLA's native vehicle blueprint library (CARLA Vehicle Catalogue, 2024). This vehicle was chosen for several reasons: its compact dimensions (length ~3.8 m, wheelbase ~2.5 m) are representative of the modern urban passenger car segment where Level 3 ADAS deployment is currently most active. Its low center of gravity and short wheelbase produce dynamic behavior that is meaningfully sensitive to the longitudinal and lateral control inputs of the PID and LCA systems under evaluation, making deviations in control performance more detectable in the safety metrics.

While the simulation framework supports both manual and automated control, the primary evaluation is grounded in the performance of the autopilot-driven ego vehicle across the five incremental ADAS configurations. For the formal test matrix, the thesis employs a fixed traffic configuration of 20 vehicles placed at predetermined highway positions, 15 ahead and 5 behind the

ego vehicle - to ensure deterministic, reproducible conflict scenarios across all 30 test runs. The CARLA Traffic Manager (TM) governs the behavior of these background vehicles using rule-based decision layers for collision avoidance, lane changes, and traffic regulation compliance (CARLA Documentation, 2024). The system additionally supports a random traffic mode with up to 80 vehicles using the TM's "floating bubble" spawn logic, which is available for exploratory testing but is not used in the formal evaluation.

3.1.3 Differences in Vehicle Simulation Models

The simulation distinguishes between the Ego vehicle, which is the vehicle to be tested and equipped with sensors and control logic, and other vehicles, which contribute to creating the scenario, introducing interaction with the ego vehicle. Both deterministic and stochastic behavioral models can be used to manage the vehicle motion:

- **Background Actors:** Exhibit variable behaviors governed by Gaussian speed distributions and randomized path choices at junctions. The Traffic Manager assigns each vehicle a target speed with random offsets and configures basic collision avoidance behaviors (CARLA Documentation, 2024).
- **Ego Vehicle:** Operates under a strict hierarchical control pipeline where high-level tactical decisions (e.g., emergency braking, gap management) are translated into stable commands by dual PID controllers for speed and distance, ensuring precise longitudinal and lateral guidance.

3.2 System Architecture and Schematic Diagrams

The work follows a modular architecture with a perception-planning-control pipeline, visualizing the data flow from environmental sensors to hardware actuation. The system processes data in the following sequential stages each simulation tick:

1. Sensor data acquisition from all the sensors[1] installed on the test vehicle,
2. Dual-range sensor fusion combining bumper-camera close-range and roof-camera long-range detections,
3. Time-based persistence filtering to reject transient dropouts,
4. Priority-based control decision logic across seven priority tiers,
5. Final actuation commands to throttle, brake, and steering

[1] In total 12 sensors have been used for the vehicles maneuvering – explained in more detail in Section 3.3

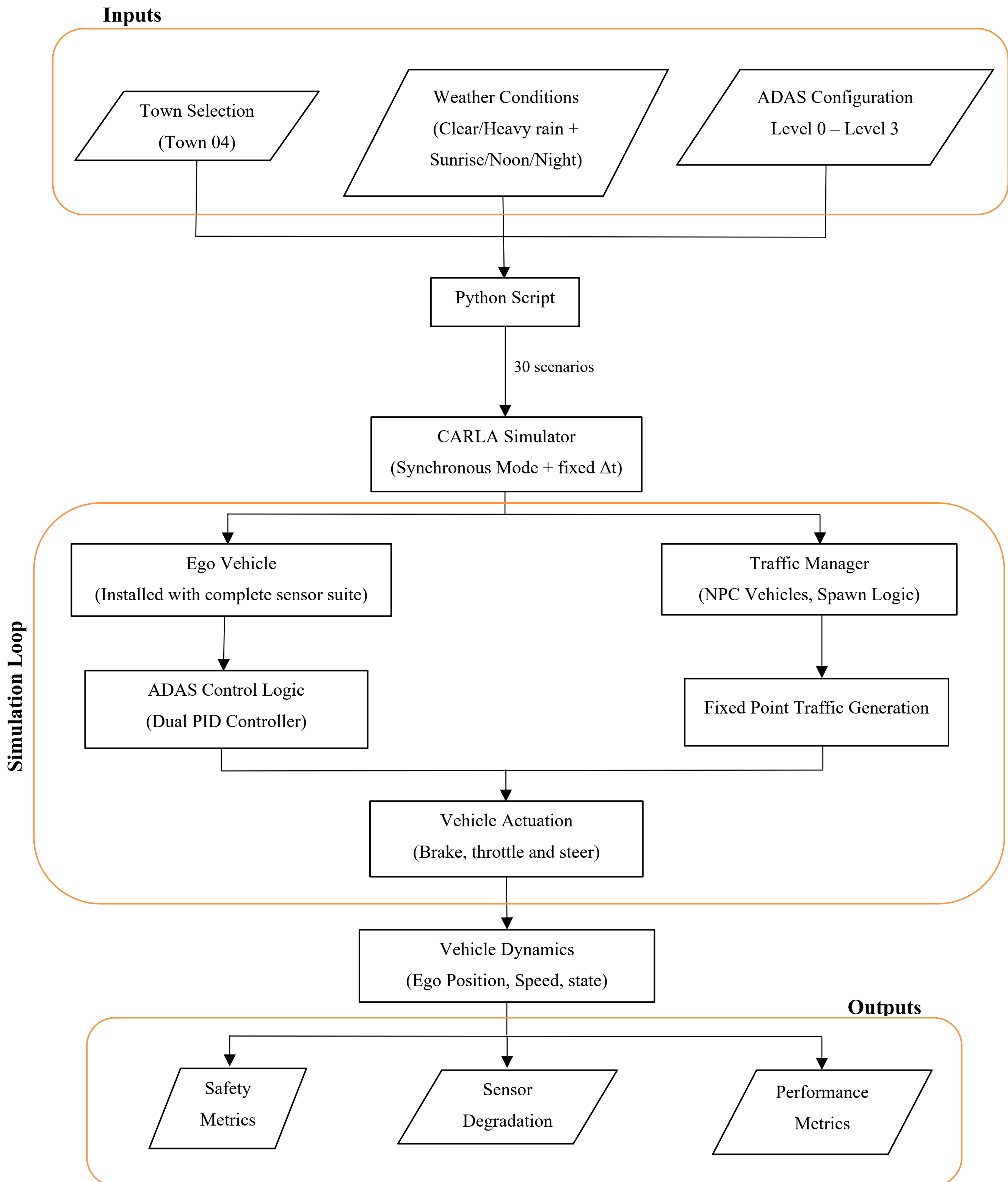

Figure 11: Methodology and System Architecture Flowchart

3.3 Simulating Perception and Virtual Sensor Configuration

The sensor specifications are derived directly from the simulation script configuration and are detailed below.

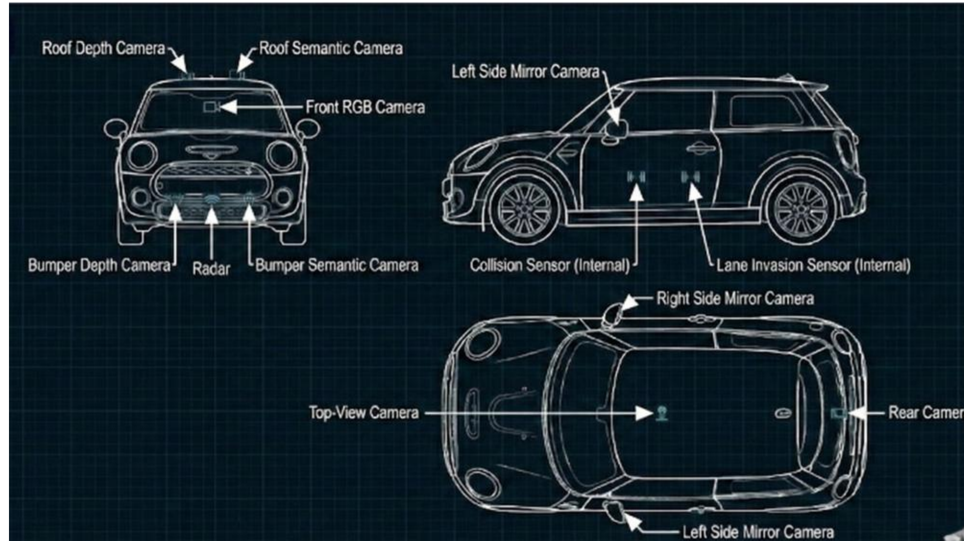

Figure 12: Ego vehicle sensor suite configuration

3.3.1 RGB Camera Array

The vision suite comprises five RGB cameras. A front-facing main camera (800×600, 90° FOV, ISO 1200, gamma 2.2, histogram exposure mode) provides high-resolution visual data for the driver display and traffic light heuristic detection. Four additional cameras emulate traditional mirror systems: left and right-side mirrors (280×180, 90° FOV) mounted at y = ±0.8 m with yaw angles of −140° and +140°, and a rear camera (450×180, 90° FOV) at x = −2.0 m, z = 2.4 m with 180° yaw. A bird's-eye-view camera (160×160, 70° FOV) is mounted 10 m above the vehicle with −90° pitch for top-down situational awareness.

Figure 13: Bird's-eye view + multi-camera display showing all feeds

3.3.2 Perception Sensors

Front-mounted semantic segmentation and depth sensors create a synchronized spatial model at two range tiers. The roof-mounted pair (800×600, 90° FOV, positioned at x = 1.5 m, z = 2.4 m) provides long-range detection from 15 to 80 meters. Semantic segmentation delivers pixel-wise labels (e.g., "Vehicles," "Roads," "Traffic Lights") using CityScapes color encoding, while the co-located depth sensor provides calibrated range for obstacle detection. Depth decoding follows CARLA's standard RGB-encoded scheme:

$$depth_m = 1000.00 * \frac{(R + G * 256 + B * 256^2)}{(256^3 - 1)} \qquad eq.2$$

The bumper-mounted pair (400×300, x = 2.3 m, z = 0.4 m, pitch = −20°) provides close-range detection from 2 to 25 meters, with both the semantic camera and depth camera using a wider 100° FOV. This dual-tier approach ensures continuous perception coverage across the full braking distance envelope, with the bumper sensors compensating for the roof sensors' blind spot at close range.

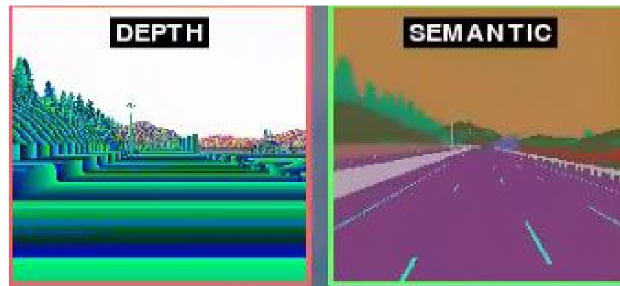

Figure 14: Semantic segmentation + depth camera side-by-side with vehicle detected

3.3.3 Radar Sensor

A 77 GHz Frequency Modulated Continuous Wave (FMCW) radar is mounted at the front of the vehicle (x = 2.0 m, z = 1.0 m) with a 30° horizontal and vertical field-of-view and a maximum detection range of 150 meters. The radar provides direct Doppler-based velocity measurements and range data for each detected point. To reject false positives from static roadside infrastructure, the system applies a map-based lane association algorithm. For each radar detection point, the system queries the CARLA HD map (using world.get_map().get_waypoint()) to obtain the road id and lane id at the detection's world-space coordinates. The detection is retained only if its road id and lane id match those of the ego vehicle's current driving lane. This approach provides ground-truth lane association that is independent of road curvature, eliminating the lateral estimation errors inherent in geometric projection methods on curved highway segments.

Additionally, detections with a relative velocity signature matching ego-speed (indicating a stationary object) and those outside the altitude range of −5° to +8° are filtered. For detections within 20 meters, the velocity filter is bypassed to ensure stopped vehicles are detected at close range; when the ACC is actively tracking a vehicle, this relaxed threshold extends to 40 meters (Appendix A-2). Radar remains the most resilient modality for Level 3 operation, maintaining functionality during heavy rain where optical sensors suffer significant signal attenuation

3.3.4 Safety and Event Sensors

A bumper-mounted semantic segmentation camera functions as the primary safety sensor for intersections. Specifically calibrated with a 100° field of view and angled downward at −20°, it identifies traffic signal states and solid stop lines to ensure the vehicle executes automated braking at red lights. This is complemented by a collision sensor and a lane invasion detector, both attached directly to the vehicle chassis. The collision sensor triggers the collision recovery mechanism and logs fatal collision events. The lane invasion detector identifies crossing of lane markings and distinguishes between dashed-line lane changes (counted as completed lane transitions) and solid-line violations (logged as safety infractions) (Appendix A-1).

3.3.5 Sensor Suite Design Rationale

The sensor suite configuration evolved through three prototype iterations during preliminary development. The initial configuration included lane detection sensors, object detection sensors alongside the camera and radar suite. A second iteration tested a reduced suite of RGB, depth, and semantic segmentation cameras without radar, isolating the contribution of vision-only perception. The underperformance of this configuration in rain conditions, where RGB-based vehicle detection proved unreliable and motivated the final design decision to retain radar as an independent distance channel and to prioritize semantic segmentation over RGB for the perception pipeline. Each subsequent design decision documented in this section emerged from observations made during these iterative tests. By implementing this iterative approach of failure-driven refinement, the final twelve-sensor configuration was established.

The dual-tier fusion architecture (bumper-mounted + roof-mounted camera pairs) was adopted to eliminate a critical detection blind spot. A single roof-mounted camera pair, positioned at z = 2.4 m with a forward-facing 90° field of view, exhibits a geometric blind zone below approximately 15 meters due to its elevation and pitch angle. At highway speeds of 80 km/h (22.2 m/s), a 15-meter gap corresponds to only 0.68 seconds of reaction time well below the critical TTC threshold of 1.5 seconds. During preliminary runs, this blind spot manifested as sudden "distance jumps" where the

measured distance would leap from 18–20 meters directly to zero upon a collision event, with no intermediate close-range tracking. The bumper-mounted pair ($z = 0.4$ m, pitch $= -20°$) was therefore introduced specifically to provide continuous 2–25 m coverage, ensuring that the perception system can track a decelerating lead vehicle through the full braking envelope without losing detection.

The radar lane filtering initially employed a parabolic curvature-adjusted lateral corridor with a 1.6-metre threshold. However, during testing on Town 04's figure-eight highway, the single-waypoint parabolic curvature estimate (computed from a single reference point 30 meters ahead) introduced lateral projection errors of 2–3 meters at depths beyond 50 meters on the tighter bends. This caused the corridor filter to reject legitimate in-lane vehicles on curves while occasionally admitting adjacent-lane vehicles on straights. The radar processor was therefore re-implemented using a map-based lane association approach: each radar detection's world-space coordinates are projected onto the CARLA HD map, and only detections whose road id and lane id match the ego vehicle's current lane are retained. This provides ground-truth lane association regardless of road curvature, eliminating both the false rejections on curves and the false admissions on straights that the geometric approach produced. The map-based filtering applies only to radar detections classified as RADAR_ACTIVE; detections already confirmed by the visual fusion pipeline (FUSION_DETECT) are not re-filtered by the radar lane check, since the radar azimuth and camera depth may reference different vehicles at different ranges. (Appendix A-2)

The decision to use semantic segmentation cameras rather than standard RGB cameras for the perception pipeline was driven by the need for weather-invariant object classification. During initial testing with RGB-only vehicle detection using color-based thresholds, heavy rain and night conditions caused detection accuracy to deteriorate significantly due to reduced contrast and water droplet artifacts. Semantic segmentation provides pixel-wise class labels (Vehicles, Roads, Traffic Lights) that remain reliable across illumination conditions, as the classification is performed by CARLA's rendering engine rather than a learned model susceptible to domain shift. This architectural choice trades detection latency (semantic rendering is computationally heavier than raw RGB) for classification robustness.

3.4 Collision Recovery and Wrong-Way Correction

To enable continuous long-duration testing without manual intervention, the system implements an automated collision recovery mechanism and a wrong-way detection and correction system.

3.4.1 Collision Recovery Mechanism

When a collision is detected by the collision event sensor, the system initiates a three-phase recovery sequence:

- **Phase 1 – Brake (0.5 seconds):** Full braking and handbrake engagement to bring the vehicle to a complete stop.
- **Phase 2 – Reverse (2.0 seconds):** The vehicle reverses at 40% throttle while applying waypoint-based steering correction to align with the road center. The cross-track error to the nearest driving waypoint is calculated and converted to a reverse-steering command:

$$\delta_{reverse} = clip\,(-cross * 0.5, -0.3, 0.3) \qquad eq.3$$

- **Phase 3 – Realign:** The vehicle is realigned to the nearest driving waypoint with the road direction. Before realignment, a forward clearance check ensures no vehicles are within 15 meters ahead. If the check fails, the target waypoint is shifted backward in 8-meter increments up to four times. Upon realignment, all residual angular velocity is zeroed and the vehicle is given an initial forward momentum of approximately 20 km/h (5.5 m/s) to prevent it from sitting stationary on the highway. A 0.5-second sensor stabilization warmup then holds the brakes to allow the camera buffers to update with the new viewpoint, followed by an 8.0-second grace period during which the vehicle is exempt from the stuck-vehicle detection timer to allow normal speed recovery.

A 3.0-second collision cooldown period follows each recovery to prevent cascading collision detections (Appendix A-3). Each collision recovery event is logged as a fatal collision in the DegradationMetrics system, attributed to the current weather condition for per-weather performance analysis.

3.4.2 Wrong-Way Detection and Correction

The system continuously monitors the ego vehicle's orientation relative to the road direction by computing the dot product between the vehicle's forward vector and the nearest waypoint's forward vector:

$$dot = v_{ego_{fwd}} * v_{road_{fwd}} = \cos\alpha \qquad eq.4$$

where α the angle between the vehicle heading and road direction. If dot < −0.5 (corresponding to an angular deviation greater than 120°), the vehicle is flagged as driving the wrong way (Appendix

A-4). The system then performs a 180° correction by teleporting the vehicle to the correct lane orientation at the nearest waypoint. This event is also logged as a fatal collision for safety metrics.

The automated collision recovery system was developed to address a practical challenge in long-duration simulation testing: without recovery, a single collision would leave the vehicle wedged against a guardrail or another vehicle, requiring manual intervention to restart the simulation. Since each of the 30 test scenarios is designed to run for the full length of the highway loop (or a fixed simulation time), even one unrecoverable collision would invalidate the remaining data collection for that run. The three-phase approach (brake → reverse → realign) was chosen over a simple immediate teleport because abruptly relocating a vehicle can cause physics instabilities in CARLA's PhysX engine, including vehicles spawning inside other actors or falling through the road surface. The 0.5-second braking phase ensures the vehicle is stationary before reversal; the 2.0-second reverse phase creates physical separation from the collision obstacle; and the final teleport to the nearest clear waypoint ensures correct lane alignment.

The forward clearance check (15-meter minimum gap with up to four 8-meter backward shifts) was added after observing that the realignment occasionally placed the ego vehicle directly behind a slow-moving or stopped vehicle, causing an immediate secondary collision and a cascading recovery loop. The 0.5-second post-respawn sensor stabilization warmup was introduced because the twelve-sensor suite requires at least one full rendering cycle after a readjustment for all camera buffers to update with the new viewpoint. Without this warmup, the bumper semantic camera would retain the pre-teleport image for one or two frames, potentially detecting the collision obstacle as a close-range vehicle and triggering an immediate emergency braking response at the new position. The 8.0-second stuck-timer grace period was added because the respawned vehicle, starting from approximately 20 km/h, requires several seconds to accelerate back to highway speed; without the exemption, the stuck-vehicle detector would interpret this acceleration phase as a stall and trigger an unnecessary secondary respawn. The 3.0-second collision cooldown prevents the collision event sensor from re-triggering on residual physics contact forces that dissipate over several simulation ticks after the vehicle has already begun recovery.

Chapter 4: ADAS implemented in simulation

4.1 Adaptive Cruise Control (ACC) Logic

The ACC system regulates longitudinal motion by selecting the optimal acceleration to maintain a target speed or a safe gap from preceding traffic (Yu & Wang, 2022).

4.1.1 PID Controller Architecture

The system utilizes two independent PID controllers for longitudinal management. The speed controller regulates the ego vehicle's velocity toward the target speed, while the distance controller manages the gap to the lead vehicle. The general PID control law (eq.1) calculates the required effort (u(t)) to minimize the error between target and actual states.

The speed PID controller is configured with Kp = 0.4, Ki = 0.0132, and Kd = 0.225 with output limits of [−1.0, +1.0]. The distance PID controller uses Kp = 0.2, Ki = 0.005, and Kd = 1.2 with the same output limits. Both controllers implement an anti-windup mechanism that clips the integral accumulator within the range [−50.0, +50.0], preventing integral saturation during prolonged stops at signalized intersections (Appendix B-1). The derivative term is computed only when the time delta (dt) is positive to avoid division-by-zero errors.

4.1.2 Sensor Fusion for Distance Measurement

The ACC system receives distance measurements through a dual-tier sensor fusion pipeline. The Visual Fusion Processor class employs a 3D point-cloud projection approach to convert 2D pixel coordinates into metric lateral offsets, enabling mathematically precise in-lane filtering that replaces the earlier Region of Interest cropping method (Appendix B-2).

Both camera tiers identify vehicle pixels using CityScapes semantic color thresholds for three vehicle classes: Cars (RGB: 0, 0, 142), Trucks (RGB: 0, 0, 70), and Buses (RGB: 0, 60, 100). For each detected vehicle pixel, the CARLA RGB-encoded depth is decoded (eq.2) to obtain a depth value in meters. The pixel's lateral offset in 3D space is then computed using the pinhole camera model:

$$\boldsymbol{d_{lateral}} = (\boldsymbol{u} - \boldsymbol{c_x}) * \frac{\boldsymbol{Z}}{\boldsymbol{f_x}} \qquad \boldsymbol{eq.5}$$

where u is the horizontal pixel coordinate, c_x is the image centre, Z is the decoded depth, and f_x is the camera focal length derived from the field of view:

$$\boldsymbol{f_x} = \frac{\boldsymbol{width}}{\left(\mathbf{2} * \mathbf{tan}\left(\frac{\boldsymbol{FOV}}{\mathbf{2}}\right)\right)} \qquad \boldsymbol{eq.6}$$

The lateral offset is then corrected for the ego vehicle's yaw drift relative to the lane direction and for road curvature using a parabolic curvature model for road geometry compensation:

$$\boldsymbol{d_{corrected}} = \boldsymbol{d_{lateral}} + \left(\boldsymbol{Z} * \boldsymbol{sin}\left(\boldsymbol{\Psi_{yaw}}\right)\right) - \left(\boldsymbol{\kappa} * \boldsymbol{Z^2}\right) \qquad \boldsymbol{eq.7}$$

Where Ψ_{yaw} is the heading error and κ is the planned curvature coefficient. The corridor width applied to $d_{corrected}$ differs between the two camera tiers and depends on the LCA operating mode. The bumper camera uses a fixed corridor of 1.8 m in all scenarios. The roof camera uses a dynamic depth-dependent corridor when LCA is enabled (Configuration 2–4):

$$\boldsymbol{dynamic_{width}} = \mathbf{min}\left(\mathbf{1.6} + \left(\frac{\boldsymbol{Z}}{\mathbf{50}}\right) * \mathbf{0.8}, \mathbf{2.6}\right) \qquad \boldsymbol{eq.8}$$

where Z is the pixel depth in meters. At 10 m depth, the corridor is 1.76 m; at 50 m, it expands to 2.4 m; capped at a maximum of 2.6 m. When LCA is disabled (Configuration 0 and 1), the roof camera also uses the flat 1.6 m corridor. The dynamic expansion compensates for the increasing lateral projection error of the single-waypoint parabolic curvature model at longer ranges, where the estimated road curvature diverges from the actual curved path by 2–3 meters on Town 04's tighter bends. Without this expansion, the roof camera experienced a dropout rate of approximately 10% on curves, as legitimate in-lane vehicles at 40–60 m depth were rejected by the tight corridor.

- **Bumper Fusion (2–30 m):** The bumper-mounted semantic and depth cameras (400×300, 100° FOV) process the full frame width with an ROI spanning 15–95% of the frame height. A minimum of 100 vehicle-class pixels must be detected in the semantic mask before lane filtering is applied. After the 3D lateral projection removes out-of-lane pixels, a minimum of 50 valid in-lane vehicle pixels at depths between 0.1 m and 30.0 m is required, and the 10th percentile depth is selected to represent the nearest point of the detected vehicle.
- **Roof Fusion (10–80 m):** The roof-mounted cameras (800×600, 90° FOV) process the central vertical band (25–75% height) across the full width. A minimum of 30 vehicle-class pixels must be present in the semantic mask before processing. After lane filtering, a minimum of 20 valid in-lane vehicle pixels at depths between 10.0 m and 80.0 m is required, using the 10th percentile depth. The dynamic corridor expansion for the roof tier means that at long range, the effective detection envelope is wider, which increases the false positive rate (adjacent-lane vehicles bleeding into the corridor) but critically reduces the dropout rate on curved highway segments.

The fusion system combines these tiers using linear interpolation in the 15–25 m overlap zone: for bumper distances between 15.0 and 25.0 m, a blending weight

$$w = \frac{d_{bumper} - 15}{10} \qquad eq.9$$

transitions smoothly from full bumper trust to full roof trust. Below 15.0 m the bumper reading is used exclusively; above 25.0 m the roof reading is used exclusively. A 5-sample median filter provides temporal smoothing. The radar provides an independent distance channel. The fusion priority logic operates as follows: if the visual fusion detects a vehicle (distance < 60 m), the fusion distance is used as the primary measurement; if visual fusion fails, the system falls back to radar-only detection with cascading bumper verification for close-range radar returns

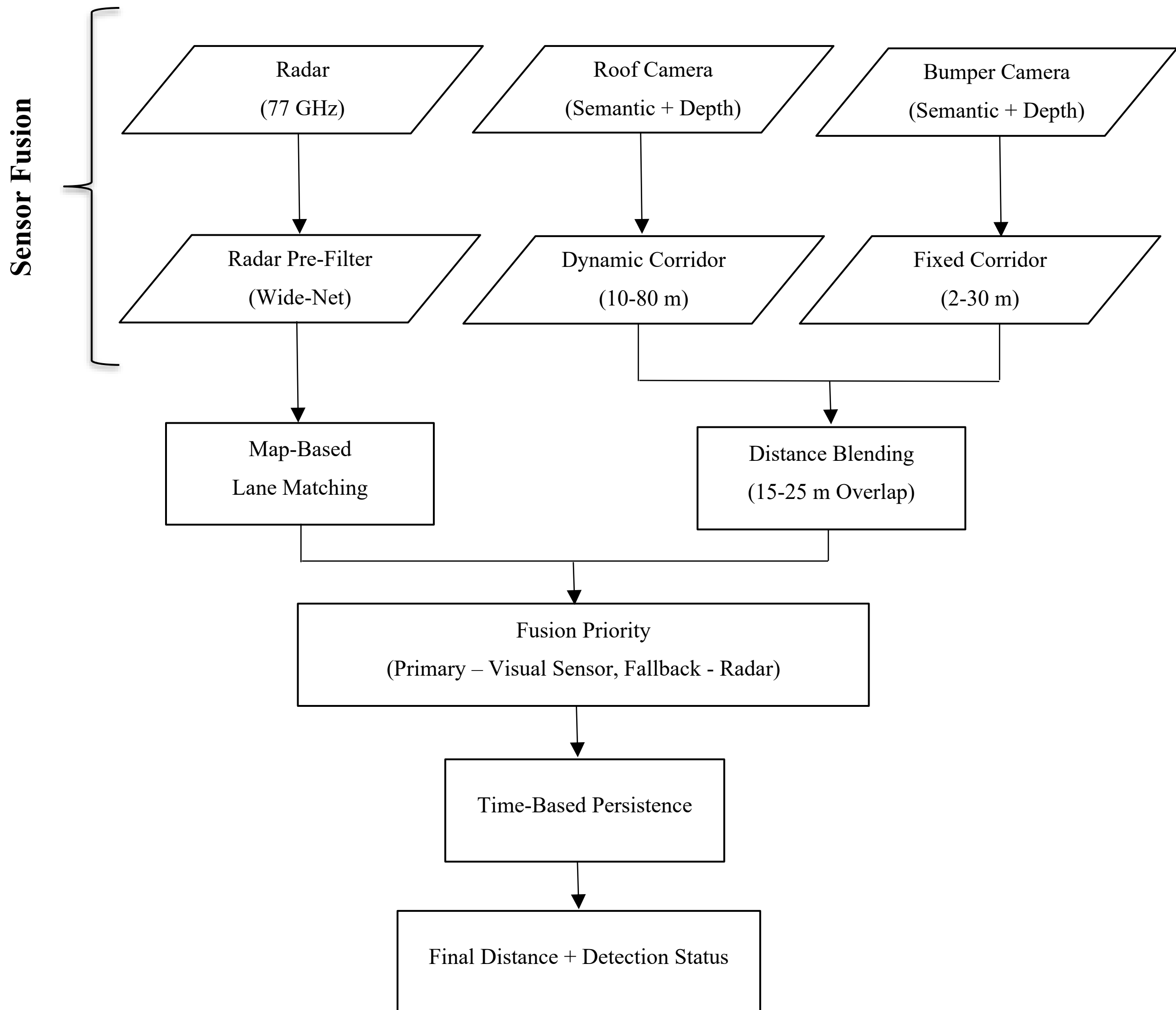

Figure 15: Sensor Fusion Pipeline

4.1.3 Time-Based Fusion Persistence

A critical challenge in real-time perception is handling transient sensor dropouts caused by weather noise, temporary occlusion, or sensor saturation. To address this, the system implements a time-based persistence mechanism that maintains the last valid detection for a configurable duration after the sensor fusion loses tracking. This mechanism is FPS-independent, operating in real seconds rather than frame counts (Appendix B-3a):

- **Normal Persistence (0.17 seconds):** During standard highway driving, the system retains the last valid distance measurement for 170 milliseconds after a dropout, bridging single-frame sensor glitches.
- **ACC Follow Persistence (1.0 second):** When the ACC was actively following a lead vehicle, the persistence window extends to 1.0 second, preventing dangerous acceleration during brief rain-induced dropouts.
- **Traffic Light Persistence (1.5 seconds):** When the vehicle was stopping for a red traffic light and a lead vehicle was detected, the persistence extends to 1.5 seconds to prevent the vehicle from accelerating into stopped traffic at an intersection during a momentary sensor blackout.

A minimum detection duration threshold of 0.067 seconds (approximately 1.3 frames at 20 FPS) is enforced before any detection is considered valid, filtering out single-frame false positives. The persistence mechanism tracks two time-based state variables:

1. fusion_detection_duration - cumulative time the sensor has been actively detecting
2. time_since_detection - elapsed time since the last valid detection

When the persistence window expires and fusion detection has been fully lost, the system falls back to radar-only detection with a cascading verification mechanism. For radar detections beyond 22 meters, the radar's own azimuth-based lane filtering is trusted directly. However, for close-range radar detections below 22 meters, where false positives from guardrails and bridge supports are most common the system requires independent confirmation from the bumper camera fusion before committing to a vehicle presence. If the bumper fusion distance also reports a vehicle within 22 meters, the radar detection is accepted and the dropout timer (time_since_detection) is reset to zero, restoring normal persistence behavior. This cascading verification approach replaced an earlier safety-persist mechanism that unconditionally maintained detection for up to 5 seconds when the ACC had been actively following a vehicle. The safety-persist approach was found to cause phantom braking when the lead vehicle changed lanes and all sensors correctly reported no vehicle, but the safety-persist override artificially maintained the stale detection. The cascading verification

provides equivalent protection against genuine sensor dropouts (where the radar continues to see the vehicle even when camera fusion is temporarily degraded) while correctly releasing detection when the lead vehicle genuinely departs the ego lane. (Appendix B-3b)

The decision to implement persistence in absolute time (seconds) rather than frame counts was driven by observed Real-Time Factor (RTF) variability across weather conditions. During testing, the simulation's effective frame rate fluctuated between 0.7x and 1.0x real-time depending on the computational load imposed by weather rendering (e.g., rain particle effect calculations) and the number of active sensor streams. A frame-count-based persistence of, for example, 5 frames would correspond to 0.25 seconds at 20 FPS under ideal conditions but would stretch to 0.357 seconds when RTF dropped to 0.7x. This inconsistency meant that the same persistence logic behaved differently across weather presets precisely the conditions the thesis aims to compare. By expressing all timing constants in seconds and scaling by the actual simulation time delta (dt), the persistence mechanism behaves identically regardless of computational load, ensuring that observed differences in safety metrics are attributable to weather degradation rather than timing artifacts.

The minimum detection duration threshold of 0.067 seconds was introduced to suppress single-frame phantom detections caused by rain noise. During heavy rain scenarios, water droplet artifacts occasionally produced clusters of vehicle-colored pixels in the semantic segmentation output that passed the 100-pixel minimum area threshold for a single frame before disappearing. Without the minimum duration filter, these transient false positives would trigger the ACC's emergency braking logic - a single phantom detection at close range that would produce a TTC below 1.5 seconds and activate hard braking on an open highway. By requiring sustained detection for at least 0.067 seconds (slightly over one frame at 20 FPS) before the system commits to a vehicle presence, these weather-induced false positives are filtered without introducing meaningful latency for genuine detections, which typically persist for hundreds of consecutive frames (Appendix B-3b).

4.1.4 Three-Zone ACC Control Strategy

The ACC implements a time-gap-based three-zone control strategy. The time gap is calculated as:

$$t_{gap} = \frac{d_{measured}}{v_{ego}} \qquad eq. 10$$

where $d_{measured}$ is the measured distance to the lead vehicle and g v_{ego} is the ego vehicle speed in m/s. The target gap is 2.5 seconds with a ±0.3-second deadband, creating three zones:

- **Zone A – Holding Zone (2.2–2.8 s):** Minor throttle modulation (0.1–0.3) maintains the current gap. The system applies gentle throttle if the gap is drifting open or gentle braking if closing.
- **Zone B – Closing Zone (>2.8 s):** The gap is too large; throttle is increased proportionally to the gap error with an intensity range of 0.4–0.8, capped at a normalized gap error of 1.5 seconds.
- **Zone C – Opening Zone (<2.2 s):** The gap is dangerously small; braking is applied with intensity 0.2–0.5 proportional to the absolute gap deficit, capped at 1.0 second of deficit.

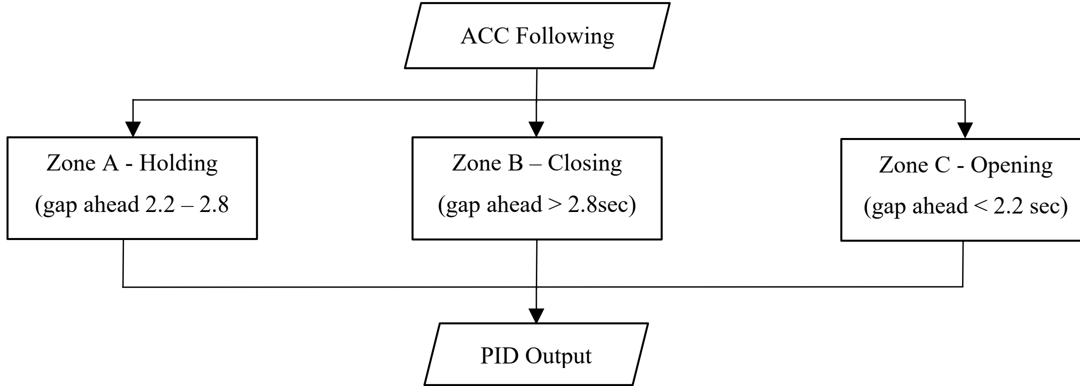

Figure 16: ACC Following Schematic Diagram

Above these zones, a priority-based emergency hierarchy overrides the three-zone logic. If the time gap falls below 0.75 seconds, an emergency braking mode activates with braking intensity scaled by urgency:

$$\boldsymbol{urgency = clip\left(\left(1.2 - t_{gap}\right)/0.5, 0.5, 1.0\right)} \qquad \textbf{eq. 11}$$

If the absolute distance falls below 5.0 meters while travelling above 10 km/h, a hard stop is executed with full braking force. During emergency braking (Priorities 1–3), the handbrake is explicitly disabled (hand_brake = False) to preserve directional stability. At highway speeds, engaging the handbrake locks the rear axle, which eliminates rear-wheel lateral grip and causes the vehicle to yaw uncontrollably, a phenomenon analogous to real-world rear-wheel lockup. The handbrake is only engaged during low-speed stationary holds: when the vehicle is stopped behind another car at a red light (Priority 4) or holding position at a stop line (Priority 6), where the vehicle is already at or near zero speed and directional stability is not a concern. When the vehicle is nearly stationary (≤ 3.0 km/h) and a close-range detection persists below 5.0 meters, the system performs a secondary semantic verification step: the roof-mounted semantic and depth cameras are queried to confirm the presence of vehicle-class pixels in the ego lane using the 3D lateral projection. If the semantic verification fails to confirm a vehicle, the detection is cleared and the system resumes

cruise mode. This prevents the ACC from becoming permanently locked in a stopped state due to residual sensor noise or false positive detections at very close range (Appendix B-4).

When a red traffic light is detected simultaneously with a lead vehicle ahead, the system uses the closer of the vehicle distance (minus a 2.5 m safety buffer) and the stop-line distance as the effective stopping target, ensuring the ego vehicle does not overshoot into the rear of a car stopped at the intersection

The three-zone control strategy was adopted after observing that conventional single-PID ACC architecture produced oscillatory behavior at the boundary between cruising and following modes. When a single PID controller was used to regulate both speed (when no vehicle was detected) and distance (when following), the transition between these two objectives caused abrupt throttle-to-brake switching at the detection boundary—particularly during sensor fusion dropouts where the measured distance would momentarily jump from a valid reading to 100 m (no detection) and back. The three-zone approach with a ±0.3-second deadband eliminates this boundary oscillation by creating a hysteresis region where neither aggressive throttle nor aggressive braking is applied. Furthermore, the priority-based emergency hierarchy (absolute distance < 5 m overrides time gap < 0.75 s, which overrides the three-zone logic) ensures that safety-critical situations are handled with deterministic, maximum-effort responses rather than proportional PID outputs that may react too gradually to sudden braking events from the lead vehicle.

When the ACC transitions from vehicle-following mode to free cruise typically because the lead vehicle has exited the sensor envelope on a curve and the system does not immediately accelerate toward the cruise target speed. Instead, a safe cruise transition mechanism monitors the time since the ACC was last active via a tick counter (ticks_since_acc). If fewer than 20 ticks have elapsed since the last ACC or emergency braking event and the vehicle speed exceeds 15 km/h, the system enters a COAST mode: throttle is set to zero and a gentle braking force of 0.10 is applied. This produces a brief deceleration phase of approximately one second before the cruise PID is permitted to accelerate. The COAST mode was introduced after observing that immediate cruise acceleration following a sensor dropout caused the ego vehicle to rapidly build speed toward 150 km/h while potentially re-entering the detection envelope of the same lead vehicle creating a dangerous oscillation between emergency braking and full acceleration. The one-second coast buffer allows the sensor fusion persistence mechanism to fully expire before committing to cruise acceleration, preventing phantom-braking oscillation cycles.

4.2 Lane Centering Assist (LCA)

The Lane Centering Assist system maintains lateral centering by computing waypoint-based steering corrections with dynamic speed adaptation (Appendix B-5). At each simulation step, the system queries the CARLA HD map for a lookahead waypoint ahead of the vehicle's current position. In autonomous mode, a fixed 12.0-metre lookahead is used when LCA is enabled; when LCA is disabled (Configuration 0 and 1), an 8.0-metre lookahead with reduced gain (×0.45) and a steering deadband of 0.04 simulates less-precise manual-like lane keeping. The deliberately short 8.0-metre lookahead causes the steering to react to curves very late, the low 0.45 gain produces sluggish corrections, and the wide 0.04 deadband allows the vehicle to drift noticeably before any correction is applied. This combination physically limits the vehicle's ability to maintain stable high-speed trajectories on curves, which as demonstrated in the results inadvertently provides a self-limiting safety effect by preventing the ego vehicle from sustaining the 120+ km/h speeds that exceed the sensor fusion detection envelope. In both cases, the lookahead defaults to a dynamic calculation clipped between 12.0 and 65.0 meters based on the vehicle's speed:

$$\boldsymbol{lookahead = clip(v_{ego} * 0.4, 12.0, 65.0)} \qquad \boldsymbol{eq. 12}$$

The steering correction is calculated as the signed angle between the vehicle's forward vector and the target waypoint direction:

$$\boldsymbol{\delta_{ideal} = atan^2(wp_y - ego_y, wp_x - ego_x) - \theta_{ego}} \qquad \boldsymbol{eq. 13}$$

A speed-dependent steering gain reduces sensitivity at higher speeds to prevent overcorrection - gain = 1.2 for speeds below 40 km/h, decreasing linearly to 0.35 at 150 km/h,

$$\boldsymbol{gain = max(0.35, 1.2 - (v_{ego} - 40) * 0.008)} \qquad \boldsymbol{eq. 14}$$

An angular damping factor further attenuates corrections at large steering angles to prevent oscillation on gentle curves:

$$\boldsymbol{damping = 1.0 - \left(\frac{\theta}{\pi}\right) * 0.3} \qquad \boldsymbol{eq. 15}$$

The LCA incorporates two highway-specific safety mechanisms. First, a highway road ID isolation check: at simulation startup, the highway's road IDs are traced dynamically and stored. If the current waypoint's road ID is not in this set (indicating the ego has drifted onto an off-ramp or service road), the system shifts the reference waypoint laterally back onto the highway, pulling the vehicle away from the diverging path. Second, a straightest-path filter: at road splits where multiple forward waypoints exist, the system selects the waypoint with the smallest absolute angular

deviation from the ego's current heading, preventing the steering from being drawn toward exit ramps or junction branches (Appendix B-5) (Coulter & R.C., 1992; Jaimin K, 2021).

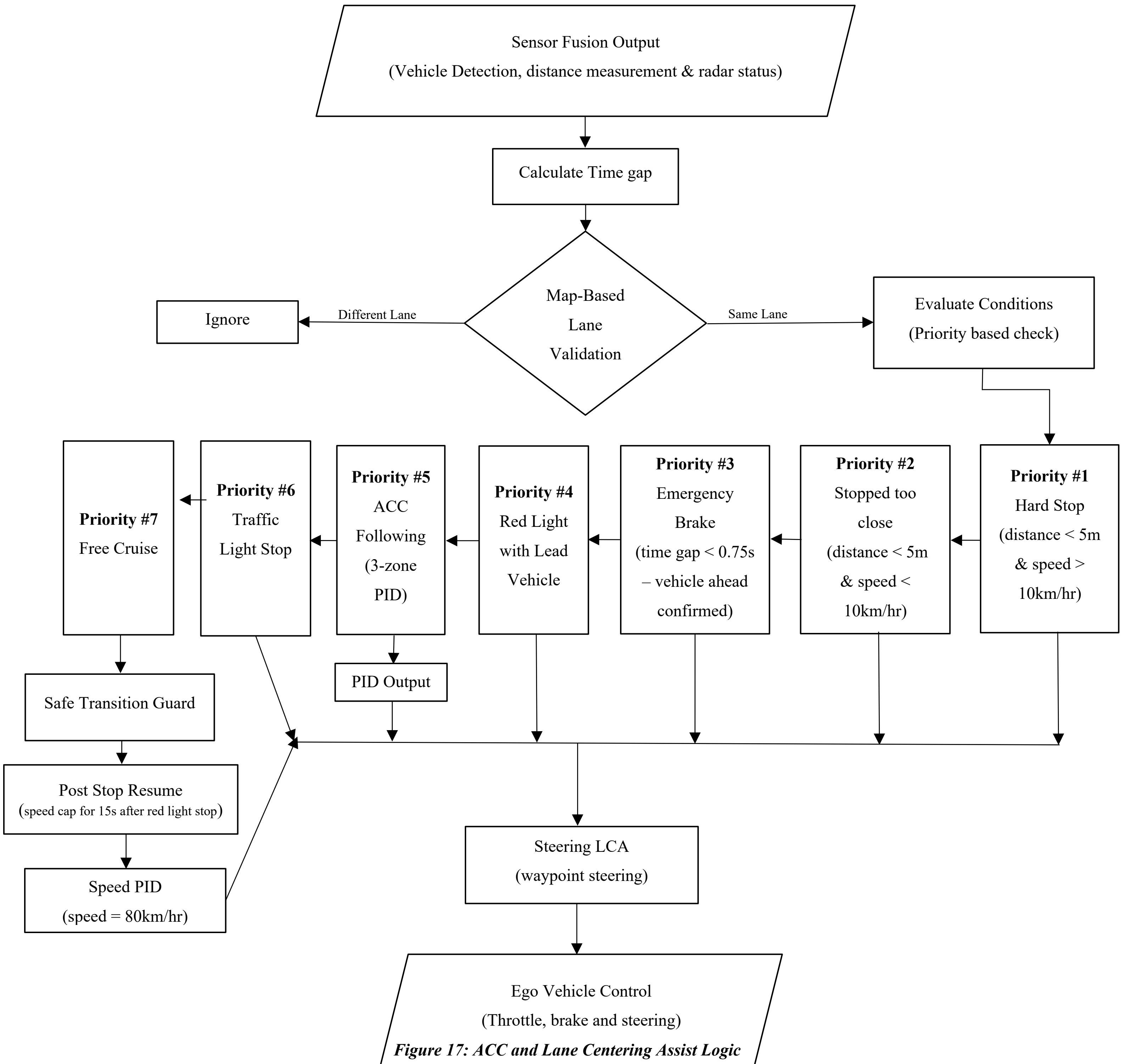

Figure 17: ACC and Lane Centering Assist Logic

4.3 Traffic Light Detection and Automated Braking

The traffic light assistance system employs a dual-method detection approach combining map-based ground truth with vision-based heuristic verification.

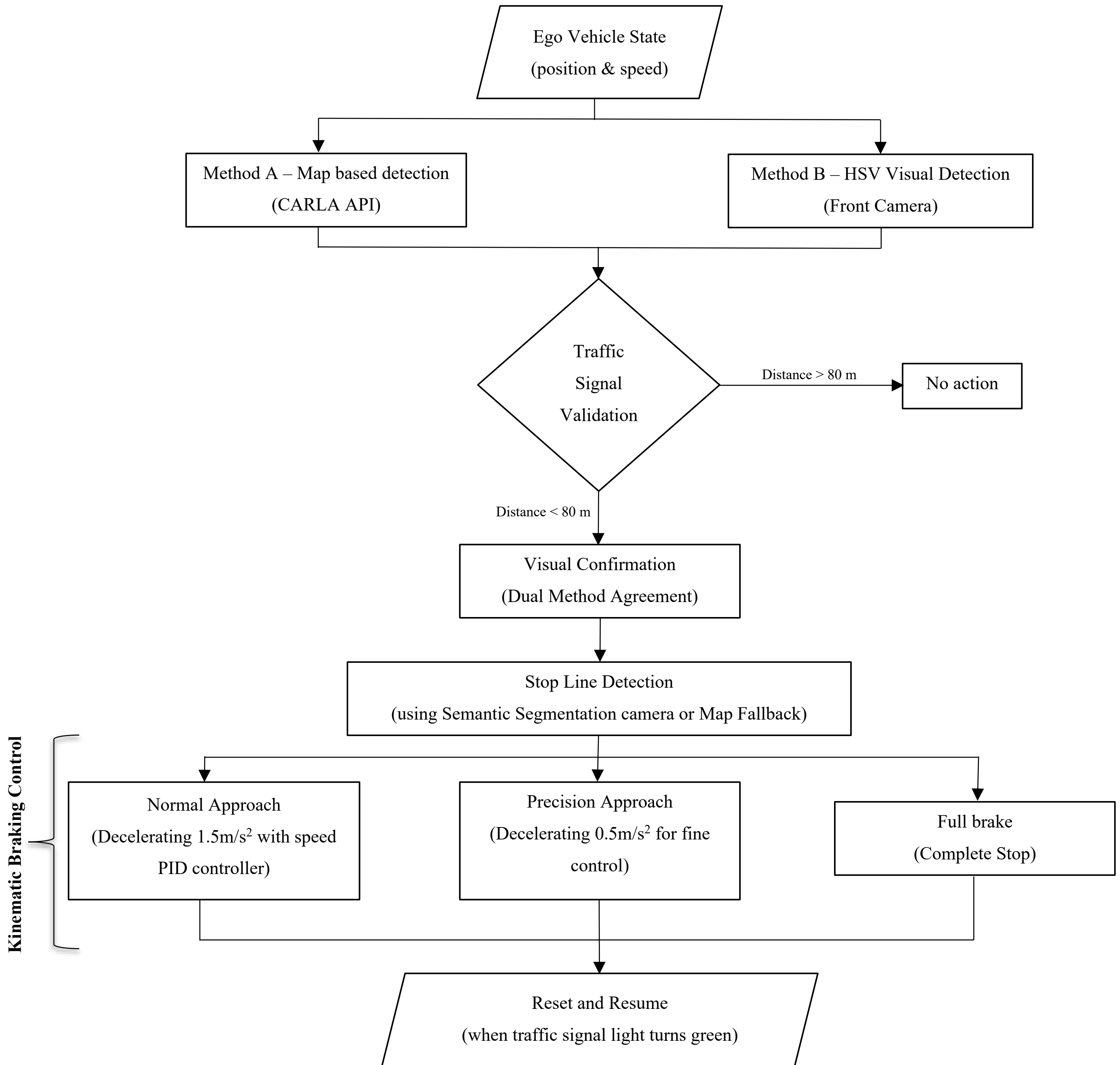

Figure 18: Traffic Light Detection Schematic Diagram

4.3.1 The Logic Adopted

The map-based detector queries CARLA's traffic light actors within a dynamic scan distance calculated as:

$$d_{scan} = \max\left(80.0, \frac{v_{ego}}{3.6} * 6.0\right) meters \qquad \text{eq. 16}$$

At 80 km/h, this provides approximately 135 meters of forward scanning distance. The system identifies traffic lights using their CARLA actor state (Red, Yellow, Green) and calculates the distance to the associated stop line using waypoint projection (Appendix B-6a).

The RGB-based heuristic detector processes the front camera image using HSV color space analysis. The upper 60% of the image is extracted as the region of interest. Red pixels are detected using dual HSV ranges: H: [0–10, 170–180], S: [80–255], V: [80–255]. Yellow pixels use H: [20–35], S: [80–255], V: [80–255]. Contour analysis filters detections by area (3–1000 pixels) and aspect ratio (0.3–2.0) to reject sun glare and other false positives (Appendix B-6b) (Shrivastava et al., 2023).

When a red light is detected, the system calculates the required deceleration using kinematic equations. For distances greater than 5 meters, the required approach speed is:

$$v_{required} = \sqrt{2 * a_{decelerate} * d_{stop}} \qquad eq. 17$$

where $a_{decelerate}$ is 1.5 m/s² for normal approach and 0.5 m/s² for final creep below 5 meters. Braking force is proportional to the excess speed:

$$brake = clip\left(\frac{v_{ego} - v_{required}}{2.5}, 0.3, 1.0\right) \qquad eq. 18$$

The dual-method detection approach (map-based + HSV heuristic) was necessitated by the complementary failure modes of each method in isolation. The map-based detector, which queries CARLA's traffic light actors directly, provides reliable state identification (Red, Yellow, Green) and precise stop-line distance calculation. However, it depends on the simulation's internal traffic light state updates, which can lag visual changes during rapid light cycling. Conversely, the RGB-based HSV heuristic provides real-time visual confirmation but is susceptible to false positives from low-angle sunlight at sunrise conditions, where the sun's color temperature overlaps with the red/yellow detection thresholds. During sunrise testing, the HSV detector alone produced false red-light triggers when the sun appeared in the upper portion of the camera frame, causing the vehicle to brake on the open highway. The area filter (3–1000 pixels) and aspect ratio constraint (0.3–2.0)

were specifically tuned to reject these solar false positives: the sun subtends a much larger angular area than a traffic light at detection distance, while its aspect ratio is approximately 1.0 regardless of position. The combination of both methods, where the map-based detector provides the primary trigger and the HSV heuristic provides visual confirmation, significantly reduces both false negatives (missed lights) and false positives (phantom braking events).

Figure 19: Front camera approaching a red light, ego vehicle braking

4.3.2 Post-Stop Resume Protocol

Critical vulnerability was identified during preliminary testing of Configuration 3 (ACC + LCA + TLA): after the ego vehicle stopped at a red traffic light and the light turned green, the vehicle would accelerate toward its unrestricted cruise while the traffic that had been ahead during the red phase was still within the first curve of the highway loop. Because the sensor fusion dropout rate increases on curved segments, the ego vehicle would frequently lose detection of the slow-moving traffic ahead and approach at closing speeds exceeding 80 km/h, well beyond the emergency braking envelope.

To mitigate this, a post-stop resume protocol was implemented. A timer variable (time_since_light_stop) tracks the elapsed time since the ego vehicle last stopped at a red traffic light. The timer resets to zero when the vehicle is stationary and a red light stop is active and increments by the simulation time delta (dt) on every subsequent tick. Within the PRIORITY 7: FREE CRUISE control branch, if the traffic light assist is enabled and time_since_light_stop is less than 15.0 seconds, the cruise target speed is capped at min(target, 80.0 km/h) instead of the unrestricted speed. The timer is initialized at 999.0 seconds within the Vehicle State Tracker class to ensure no false speed cap is applied before the first red light stop (Appendix C-6).

This protocol is scoped exclusively to Configurations 3 and 4, where traffic light assist feature is active. Configurations 0–2 do not stop at red lights and therefore never trigger the timer. The 15-second window was calibrated to cover the typical distance between the traffic light intersection and the first highway curve where dropout-induced collisions were observed. The protocol is temporal rather than spatial, which expires after 15 seconds regardless of the ego vehicle's position, which means it cannot protect against collisions that occur beyond the 15-second window on longer straight-to-curve transitions. This limitation is reflected in the results, where Configuration 3 under Clear Sky – Noon conditions still produced one collision at T:384 seconds, occurring 86 seconds after the most recent light stop.

Chapter 5: Experimental Conditions

The primary focus of this work is to see the impact of weather and different times of day have on the sensor perception and overall driving ability of an otherwise conditionally autonomous vehicle.

5.1 Weather Degradation Model

The camera degradation system applies weather-dependent noise to simulate real-world environmental effects on image sensors. The system supports optional GPU acceleration using CuPy for parallel noise generation, with a NumPy fallback for CPU-only environments. A cockpit mask is applied to restrict degradation to the visible windshield area (upper 85% of the image). The degradation model includes additive Gaussian noise scaled by precipitation intensity and reduced brightness for night conditions (Appendix C-1).

5.2 Experimental Test Matrix and Scenarios

The evaluation is conducted through a matrix of 30 distinct scenarios, composed of 6 weather conditions × 5 ADAS functional stages.

5.2.1 Weather and Time of Day

Testing is distributed across six atmospheric conditions defined in the Weather Manager class. These represent the combination of two weather types during three different times of day (Appendix C-2):

- **Clear Sky - Sunrise:** Cloudiness 10%, no precipitation, sun altitude 15°. This condition tests the camera suite's resilience to low-angle glare and high dynamic range scenes.
- **Clear Sky - Noon:** Cloudiness 10%, no precipitation, sun altitude 75°. Optimal illumination conditions serve as the baseline for maximum sensor performance.
- **Clear Sky - Night:** Cloudiness 10%, no precipitation, sun altitude −30°. Tests the system's reliance on active headlight illumination and its impact on camera-based detection range.
- **Heavy Rain - Sunrise:** Cloudiness 100%, precipitation 100%, wetness 100%, wind 80%, fog density 10, fog distance 20 m, sun altitude 15°. Combines rain scatter with low-angle glare for maximum optical degradation.
- **Heavy Rain - Noon:** Cloudiness 100%, precipitation 100%, wetness 100%, wind 80%, fog density 10, fog distance 20 m, sun altitude 45° (reduced from 75° due to cloud cover). Tests compounding rain and diffused lighting effects.

- **Heavy Rain - Night:** Cloudiness 100%, precipitation 100%, wetness 100%, wind 80%, fog density 10, fog distance 20 m, sun altitude −30°. The most challenging condition combining rain, fog, and darkness represents the worst-case scenario for the sensor suite.

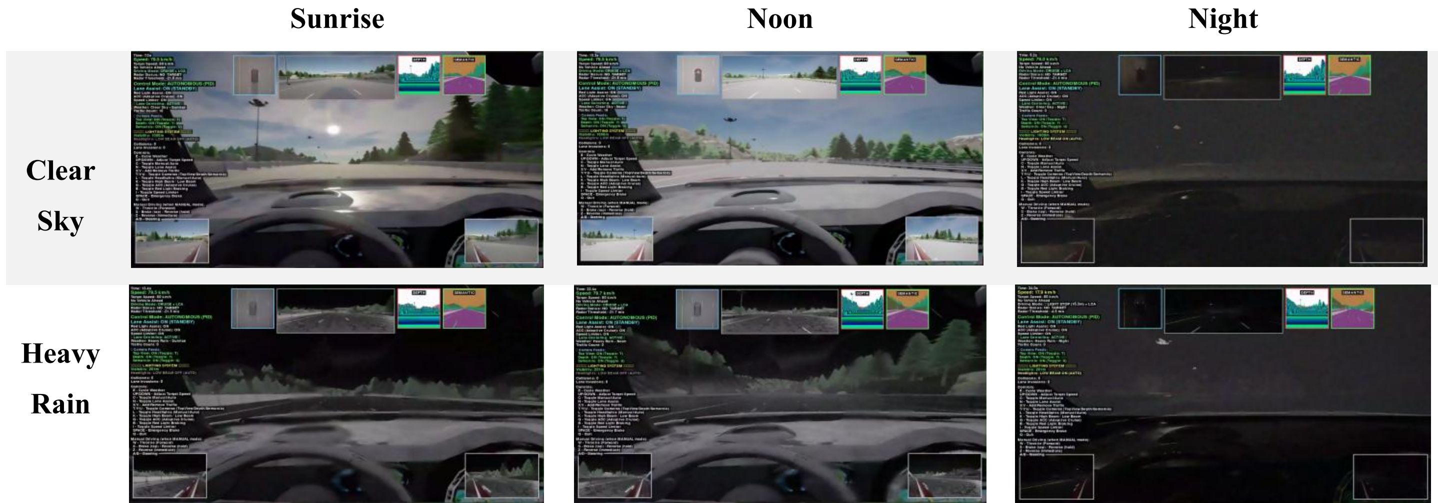

Figure 20: **Comparing different weather conditions from the front camera**

5.2.2 Complete ADAS Configuration Matrix

Within each weather condition. the vehicle is evaluated through five functional stages, each adding an incremental ADAS capability. The configuration-to-feature mapping is as follows:

Table 1: Testing Configurations

ADAS Configuration	ACC	LCA	Red Light Assist	Speed Limiter	Description
Configuration #0	OFF	OFF	OFF	OFF	Baseline (No ADAS)
Configuration #1	ON	OFF	OFF	OFF	ACC Only
Configuration #2	ON	ON	OFF	OFF	ACC + LCA
Configuration #3	ON	ON	ON	OFF	ACC + LCA + Red Light
Configuration #4	ON	ON	ON	ON	Full ADAS

In ADAS configuration 0–3, the unrestricted cruise target speed is 150 km/h, representing the vehicle's maximum acceleration capability without external speed governance. In configuration 4, the speed limiter caps the target speed at 80 km/h, corresponding to the Italian highway speed limit for the vehicle class under test. This distinction is critical: the 150 km/h cruise target means that during sensor dropout events on curves, the ego vehicle in configuration 1–3 can reach closing speeds that physically exceed the emergency braking envelope, while the 80 km/h cap in

configuration 4 ensures that even worst-case closing speeds remain within the braking distance available from the sensor detection horizon.

This incremental ADAS staging is conceptually grounded in an upper/lower bound evaluation framework established during the preliminary research phase, where minimum capability (emergency braking only) defines the safety floor and the full Level 3 package defines the performance ceiling. This yields a total of $6 \times 5 = 30$ unique test configurations. Each scenario is executed with 20 fixed-position background traffic vehicles (15 ahead, 5 behind) to ensure reproducibility.

5.3 Test Environment in Town 04

Town 04 provides a mountainous landscape with a unique "figure of 8" highway configured for continuous loop testing (CARLA Documentation, 2024). The ego vehicle spawns at coordinates (310.0, 5.0) with a lane shift of −2 (far left lane) and 0° yaw. The highway road IDs are traced dynamically at startup by the trace highway road ids function, which follows waypoints for 5000 meters at 5-meter intervals to identify all road segments belonging to the highway loop (Appendix C-3). This set of highway road IDs is used to verify that the ego vehicle remains on the highway and to classify lane invasions as highway relevant.

5.4 Traffic Configuration

The thesis supports two distinct traffic configurations, selectable via command-line arguments:

- **Random Traffic (default):** Up to 80 background actors are spawned using the CARLA Traffic Manager with stochastic behaviors including Gaussian speed variations and randomized path choices at junctions. The Traffic Manager's "floating bubble" logic dynamically spawns and destroys vehicles around the ego to maintain constant traffic density (CARLA Documentation, 2024). If the initial spawn falls short due to spawn-point collisions, a fallback mechanism gradually spawns additional vehicles over subsequent simulation ticks until the target count is reached.
- **Fixed Traffic:** Up to 20 vehicles are spawned at predefined highway waypoint positions for 100% reproducible testing. Vehicles are placed in both the driving lane and the passing lane at regular intervals along the highway, with deterministic, repeatable paths to allow for the assessment of control system responses to controlled conflict events (Appendix C-4).

Table 2:Spawn Point Coordinates for Fixed Traffic Configuration – Ahead of the test vehicle

Sr#	Waypoint (m)	Lane offset
1.	60.1	1
2.	100.0	-1
3.	150.0	0
4.	200.0	1
5.	250.0	0
6.	300.0	-1
7.	350.0	-1
8.	400.0	0
9.	460.0	-1
10.	520.0	0
11.	580.0	1
12.	650.0	0
13.	720.0	-1
14.	800.0	1
15.	900.0	0

Table 3:Spawn Point Coordinates for Fixed Traffic Configuration – Behind the test vehicle

Sr#	Waypoint (m)	Lane offset
1.	-50.0	1
2.	-120.0	0
3.	-200.0	1
4.	-300.0	1
5.	-400.0	0

5.5 Input Parameters for Simulation

The simulation is executed with the following fixed input parameters across all 30 test runs:

Table 4: Input Parameters for each simulation run

Sr.#	Parameter	Value	Description
1	Simulation Time Step	0.05 sec	Corresponds to 20 FPS tick rate
2	Target RTF	1.00	Real Time Factor
3	Simulation Duration	500 sec	Time for each simulation run
4	Spawn Location	310.0 , 5.0	Ego Vehicle spawn location
5	Spawn Yaw	0.0°	Ego Vehicle spawn rotation
6	Lane Shift	Lane -2	Ego Vehicle spawn lane
7	Target Speed	80 km/hr	Only applicable when configuration #4 of ADAS is enabled
8	Crash Snapshot	Enabled	PNG generation for each collision event

These parameters remain constant across all 30 test configurations. The only variables between runs are the weather condition and the ADAS configurations.

5.5.1 Real-Time Performance Optimization

Achieving a stable Real-Time Factor (RTF) of 1.00× where one simulation second corresponds to one wall-clock second - was essential for ensuring that the time-based persistence mechanism and the post-stop resume timer behaved consistently across all 30 test runs. During initial development, the twelve-sensor suite combined with weather degradation processing produced an RTF of approximately 0.6–0.7×, meaning the simulation ran 30–40% slower than real time. This inconsistency would have invalidated cross-weather comparisons, as the persistence windows would effectively stretch under higher computational loads. Five optimization layers were implemented to achieve consistent 1.00× RTF:

First, parallel display rendering: a Thread Pool Executor with five worker threads processes the weather degradation for all five RGB camera feeds (front, left, right, rear, and bird's-eye view) simultaneously rather than sequentially. Since the degradation function applies independent

Gaussian noise to each image, the five operations are embarrassingly parallel and produce a near-linear speedup.

Second, GPU-accelerated weather degradation with mask caching: the CuPy library offloads the noise generation and brightness adjustment computations to the GPU. Additionally, the cockpit mask a binary image that restricts degradation to the windshield area is computed once and stored in a global cache (MASK_CACHE), keyed by image dimensions. This eliminates redundant mask generation for the approximately 10,000 frames processed per simulation run.

Third, traffic light geometry pre-caching: before the main simulation loop begins, all traffic light actors are queried for their affected lane waypoints and stop-line locations. These geometric properties are stored in a light cache list, allowing the traffic light detection function (Section 4.3) to perform simple distance calculations against cached coordinates rather than issuing per-tick CARLA API queries that incur network round-trip latency.

Fourth, local ground truth calculation: rather than querying the CARLA server for nearest-vehicle distance each tick (which requires a server-side spatial query), the system iterates through the local list of traffic vehicles maintained by the Traffic Manager and computes Euclidean distances client-side. This eliminates the most expensive per-tick RPC call.

Fifth, garbage collection management: Python's automatic garbage collector is disabled at the start of the simulation loop (via gc.disable()). The collector's non-deterministic activation during time-critical ticks caused sporadic RTF drops to 0.7× lasting 50–100 milliseconds. With automatic GC disabled, the system runs manual garbage collection during the RTF pacing sleep window the idle time between simulation tick completion and the next tick deadline, where the latency is absorbed without affecting simulation timing.

These five layers collectively reduced per-tick processing time from approximately 70 ms to under 50 ms, achieving a stable RTF of 1.00× with less than 0.2% of ticks falling below 0.9× under clear weather conditions. Night-time conditions with rain produce the highest computational load due to combined headlight rendering and noise generation, resulting in a mean RTF of 0.99× with approximately 5–6% of ticks below 0.9×, as reflected in the simulation results.

5.6 Data Extraction and Performance Metrics

Quantitative results are extracted from simulation logs at each simulation tick and aggregated by the DegradationMetrics class (Appendix C-5).

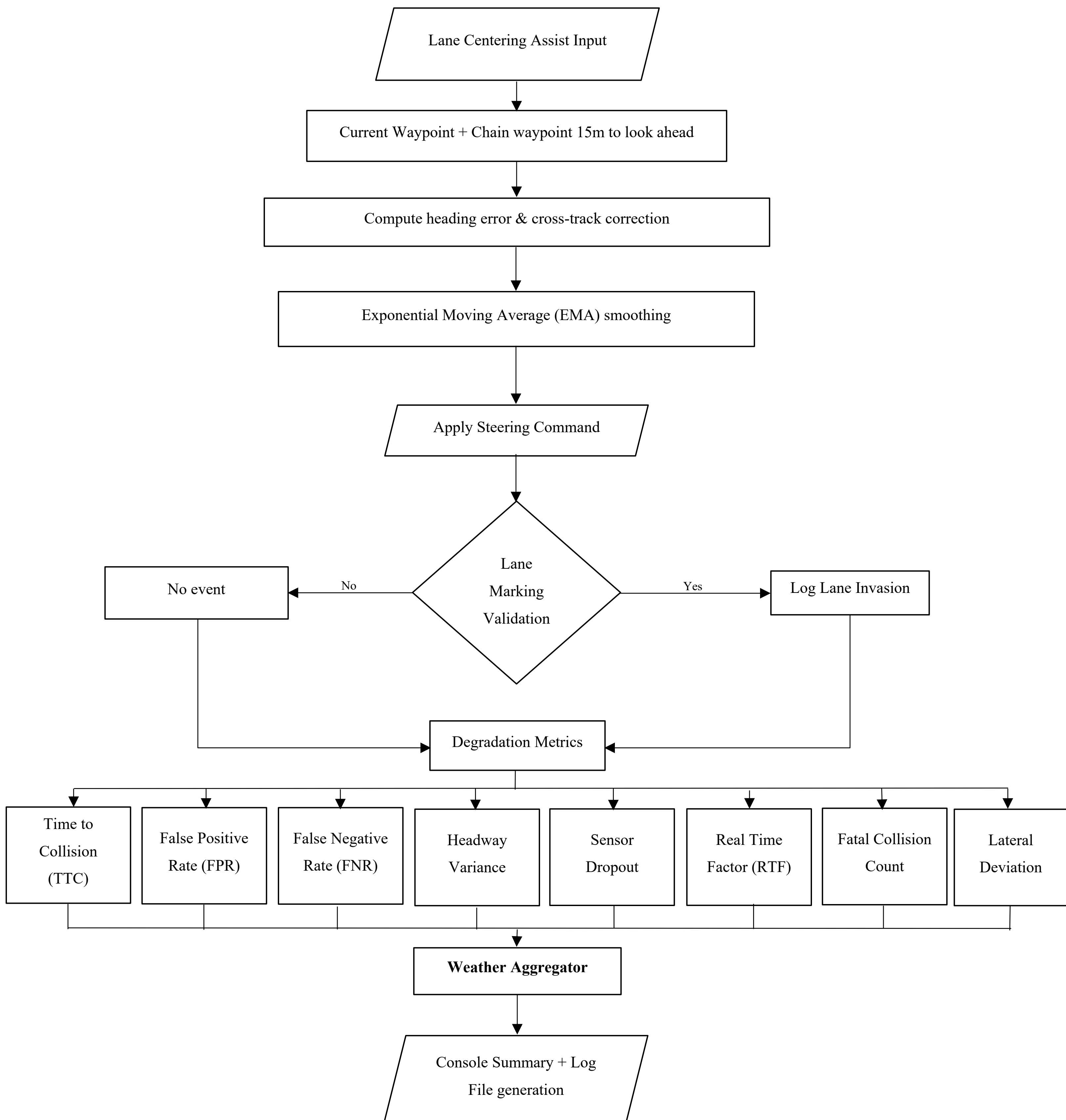

Figure 21: Safety Metrics Collection and Per-Weather Aggregation Pipeline

The following metrics are computed both globally and per-weather condition:

- **Time-to-Collision (TTC):** Computed every tick when a vehicle is detected and the ego speed exceeds 1.0 m/s. Uses radar Doppler relative velocity when available, otherwise falls back to ego speed as the closing speed estimate. The minimum TTC across the entire run is recorded as the primary safety indicator. A critical threshold of TTC < 1.5 seconds identifies dangerous conflicts (Yan et al., 2023).

$$TTC = \frac{d_{measured}}{v_{closing}}, \text{where } v_{closing} > 0.5\text{m/s} \ \& \ 0 < TTC < 30\text{sec} \qquad eq.19$$

- **Radar False Positive Rate (FPR) and False Negative Rate (FNR):** Computed using a confusion matrix approach where ground-truth vehicle presence is defined as a vehicle existing within 2–80 m of the ego vehicle.

$$FPR = \frac{FP}{(FP + TN)} * 100\% \qquad eq.20$$

$$FNR = \frac{FN}{(FN + TP)} * 100\% \qquad eq.21$$

  Ground truth distance is computed locally with each simulation tick by iterating through all traffic vehicles managed by the Traffic Manager. For each background vehicle, the system calculates the Euclidean distance to the ego vehicle, then queries the CARLA HD map to determine whether the traffic vehicle occupies the same lane as the ego vehicle (matching road id and lane id). The closest same-lane vehicle within 80 meters is recorded as the ground truth distance for that tick. If no same-lane vehicle exists within 80 meters, the ground truth distance is set to 100.0 m (indicating no valid target). This local ground truth calculation avoids server-side RPC calls that would degrade the Real-Time Factor and provides the reference baseline against which all sensor fusion measurements are compared for error metrics, FPR/FNR computation, and dropout rate classification.

- **Mean Distance Error and Headway Variance:** The absolute error between ground-truth and measured distance is recorded when both values are in the valid range (0.1–99.0 m). The mean error quantifies systematic measurement bias, while the headway variance (σ^2) quantifies control stability.

$$\sigma^2 = \left(\frac{1}{N}\right) . \sum (e_i - \mu_e)^2 \qquad eq.22$$

- **Sensor Dropout Rate:** Percentage of samples where the detection status is "NO_TARGET" or "FUSION_PERSIST", calculated as.

$$\frac{dropout_{count}}{total_{samples}} \; x\; 100\% \qquad eq.23$$

- **Real-Time Factor (RTF):** Tracked per-tick and averaged per weather condition. The percentage of ticks below 0.9x is reported as a computational stress indicator.
- **Lateral Offset from Lane Centre:** Measured every tick when the ego speed exceeds 5.0 km/h and the vehicle is not in collision recovery or post-respawn warmup. The offset is computed as the perpendicular distance between the ego vehicle's position and the nearest driving waypoint's lane center using the cross-product of the waypoint's forward vector and the ego-to-waypoint displacement vector. Three statistics are reported by weather condition: mean lateral offset (LAT_{μ}), maximum lateral offset (LAT_{max}), and standard deviation of lateral offset (LAT_{σ}). These metrics quantify the effectiveness of the Lane Centering Assist across weather conditions and directly measure the lateral control stability
- **Fatal Collision Count:** Incremented each time the collision recovery or wrong-way correction mechanism is triggered, attributed to the current weather condition.

At the end of each simulation run, a comprehensive per-weather breakdown table is printed displaying all metrics side by side, enabling direct comparison of sensor degradation across weather conditions for the given ADAS configuration.

Chapter 6: Experiment Simulations and Results

6.1 Simulation Results of the various ADAS Features

The 30-run test matrix, comprising five progressive ADAS configurations across six weather and lighting conditions, produced a comprehensive dataset of safety metrics, sensor performance indicators, and control stability measures. Each simulation run generated approximately 500 seconds of continuous driving data at 20 FPS, yielding between 2,063 and 9,763 measurement samples per run depending on the configuration's stopping behavior at traffic lights. The minimum TTC values below 0.1 seconds recorded across multiple runs correspond to low-speed stop-line approaches where the ego vehicle creeps toward a stationary lead vehicle at 2–5 km/h, rather than high-speed near-miss events.

The collision counts across the five configurations are: Configuration 0 (28), Configuration 1 (3), Configuration 2 (6), Configuration 3 (10), and Configuration 4 (0). This non-monotonic progression, where adding ADAS features does not uniformly reduce collisions, is examined in detail below and interpreted in Chapter 7. The following sections present the per-configuration results, progressing from the unassisted baseline through to the complete Level 3 ADAS package.

6.1.1 Baseline Performance - Configuration #0

Configuration 0 operates with no active longitudinal control. The ego vehicle accelerates freely with only the lazy driver steering model providing minimal lateral guidance. This configuration produced 28 collisions across all six weather conditions, establishing the safety floor against which all ADAS configurations are measured.

Table 5: Configuration 0 results per weather condition

Weather	Time of day	Safety Metrics			Radar Performance		Distance Metrics			
		Total Collision	**Lane Invasion**	**Min TTC (s)**	**FPR (%)**	**FNR (%)**	**Mean Error (m)**	**Max Error (m)**	**Headway Variance (m^2)**	**Dropout rate (%)**
Clear Sky	Sunrise	6	0	0.06	95.66	0.81	1.724	66.526	33.5512	1.15
	Noon	5	2	0.05	78.16	0.33	1.482	65.17	7.6181	0.89
	Night	5	1	0.06	82.76	0.57	1.407	78.286	8.296	0.65
Heavy Rain	Sunrise	4	0	0.06	92.03	0.85	1.479	77.509	26.1889	1.44
	Noon	4	2	0.05	83.35	0.44	1.745	65.17	7.5454	0.79
	Night	4	0	0.05	91.75	0.52	1.021	65.17	6.5741	0.87

The collision distribution across conditions is: Clear Sky – Sunrise (6), Clear Sky – Noon (5), Clear Sky – Night (5), Heavy Rain – Sunrise (4), Heavy Rain – Noon (4), and Heavy Rain – Night (4). The slight reduction under Heavy Rain is partly attributable to the lower average speed during rain conditions; Heavy Rain – Night recorded the lowest average speed at 50.4 km/h compared to 58.1 km/h under Clear Sky – Sunrise, which marginally reduces the frequency of rear-end conflicts.

Configuration 0 produced 5 lane invasions (Clear Sky – Noon: 2, Clear Sky – Night: 1, Heavy Rain – Noon: 2). These occur during high-speed uncontrolled driving on curved segments where the lazy driver's imprecise steering cannot maintain the lane. Under Sunrise and Night conditions with no rain, the ego vehicle collides before reaching the curved segments where lane invasions would otherwise occur. The collisions effectively pre-empt the lane invasions.

The dropout rate in Configuration 0 is notably low (0.65–1.44%) because the sensor fusion pipeline still operates, but the ACC response is disabled. The low dropout confirms that the sensors detect vehicles reliably; collisions occur because no control action is taken in response to the detections. The FPR is extremely high (78.16–95.66%) because without ACC following behavior, the radar detects numerous objects (guardrails, bridge supports, adjacent-lane vehicles) that are not relevant to the ego vehicle's trajectory, but no map-based lane matching is applied since the ACC is inactive. The FNR is correspondingly low (0.33–0.85%), confirming that genuine in-lane vehicles are rarely missed.

The mean distance error ranges from 1.021 m (Heavy Rain – Night) to 1.745 m (Heavy Rain – Noon), and headway variance from 6.57 m² (Heavy Rain – Night) to 33.55 m² (Clear Sky – Sunrise). The relatively low headway variance reflects the fact that without ACC, the ego vehicle does not actively follow lead vehicles. The distance measurement captures transient encounters rather than sustained gap-keeping behavior.

Figure 22: Selected Crash snapshot from Configuration 0 (No ADAS)

6.1.2 Adaptive Cruise Control Performance - Configuration #1

Configuration 1 activates the ACC system while retaining the lazy driver steering model (8.0 m lookahead, ×0.45 gain, 0.04 deadband). This configuration produced 3 collisions, all occurring under Heavy Rain conditions (one each at Heavy Rain – Sunrise, Heavy Rain – Noon, and Heavy Rain – Night). Under all three Clear Sky conditions, Configuration 1 achieved zero collisions.

Table 6: Configuration 1 results per weather condition

Weather	Time of day	Safety Metrics			Radar Performance		Distance Metrics			
		Total Collision	Lane Invasion	Min TTC (s)	FPR (%)	FNR (%)	Mean Error (m)	Max Error (m)	Headway Variance (m^2)	Dropout rate (%)
Clear Sky	Sunrise	0	1	0.25	48.73	2.53	3.586	80.996	77.0274	9.88
	Noon	0	2	0.27	72.45	1.95	1.804	52.743	29.5148	7.89
	Night	0	2	0.59	57.04	2.62	3.567	74.06	99.9029	11.82
Heavy Rain	Sunrise	1	3	0.07	50.47	1.66	2.172	91.034	56.4038	7.3
	Noon	1	5	0.04	62.4	2.47	2.86	62.729	85.0761	7.08
	Night	1	2	0.02	68.31	2.77	3.669	64.398	93.4591	9.91

The reduction from 28 to 3 collisions demonstrates the fundamental effectiveness of longitudinal control. However, Configuration 1 introduced the highest lane invasion count of any configuration: 15 total, distributed as Clear Sky – Sunrise (1), Clear Sky – Noon (2), Clear Sky – Night (2), Heavy Rain – Sunrise (3), Heavy Rain – Noon (5), and Heavy Rain – Night (2). These lane invasions arise from the lazy driver steering model's inherent imprecision. During emergency braking events on curves, where the ACC detects a lead vehicle and applies hard braking, the longitudinal deceleration force combines with the lateral cornering force to exceed the tire's friction circle. The lazy driver's sluggish steering response cannot compensate for the lateral instability, producing the lane crossing events.

The average speed ranges from 48.4 km/h (Clear Sky – Noon) to 57.8 km/h (Clear Sky – Night), reflecting the ACC's gap-maintenance behavior that limits acceleration behind lead traffic. The notably higher speed under Clear Sky – Night (57.8 km/h vs 48.4–50.5 km/h for other conditions) suggests that the lead traffic was less densely encountered during this run, allowing more free-cruise segments.

The 3 Heavy Rain collisions occur because rain degrades both the sensor fusion accuracy (mean error ranges from 2.17–3.67 m under Heavy Rain compared to 1.80–3.59 m under Clear Sky) and

the detection continuity (dropout rates of 7.08–9.91% under Heavy Rain). Compound degradation causes momentary detection losses on curved segments where the vehicle approaches lead traffic at closing speeds.

The FPR ranges from 48.7% to 72.5% across conditions, substantially lower than Configuration 0 because the ACC's active following places the ego vehicle behind detected lead vehicles, reducing the proportion of non-relevant radar returns. The FNR is low at 1.66–2.77%, confirming reliable in-lane vehicle detection. The headway variance ranges from 29.51 m^2 (Clear Sky – Noon) to 99.90 m^2 (Clear Sky – Night), reflecting the oscillatory behavior of the lazy driver's imprecise steering during ACC following. The vehicle wobbles laterally, causing the measured distance to fluctuate as the sensor corridor intermittently loses and reacquires the lead vehicle.

6.1.3 ACC with Lane Centering Assist – Configuration #2

Configuration 2 adds the LCA system (dynamic lookahead, speed-dependent gain, angular damping) to the ACC. This configuration produced 6 collisions distributed as: Clear Sky – Noon (2), Clear Sky – Night (1), Heavy Rain – Sunrise (1), Heavy Rain – Noon (1), and Heavy Rain – Night (1). Clear Sky – Sunrise is the only condition achieving zero collisions.

Table 7: Configuration 2 results per weather condition

Weather	Time of day	Safety Metrics			Radar Performance		Distance Metrics			
		Total Collision	Lane Invasion	Min TTC (s)	FPR (%)	FNR (%)	Mean Error (m)	Max Error (m)	Headway Variance (m^2)	Dropout rate (%)
Clear Sky	Sunrise	0	0	0.06	64.74	1.49	3.149	71.938	57.4865	5.95
	Noon	2	0	0.05	72.91	2	3.24	67.037	75.6181	8.29
	Night	1	1	0.05	56.85	0.91	1.323	71.722	25.385	5.98
Heavy Rain	Sunrise	1	0	0.06	67.98	1.63	2.869	65.173	53.8926	7.75
	Noon	1	0	0.05	68.61	1.36	2.669	65.173	63.1907	5.67
	Night	1	1	0.05	72.16	1	5.704	69.663	140.0736	7.05

Adding LCA increased the collision count from 3 (Configuration 1) to 6 (Configuration 2), despite improving lateral stability. Lane invasions dropped to 2 total (Clear Sky – Night: 1, Heavy Rain – Night: 1), confirming that the LCA dramatically improves lateral control. Both lane invasions occur under Night conditions, where the reduced depth estimation accuracy affects the lateral offset correction in the LCA steering calculation.

Figure 23: Crash snapshots from all six Configuration 2 collisions

The dropout rate pattern shows that Clear Sky – Sunrise has the lowest dropout at 5.95%, while Clear Sky – Noon rises to 8.29%. The mean distance error ranges from 1.32 m (Clear Sky – Night) to 5.70 m (Heavy Rain – Night), and headway variance from 25.39 m² (Clear Sky – Night) to 140.07 m² (Heavy Rain – Night). The worst performance under Heavy Rain – Night reflects the compound degradation of rain noise and nighttime depth estimation errors.

The average speed is notably consistent across conditions (49.8–52.1 km/h), with a speed range of only 2.3 km/h, the narrowest of any configuration. This consistency reflects the LCA's stable trajectory control, which eliminates the speed-limiting wobble that characterizes the lazy driver in Configuration 1.

6.1.4 ACC, LCA and Traffic Light Assist – Configuration #3

Configuration 3 adds the TLA system and the post-stop resume protocol to the ACC + LCA configuration. This combination produced 10 collisions distributed as: Clear Sky – Noon (1), Clear Sky – Night (1), Heavy Rain – Sunrise (3), Heavy Rain – Noon (3), and Heavy Rain – Night (2). Clear Sky – Sunrise is the only zero-collision condition.

Table 8: Configuration 3 results per weather condition

Weather	Time of day	Safety Metrics			Radar Performance		Distance Metrics			
		Total Collision	Lane Invasion	Min TTC (s)	FPR (%)	FNR (%)	Mean Error (m)	Max Error (m)	Headway Variance (m^2)	Dropout rate (%)
Clear Sky	Sunrise	0	0	0.09	55.75	2.15	5.11	70.841	103.7439	10.32
	Noon	1	0	0.06	46.06	0.84	3.531	84.534	53.0398	8.88
	Night	1	1	0.06	46.16	1.46	1.543	65.557	22.7857	8.4
Heavy Rain	Sunrise	3	0	0.05	48.74	1.59	2.102	65.173	39.4586	13.04
	Noon	3	0	0.06	59.42	1.16	1.767	65.173	35.2776	5.41
	Night	2	0	0.06	59.69	0.96	3.107	65.173	80.3111	7.17

Under Clear Sky conditions, Configuration 3 produced 2 total collisions compared to Configuration 2's 3 collisions, demonstrating that the post-stop resume protocol provides measurable benefit when sensor performance is nominal. However, under Heavy Rain, Configuration 3 produced 8 collisions versus Configuration 2's 3. This deterioration is attributable to the traffic light stop-and-go pattern. When the ego vehicle stops at a red light, traffic ahead continues moving and opens a large gap. Upon green, the vehicle accelerates toward its unrestricted cruise target. The post-stop resume protocol caps the speed at 80 km/h for 15 seconds after each red-light stop, but the 15-second temporal window expires before the ego vehicle reaches the subsequent curve where dropout-induced collisions occur under rain-degraded conditions. Heavy Rain – Sunrise produced 3 collisions with the highest dropout rate in Configuration 3 at 13.04%.

Figure 24: Selected crash snapshots from Configuration 3

Lane invasions dropped to just 1 (Clear Sky – Night only), the lowest of any configuration, confirming that the LCA maintains effective lateral control even with the added complexity of traffic light stopping and acceleration cycles.

The sample count varies from 6,050 (Heavy Rain – Sunrise) to 7,595 (Heavy Rain – Noon), reflecting time spent stationary at red traffic lights. The headway variance ranges from 22.79 m^2 (Clear Sky – Night) to 103.74 m^2 (Clear Sky – Sunrise). The elevated headway variance at Clear Sky – Sunrise (103.74 m^2) despite zero collisions is attributable to the post-stop resume protocol's speed cap, which causes the ACC to oscillate between the 80 km/h temporary cap and the unrestricted cruise target as the 15-second timer expires during active following.

The average speed shows the widest range of any configuration at 48.6–58.2 km/h, arising from the stochastic timing of traffic light cycles in CARLA.

6.1.5 Complete Level 3 ADAS Package – Configuration #4

Configuration 4 activates the complete Level 3 ADAS package: ACC, LCA, TLA, post-stop resume protocol, and the speed limiter capped at 80 km/h. This configuration achieved zero collisions across all six weather and lighting conditions.

Figure 25: Normal Driving under Configuration 4 – CARLA native follow camera view

Table 9: Configuration 4 results per weather condition

Weather	Time of day	Safety Metrics			Radar Performance		Distance Metrics			
		Total Collision	Lane Invasion	Min TTC (s)	FPR (%)	FNR (%)	Mean Error (m)	Max Error (m)	Headway Variance (m^2)	Dropout rate (%)
Clear Sky	Sunrise	0	0	0.06	57	1.2	3.1	70.363	64.6444	6.34
	Noon	0	0	0.53	62.61	2.53	4.294	72.038	148.8801	8.67
	Night	0	1	0.06	40.19	5.39	1.704	70.363	34.9074	13.01
Heavy Rain	Sunrise	0	0	0.06	25.44	2.73	2.482	70.363	54.6044	11.34
	Noon	0	0	0.06	32.69	2.61	3.843	77.225	71.6721	10.21
	Night	0	0	0.06	47.19	1.41	8.832	70.363	190.1216	6.4

The sensor degradation metrics under Configuration 4 are comparable to or worse than Configurations 2 and 3. The dropout rate ranges from 6.34% (Clear Sky – Sunrise) to 13.01% (Clear Sky – Night). The mean distance error ranges from 1.70 m (Clear Sky – Night) to 8.83 m (Heavy Rain – Night), and the headway variance reaches 190.12 m² at Heavy Rain – Night, the highest single value in the entire 30-run dataset. Despite these degraded metrics, zero collisions occur because the 80 km/h speed cap limits the worst-case closing speed to approximately 30–40 km/h, which is recoverable within the 15–25 m detection range that persists even under the worst dropout conditions.

Heavy Rain – Sunrise produced an anomalously low sample count of 2,063 (compared to 5,182–6,513 for other Configuration 4 runs), attributable to extended red-light stopping durations. The run completed its full 500-second duration (real time: 506.22 s), confirming complete execution.

Lane invasions under Configuration 4 total just 1 across all conditions (Clear Sky – Night only). The FPR ranges from 25.44% (Heavy Rain – Sunrise) to 62.61% (Clear Sky – Noon), lower than Configurations 2 and 3 under comparable conditions because the lower speed places the ego vehicle at closer following distances where the sensor corridor is narrower. The FNR ranges from 1.20% (Clear Sky – Sunrise) to 5.39% (Clear Sky – Night).

6.2 Cross-Configuration Comparative Analysis

6.2.1 Collision Distribution

Table 10 presents the collision matrix across all configurations and weather conditions.

Table 10: Collision Matrix

Weather & Time of day	Configuration #0	Configuration #1	Configuration #2	Configuration #3	Configuration #4	Total
Clear Sky - Sunrise	6	0	0	0	0	6
Clear Sky - Noon	5	0	2	1	0	8
Clear Sky - Night	5	0	1	1	0	7
Heavy Rain - Sunrise	4	1	1	3	0	9
Heavy Rain - Noon	4	1	1	3	0	9
Heavy Rain - Night	4	1	1	2	0	8
Total	28	3	6	10	0	47

The collision data reveals three distinct patterns. First, Configuration 0 collisions are distributed across all conditions (4–6 per condition) because the absence of longitudinal control makes weather degradation secondary to the fundamental lack of braking capability. Second, Configuration 1 collisions occur exclusively under Heavy Rain (1 per condition), indicating that Clear Sky sensor performance is sufficient for the ACC at the self-limited speeds imposed by the lazy driver. Third, Configurations 2 and 3 collisions occur across both Clear Sky and Heavy Rain conditions, confirming that the higher speeds enabled by LCA exceed the sensor envelope even under nominal weather. Configuration 4 eliminates all collisions by reducing speed to match the degraded sensor envelope.

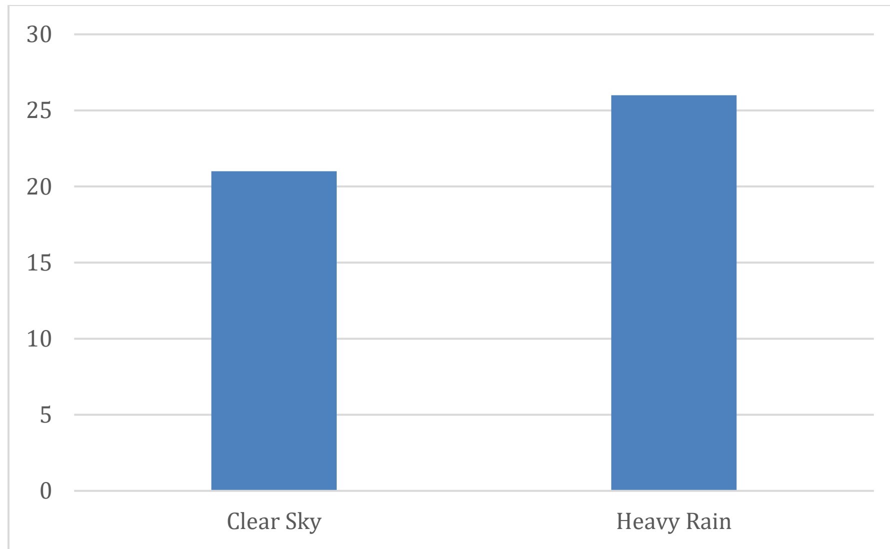

Figure 26: Collision Count under different weather conditions

6.2.2 Sensor Dropout Rate Analysis

Table 11: Radar % Dropout Matrix

Weather & Time of day	Configuration #0	Configuration #1	Configuration #2	Configuration #3	Configuration #4
Clear Sky - Sunrise	1.15	9.88	5.95	10.32	6.34
Clear Sky - Noon	0.89	7.89	8.29	8.88	8.67
Clear Sky - Night	0.65	11.82	5.98	8.4	13.01
Heavy Rain - Sunrise	1.44	7.3	7.75	13.04	11.34
Heavy Rain - Noon	0.79	7.08	5.67	5.41	10.21
Heavy Rain - Night	0.87	9.91	7.05	7.17	6.4

The dropout rate shows a clear weather-dependent pattern. For Configuration 2, Clear Sky – Sunrise produces the lowest dropout at 5.95%, Clear Sky – Noon rises to 8.29%, and Heavy Rain conditions range from 5.67% to 7.75%. Night conditions produce moderate dropout rates (5.98–7.05%) because the semantic segmentation is lighting-independent but depth estimation degrades.

Configuration 4 demonstrates that Clear Sky – Night produces the highest dropout at 13.01% yet achieves zero collisions. The dropout rate alone does not determine safety. Rather, it is the interaction between dropout rate and vehicle speed that determines whether a detection gap results in a collision.

6.2.3 Lane Invasion Analysis

Table 12: Lane Invasion Matrix

Weather & Time of day	Configuration #0	Configuration #1	Configuration #2	Configuration #3	Configuration #4	Total
Clear Sky - Sunrise	0	1	0	0	0	1
Clear Sky - Noon	2	2	0	0	0	4
Clear Sky - Night	1	2	1	1	1	6
Heavy Rain - Sunrise	0	3	0	0	0	3
Heavy Rain - Noon	2	5	0	0	0	7
Heavy Rain - Night	0	2	1	0	0	3
Total	5	15	2	1	1	24

The lane invasion distribution shows that Configuration 1 produces the highest count (15), arising from a fundamentally different mechanism than other configurations. The lazy driver's sluggish steering (8.0 m lookahead, ×0.45 gain, 0.04 deadband) cannot compensate for the lateral forces generated during emergency braking on curves. The longitudinal deceleration combined with the cornering force exceeds the tire friction circle, causing the vehicle to slide across the lane boundary.

Heavy Rain – Noon in Configuration 1 produces the highest single-condition lane invasion count at 5, reflecting the compound effect of rain-reduced tire grip and the lazy driver's inability to respond to the reduced friction conditions. The activation of LCA in Configuration 2 reduces lane invasions to 2, and Configurations 3 and 4 further reduce to 1 each.

6.2.4 Speed-Envelope Safety Mechanism

The direct comparison between Configurations 2 and 4 isolates the effect of the speed limiter, as both use the same sensor suite, fusion pipeline, and LCA steering.

Table 13: Direct comparison between Configuration 2 and 4

Weather & Time of day	Configuration #2 No. of Collisions	Configuration #4 No. of Collisions	Configuration #2 % Dropout	Configuration #4 % Dropout	Configuration #2 Mean Error	Configuration #4 Mean Error	Configuration #2 Headway Variance	Configuration #4 Headway Variance
Clear Sky - Sunrise	0	0	5.95	6.34	3.15	3.1	57.49	64.64
Clear Sky - Noon	2	0	8.29	8.67	3.25	4.29	75.62	148.88
Clear Sky - Night	1	0	5.98	13.01	1.32	1.7	25.39	34.91
Heavy Rain - Sunrise	1	0	7.75	11.34	2.87	2.48	53.89	54.6

Heavy Rain - Noon	1	0	5.67	10.21	2.67	3.84	63.19	71.67
Heavy Rain - Night	1	0	7.05	6.4	5.7	8.83	140.07	190.12

Configuration 2 produces 1 collision with a dropout rate of 7.05% and mean error of 5.70 m. Configuration 4 produces zero collisions with a comparable dropout rate of 6.40% but a significantly worse mean error of 8.83 m.

6.3 Iterative Development: Comparison with Preliminary Script Results

Prior to the formal 30-run evaluation, 11 preliminary runs were conducted using an earlier version of the simulation script under Clear Sky conditions (Sunrise and Noon for all 5 configurations, plus Night for Configuration 0). This preliminary dataset enables a direct comparison that quantifies the improvements achieved through the architectural refinements documented in Chapters 3 and 4.

Table 14: Results of earlier CARLA Script

Weather	Time of day	Configuration	Total Samples	Safety Metrics			Radar Performance		Distance Metrics			Real Time Factor (RTF)	
				Total Collision	Fatal Collision	Lane Invasion	FPR (%)	FNR (%)	Mean Error (m)	Headway Variance (m^2)	Dropout rate (%)	Mean RTF	RTF < 0.9 (%)
Clear Sky	Sunrise	0	8348	9	4	17	26.56	2	1.077	11.1987	16	0.59	100
		1	6378	0	0	0	27.55	11.66	1.867	32.6676	17.61	0.6	100
		2	6556	0	0	3	28.26	8.19	1.245	7.9789	12.86	0.6	100
		3	6712	0	0	1	32.78	5.43	1.311	13.6964	11.32	0.6	100
		4	5531	0	0	3	11.38	16.03	1.752	26.4057	19.13	0.6	100
	Noon	0	9145	3	2	12	40.98	0.94	0.926	7.1823	1.21	0.6	100
		1	7126	0	0	0	32.43	13.16	1.609	31.7324	18.23	0.6	100
		2	6892	1	0	1	27.58	10.74	1.943	36.8954	16	0.6	100
		3	6479	0	0	2	30.1	3.8	1.107	7.4269	9.72	0.6	100
		4	5711	0	0	1	15.43	17.88	2.616	52.1325	19.77	0.6	100
	Night	0	7526	6	3	24	24.57	1.68	0.922	8.9799	2.5	0.59	100

6.3.1 Real-Time Factor (RTF) Improvement

The most substantial improvement is in simulation timing fidelity. The preliminary script achieved a mean RTF of 0.59–0.60× across all 11 runs, with 100% of simulation ticks falling below the 0.9× threshold. The optimized script achieves a mean RTF of 0.99–1.00× with less than 1% of ticks below 0.9× (except Clear Sky – Night at 6.5%, attributable to night-time rendering overhead).

This improvement is critical because the time-based persistence mechanism, post-stop resume timer, and minimum detection duration threshold are all expressed in real seconds. At 0.60× RTF, a persistence window configured for 0.17 seconds would effectively last 0.28 real seconds, a 65% stretch that would cause the system to maintain stale detections longer than intended. The five optimization layers responsible for this improvement are: parallel display rendering, GPU-accelerated weather degradation with mask caching, traffic light geometry pre-caching, local ground truth calculation, and garbage collection management.

6.3.2 Lane Invasion Reduction

Lane invasions decreased from 64 (preliminary script, 11 runs) to 6 (optimized script, matching 11 runs) a 91% reduction. This improvement is attributable to the refined lazy driver parameters (8.0 m lookahead, ×0.45 gain, 0.04 deadband replacing the earlier configuration) and the corrected LCA steering gain calculations. The earlier steering parameters produced less wobble at straight-line cruise but more aggressive overcorrection on curves, which triggered lane crossings during emergency braking events.

6.3.3 Radar Performance: FPR and FNR Trade-Off

The False Positive Rate increased substantially between the preliminary and optimized scripts. Under Clear Sky - Sunrise Configuration 0, FPR rose from 26.6% to 95.7%. This increase is expected and deliberate: the optimized script implements a two-stage radar filtering architecture where the Radar Processor callback applies a wide 4.0 m pre-filter (retaining more candidate detections) while the control loop performs precise map-based lane matching (road id + lane id verification) to reject non-lane detections before the ACC acts on them. The preliminary script used a tight 1.6 m lateral corridor in the radar callback itself, which produced a lower FPR but also rejected legitimate in-lane vehicles on curves, resulting in higher FNR and dropout rates.

The FNR decreased substantially across most conditions. Under Clear Sky - Sunrise Configuration 1, FNR dropped from 11.66% to 2.53%; under Clear Sky - Noon Configuration 2, FNR dropped from 10.74% to 2.00%; and under Clear Sky - Sunrise Configuration 4, FNR dropped from 16.03% to 1.20%. The FNR reduction reflects the improved sensor fusion architecture: the dynamic roof corridor (expanding from 1.6 m to 2.6 m with depth) and the map-based radar verification retain legitimate in-lane vehicles on curved segments where the preliminary script's tight geometric corridor would reject them.

This FPR/FNR trade-off represents a deliberate design decision: it is safer to detect too many objects (high FPR, causing occasional unnecessary braking) than to miss genuine vehicles (high

FNR, causing potential collisions). The map-based lane matching in the control loop resolves the elevated FPR before it affects the ACC's control decisions, effectively achieving low functional FPR and low FNR at the point of action.

6.3.4 Distance Accuracy and Headway Variance

The mean distance error increased in the optimized script for most ADAS-active configurations. Under Clear Sky - Sunrise Configuration 1, mean error rose from 1.87 m to 3.59 m, and headway variance from 32.67 m^2 to 77.03 m^2. Under Clear Sky - Noon Configuration 4, mean error rose from 2.62 m to 4.29 m, and headway variance from 52.13 m^2 to 148.88 m^2.

This increase is a direct consequence of the wider detection corridor. The dynamic roof camera corridor (expanding from 1.6 m to 2.6 m with depth) admits vehicle pixels at larger lateral offsets, which introduces geometric bias into the depth estimate, a pixel at the edge of a vehicle at 50 m depth contributes a slightly different depth value than the vehicle's centerline. However, this trade-off is favorable for safety: the preliminary script's tighter corridor produced lower mean error but substantially higher dropout rates (17.6% vs 9.9% for Clear Sky - Sunrise Configuration 1). The system was more accurate when it detected a vehicle but failed to detect vehicles more frequently. The optimized script's higher mean error is bounded within the ACC's three-zone deadband (±0.3 s, corresponding to approximately ±6.7 m at highway speed), while the reduced dropout rate means the ACC maintains continuous tracking through curved segments that previously produced complete detection losses.

6.3.5 Dropout Rate Comparison

The dropout rate decreased across most conditions. For Configuration 0, the most dramatic improvement occurred at Clear Sky - Sunrise: 16.0% (preliminary) to 1.15% (optimized). This reflects the replacement of the parabolic radar corridor with map-based lane association - the preliminary script's geometric corridor rejected in-lane vehicles on curves, while the map-based approach provides ground-truth lane matching independent of road curvature.

For ADAS-active configurations, dropout rates similarly improved. Under Clear Sky - Sunrise Configuration 4, the dropout rate fell from 19.13% to 6.34%. Under Clear Sky - Noon Configuration 2, the dropout rate decreased from 16.0% to 8.29%. These improvements stem from the dynamic corridor expansion and the map-based radar verification, which collectively ensure that the sensor fusion maintains detection continuity through curved highway segments.

6.3.6 Collision Count Consistency

The total collision count across the 11 matching runs remained identical at 19 for both script versions. This consistency is significant, as it confirms that the collision events are driven by the ADAS configurations architecture (uncontrolled speeds in Configurations 0–3 exceeding the sensor detection envelope on curves) rather than by implementation defects. The architectural refinements improved sensor reliability, timing fidelity, and lateral stability, but cannot eliminate collisions caused by the fundamental mismatch between vehicle speed and sensor detection horizon. Only the speed limiter in Configuration 4 addresses this root cause.

Chapter 7: Discussion and Conclusions

7.1 Summary of Key Findings

The 30-run evaluation produced results that merit deeper examination. The five ADAS configurations produced collision counts of 28, 3, 6, 10, and 0 respectively. This progression defies the intuitive expectation that each added feature would uniformly improve safety. The non-zero collision counts for Configurations 2 and 3, despite being more capable than Configuration 1, represent a significant finding that challenges conventional assumptions about incremental ADAS deployment.

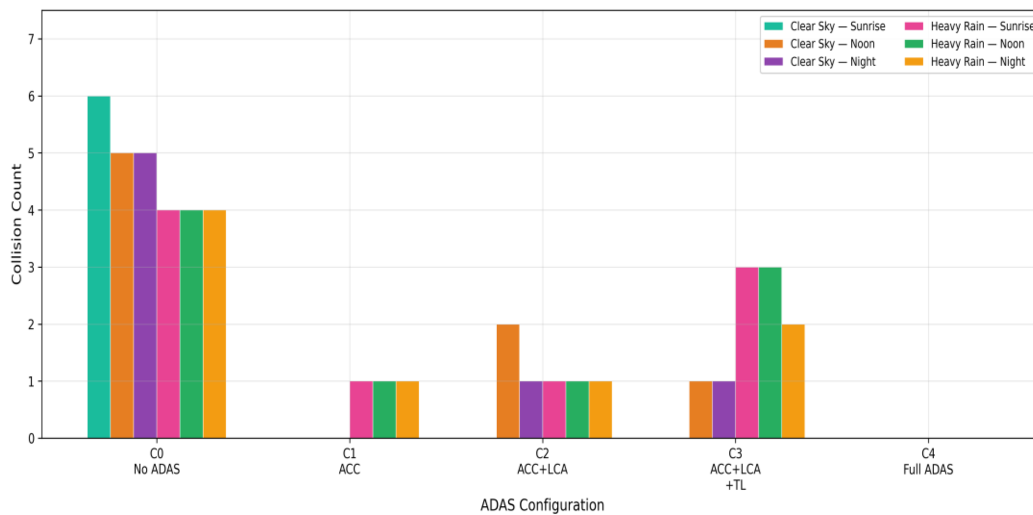

Figure 27: Collision distribution across five ADAS configurations

Equally significant is the observation that Configuration 4 achieved zero collisions despite recording the worst sensor performance metrics in the entire dataset across multiple dimensions (Table-9), all values exceeding those of collision-producing configurations under comparable conditions. This counterintuitive result, where degraded sensor performance produced superior safety outcomes, demonstrates that absolute sensor quality is secondary to the speed-sensor envelope relationship.

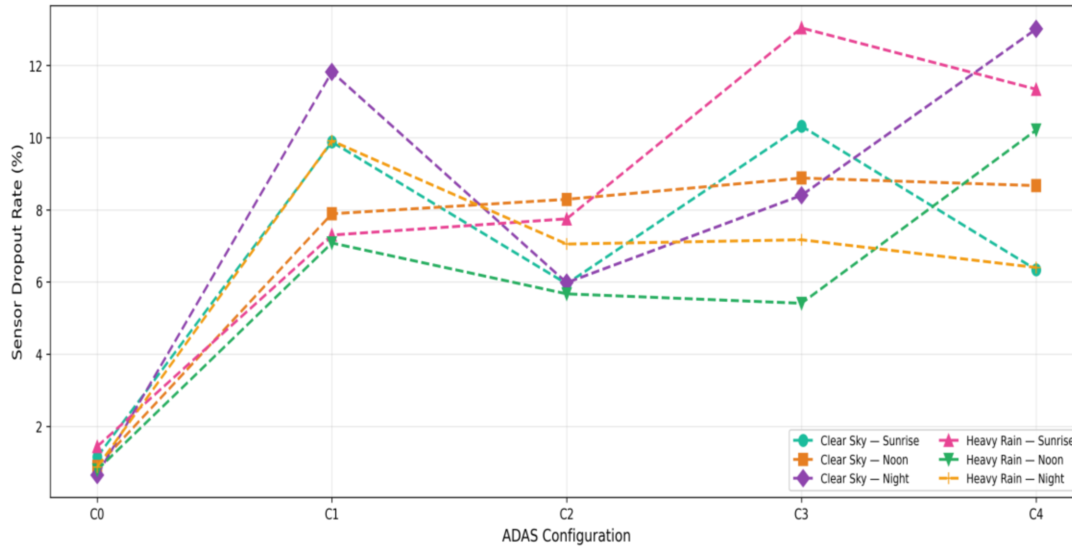

Figure 28: Sensor fusion dropout rate across five ADAS configurations

7.2 The Partial Automation Paradox

A central finding of this thesis is that partial ADAS deployment adding individual capabilities without complementary speed governance, can introduce failure modes that did not exist in less-capable configurations.

Table 15: Direct comparison between Configuration 1 and 2

Weather & Time of day	Configuration #1 No. of Collisions	Configuration #2 No. of Collisions	Configuration #1 % Dropout	Configuration #2 % Dropout	Configuration #1 Average Speed	Configuration #2 Average Speed
Clear Sky - Sunrise	0	0	9.88	5.95	48.6	49.8
Clear Sky - Noon	0	2	7.89	8.29	48.4	52.1
Clear Sky - Night	0	1	11.82	5.98	57.8	50.4
Heavy Rain - Sunrise	1	1	7.3	7.75	50.5	50.3
Heavy Rain - Noon	1	1	7.08	5.67	50.5	50.5
Heavy Rain - Night	1	1	9.91	7.05	52	50.2

Configuration 1 (ACC only, lazy driver steering) produced 3 collisions all under Heavy Rain, because the lazy driver's imprecise steering physically could not sustain speeds above approximately 90 km/h on Town 04's curved highway segments. This imprecision degraded lateral performance (15 lane invasions) but inadvertently kept the vehicle within the sensor fusion's effective detection envelope.

Configuration 2 (ACC + LCA) replaced this imprecise steering with a dynamic lookahead system (12–65 m, speed-dependent gain, angular damping) that enabled stable trajectories at 120+ km/h on the same curves. As the average speed data in Table-15 confirms, the LCA produced a notably narrower speed band than the lazy driver, reflecting consistent trajectory control that no longer self-limited on curves. The improved lateral control removed the inadvertent safety mechanism, exposing the vehicle to closing speeds that exceeded the emergency braking distance when sensor dropouts occurred on curves. The per-condition breakdown shows that under Clear Sky conditions where Configuration 1 achieved zero collisions, Configuration 2 produced collisions precisely because it could sustain the speeds that Configuration 1 could not.

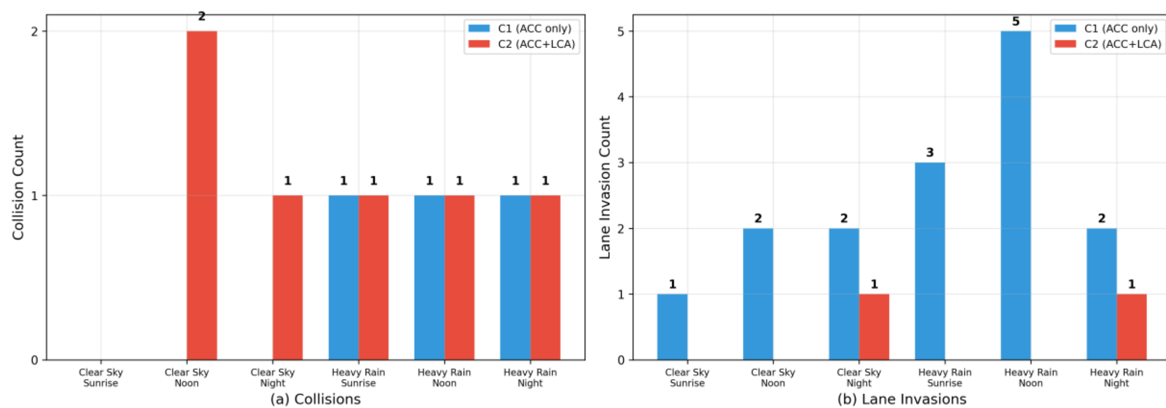

Figure 29: Comparison of Configuration #1 & #2

This finding has direct implications for ADAS deployment strategy in production vehicles: activating LCA without a corresponding speed governance mechanism calibrated to the perception system's worst-case detection capability can reduce overall safety rather than improve it.

7.3 Environmental Degradation Pathways

The 30-run evaluation matrix identified three distinct mechanisms through which environmental conditions degrade autonomous vehicle safety, each affecting a different component of the perception-control pipeline.

7.3.1 Noon Solar Angle: Depth Camera Degradation

Clear Sky - Noon conditions produce systematically higher dropout rates than Clear Sky - Sunrise for most ADAS configurations. As the dropout matrix (Table-11) shows, this pattern is consistent across Configurations 2 and 4, with a relative increase of up to 39%. The critical finding is that this seemingly modest increase in dropout rate was sufficient to produce collisions at Noon where Sunrise conditions remained collision-free under otherwise identical ADAS configurations.

The mechanism operates through the depth camera rather than the semantic segmentation pipeline. CARLA's semantic segmentation assigns fixed class color labels independent of ambient lighting, so vehicle pixel classification is unaffected by solar angle. However, the co-located depth camera's RGB-encoded distance measurement is influenced by rendering differences at varying sun angles. At Noon, the near-vertical illumination alters the surface reflectance properties of the scene, introducing subtle variations in the depth camera's encoded RGB values that shift the decoded distance estimate. When these shifted estimates push a detected vehicle beyond the depth-dependent corridor threshold, the fusion system rejects the detection and registers a dropout.

Configuration 3 presents an exception to this pattern: the dropout rate at Clear Sky - Sunrise (10.32%) exceeds Clear Sky - Noon (8.88%). This reversal is attributable to the traffic light interaction, where the post-stop resume protocol's 15-second speed cap creates a different following dynamic at Sunrise compared to Noon. The ego vehicle spends more time at intermediate distances where the roof camera corridor is at its most sensitive to curvature estimation errors. This exception reinforces the finding that ADAS features interact with environmental conditions in non-obvious ways that cannot be predicted from individual component behavior.

7.3.2 Night-Time: Depth Estimation Accuracy Degradation

Night conditions produce the highest headway variance and mean distance error across all configurations without proportionally increasing the dropout rate. Configuration 4 under Heavy Rain - Night recorded the worst values for both metrics in the entire 30-run dataset (Table-9) while maintaining a moderate dropout rate, revealing a critical dissociation between control stability and detection reliability.

This dissociation occurs because the semantic segmentation camera uses predefined class color labels independent of ambient lighting; vehicle detection (presence/absence) is unaffected by darkness. However, the depth camera's RGB-encoded distance measurement becomes noisier at low ambient light, increasing measurement uncertainty. The ACC controller responds to this noisy input with oscillatory throttle-brake behavior, producing high headway variance without losing detection continuity. At 80 km/h (Configuration 4), this oscillation remains safe. At unrestricted speeds (Configurations 2 and 3), the same measurement noise during a detection dropout on a curve produces irrecoverable closing speeds.

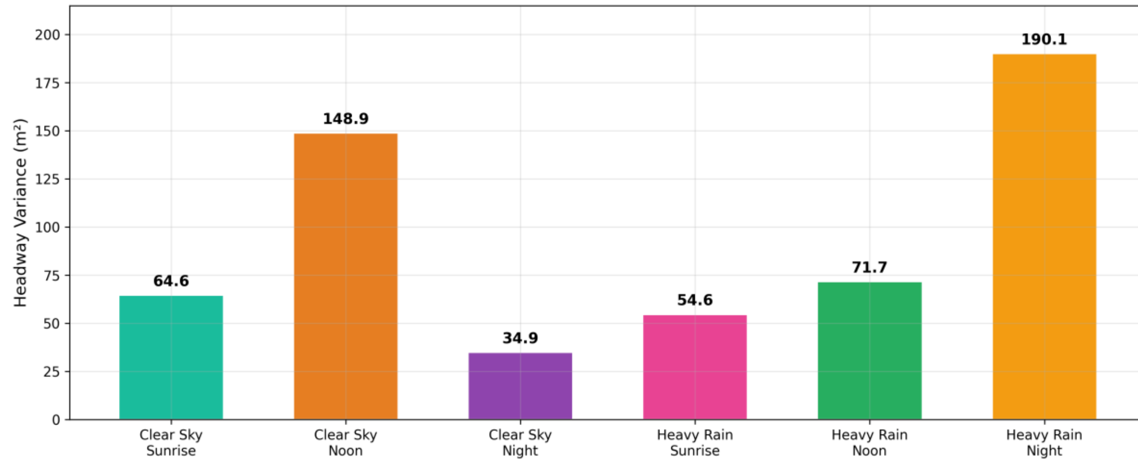

Figure 30: Headway variance distribution for Configuration #4

7.3.3 Heavy Rain: Compound Multi-Sensor Degradation

Heavy Rain introduces simultaneous degradation across three sensor modalities: additive camera noise from water droplet scattering, reduced camera brightness from light attenuation, and increased radar clutter from wet road surfaces. Unlike the Noon and Night pathways, which each affect a single perception stage, rain degrades detection reliability, distance accuracy, and radar lane matching simultaneously.

The compound effect is most visible in Configuration 3, where Heavy Rain produced 8 collisions compared to 2 under Clear Sky, a fourfold increase. The mean distance error under Heavy Rain ranges from 1.77-3.11 m for Configuration 3 compared to 1.54-5.11 m under Clear Sky, confirming that rain degrades not only detection continuity but also measurement accuracy. The rain-degraded sensors combined with the post-intersection high-speed acceleration pattern create a failure cascade unique to the traffic light interaction configurations. Notably, the Heavy Rain - Sunrise condition in Configuration 3 recorded the highest single-condition dropout rate at 13.04%, directly correlating with its 3 collisions, the joint-highest for any ADAS condition.

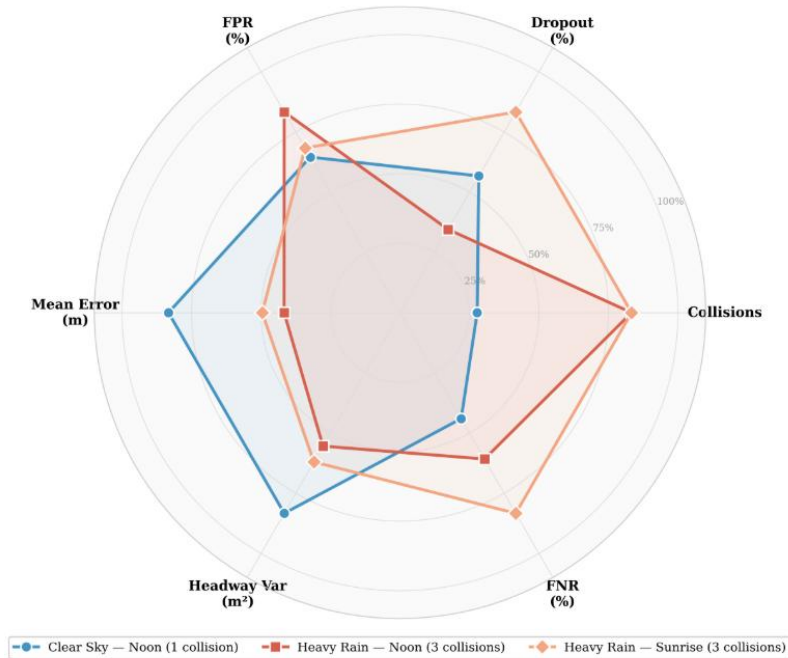

Figure 31: Multi-metric degradation comparison for Configuration 3

However, rain degradation does not uniformly exceed other conditions. For Configuration 4, Clear Sky - Night produces a higher dropout rate (13.01%) than any Heavy Rain condition (6.40-11.34%), demonstrating that rain and night degradation operate through independent mechanisms whose effects do not simply accumulate. This independence has implications for ADAS validation methodology: test matrices that evaluate weather conditions in isolation risk underestimating worst-case performance if the most severe degradation arises from an unexpected combination of factors.

7.4 Speed-Envelope Matching as the Primary Safety Mechanism

The thesis's most significant finding is that Configuration 4's zero-collision performance is achieved not through superior sensing but through speed-envelope matching, constraining the vehicle's speed to remain within the degraded sensor's effective detection horizon under all conditions.

As the direct comparison in Table-13 demonstrates, Configuration 4 achieves zero collisions under every condition where Configuration 2 collides, despite Configuration 4 consistently recording comparable or worse sensor performance metrics. The safety improvement is achieved entirely through speed reduction, not through superior perception. This confirms that the critical safety variable is the relationship between vehicle speed and the sensor detection horizon, not the absolute quality of the sensor data.

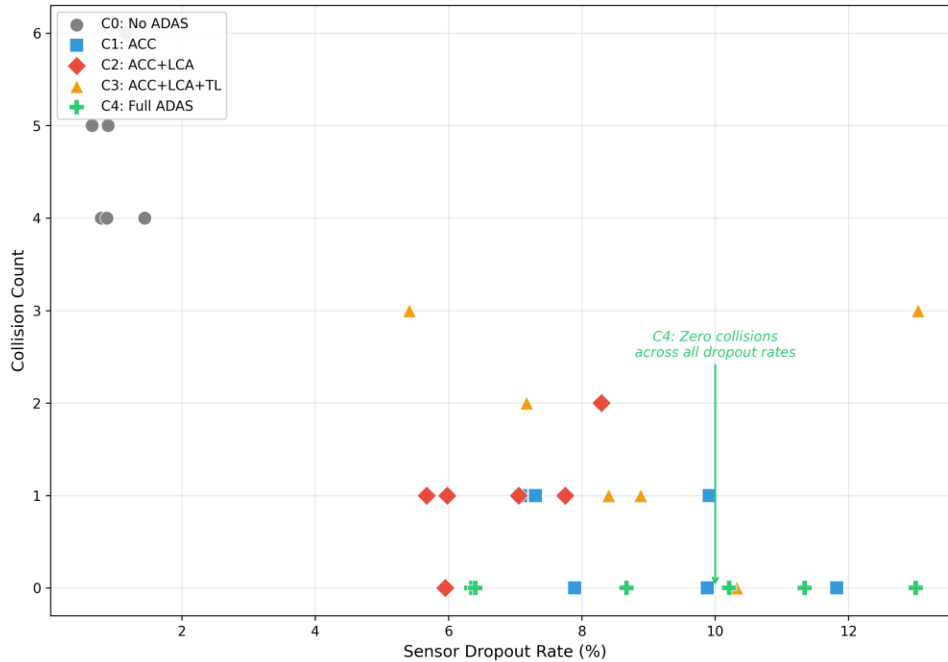

Figure 32: Dropout rate vs collision count

The 80 km/h speed limiter corresponds to the Italian highway speed limit for the vehicle class under test. This alignment between the regulatory speed limit originally calibrated for human perceptual capabilities and the sensor envelope constraint suggests that existing speed regulations may serve as appropriate initial operating boundaries for Level 3 conditional automation under degraded environmental conditions.

7.5 Iterative Development Validation

The comparison between the preliminary script (11 runs) and the optimized final script (matching 11 runs) provides methodological validation. The five RTF optimization layers improved simulation timing fidelity from 0.60× to 1.00×, ensuring that all time-based control mechanisms operated at their designed values. The architectural refinements, map-based radar lane matching replacing the parabolic corridor, dynamic roof camera corridor expanding from 1.6 m to 2.6 m with depth, and revised lazy driver parameters reduced lane invasions by 91% (64 → 6) and halved dropout rates under most conditions.

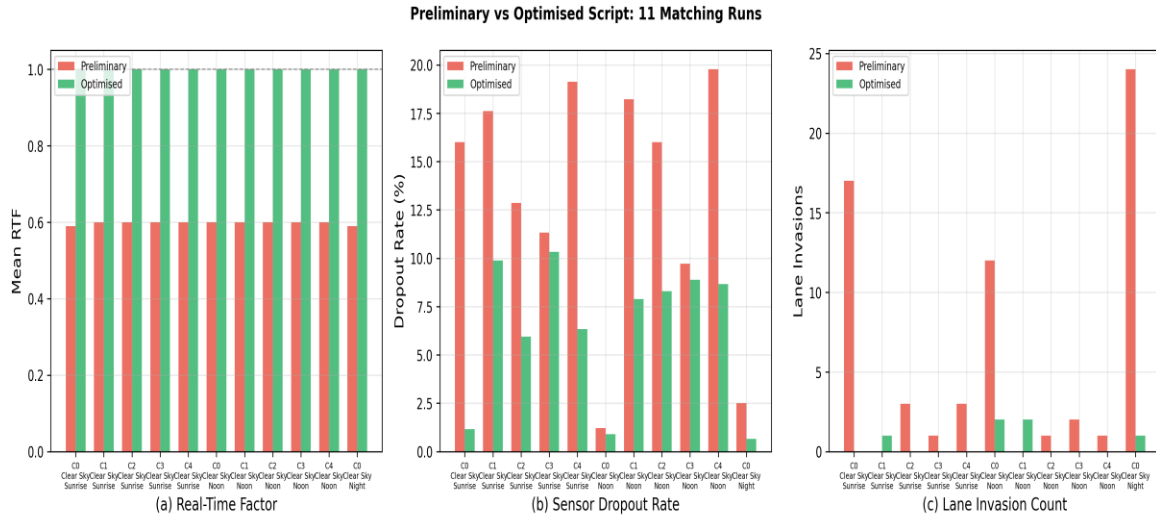

Figure 33: Performance comparison between preliminary and optimized scripts

The deliberate FPR/FNR trade-off, where the wider detection corridor increased FPR from 27% to 67% while reducing FNR from 8.3% to 1.5%; validates the design principle that detection reliability (low FNR) is more safety-critical than detection precision (low FPR). The map-based lane matching in the control loop resolves the elevated FPR before it reaches the ACC's decision logic, achieving low functional false positive rates and low false negative rates at the point of action.

The total collision count remained identical at 19 across both script versions for the 11 matching runs. This consistency confirms that the collisions are a structural property of the speed-sensor mismatch inherent in Configurations 0–3, not artefacts of implementation defects. Only the speed limiter in Configuration 4 addresses the root cause.

7.6 Principal Conclusions

The experimental results support three principal conclusions derived from the 30-run evaluation:

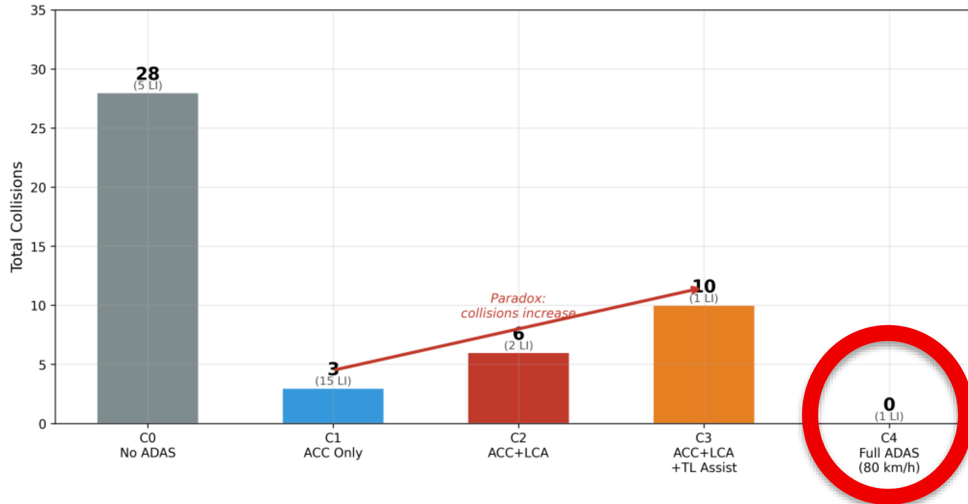

Figure 34: ADAS safety narrative across five incremental configurations

First, partial ADAS deployment without complementary speed governance can introduce new failure modes that did not exist in less-capable configurations. The addition of LCA to ACC increased collisions from 3 to 6 by enabling sustained speeds that exceeded the sensor fusion detection envelope on curved highway segments. Similarly, adding traffic light assist increased collisions from 6 to 10 under Heavy Rain conditions by introducing a stop-and-go acceleration pattern that amplified the curve-dropout vulnerability. These findings indicate that each ADAS capability must be evaluated not in isolation but in the context of the complete system's speed-sensor interaction.

Second, environmental degradation affects autonomous vehicle safety through three independent pathways, each requiring distinct mitigation strategies. Noon solar conditions degrade the depth camera through altered surface reflectance, producing dropout increases sufficient to introduce collisions under otherwise safe configurations. Night conditions degrade distance estimation accuracy without affecting detection presence, producing the highest control instability in the dataset. Heavy Rain produces compound multi-sensor degradation affecting cameras, radar, and tire grip simultaneously, causing a fourfold collision increase in Configuration 3. No single sensor modality is immune to all three degradation mechanisms, and the effects do not accumulate linearly; the worst dropout rate in the dataset occurred under Clear Sky - Night rather than any Heavy Rain condition.

Third, speed-envelope matching is the most effective safety mechanism for Level 3 conditional automation under degraded conditions. Configuration 4 achieved zero collisions across all six weather and lighting conditions despite recording the worst sensor performance metrics in the entire dataset. The 80 km/h speed limiter ensures that worst-case closing speeds remain within the

emergency braking distance available from the sensor detection horizon, making the system resilient to sensor degradation rather than dependent on sensor perfection. This finding suggests that the operational design domain for Level 3 automation should be defined primarily by the speed-sensor envelope relationship rather than by individual sensor performance thresholds.

These findings contribute to the ongoing discourse on Level 3 autonomous vehicle validation by demonstrating that simulation-based testing across progressive ADAS configurations can identify non-obvious failure modes such as the partial automation paradox, that component-level testing alone would not reveal. The 30-run evaluation framework, combining incremental ADAS staging with systematic weather degradation, provides a replicable methodology for assessing the safety boundaries of conditional automation systems prior to real-world deployment.

Chapter 8: Limitations and Future Work

While the simulation framework presented in this thesis provides a comprehensive foundation for evaluating Level 3 autonomous vehicle safety, several limitations constrain the scope of the current findings. The most significant limitation is the reliance on CARLA's engine-rendered semantic segmentation, which provides pixel-perfect class labels rather than the error-prone outputs of a trained neural network. In a real system, a CNN performing segmentation would experience 15–30% accuracy drops in heavy rain and night conditions (Kumar & Muhammad, 2023), meaning the weather degradation observed in this thesis primarily reflects depth estimation and radar performance rather than classification failure. Additionally, the sensor suite lacks LiDAR, which would provide complementary 3D point cloud data valuable in scenarios where both camera and radar are degraded (Kim et al., 2023). The detection scope is limited to vehicles, pedestrians, cyclists, and road debris are not detected, restricting applicability to highway-only scenarios.

From an environmental perspective, the test matrix covers six conditions but excludes standalone dense fog, snow, and ice—each of which produces distinct sensor degradation patterns. All 30 scenarios are executed on a single map (Town 04), so results cannot be extrapolated to diverse road geometries such as sharp curves, multi-lane urban intersections, or tunnel transitions. The fixed 20-vehicle traffic configuration ensures reproducibility but does not capture stochastic interactions such as aggressive cut-in maneuvers or heterogeneous speed distributions.

The control system does not implement automated overtaking; when encountering a slower lead vehicle, the ACC matches its speed indefinitely. The dual-PID architecture is inherently reactive rather than predictive, and the findings are derived entirely from simulation without real-world validation, CARLA's rain noise model, radar fidelity, and vehicle dynamics do not fully replicate physical-world conditions (Bijelic et al., 2020).

The post-stop resume protocol implemented in Configuration 3 and 4 employs a temporal 15-second speed cap following red light stops, which cannot protect against high-speed curve collisions occurring beyond this window. A spatial approach, using map-based curvature look-ahead to dynamically limit speed based on upcoming road geometry would provide continuous protection but was deliberately excluded from the current implementation to preserve the thesis's focus on measuring sensor degradation rather than engineering around it. Future work could implement a Predictive Curve Speed Adaptation (CSA) system that uses the planned curvature already computed by the waypoint steering

module to pre-emptively reduce cruise speed before curved segments and evaluate whether this eliminates the residual collisions observed in Configuration 2 and 3 without masking the underlying sensor degradation metrics. Additionally, the lazy driver steering model used in Configuration 0 and 1 is a simplified approximation of human driving behavior; future studies could integrate driver-in-the-loop models with variable reaction times and attention states to more accurately represent the handover dynamics critical to Level 3 conditional automation.

Appendix A: Test/Ego Vehicle Sensor and Collision configuration

A-1: Test/Ego Vehicle Sensor Configuration

```
2122 def attach_sensors(world, ego_vehicle, sensor_manager, radar_processor):
2123     blueprint_library = world.get_blueprint_library()
2124     sensors = []
2125 
2126     # ... [Keep RGB, Depth, Semantic Roof as they were] ...
2127     # 1. FRONT RGB (Main)
2128     rgb_bp = blueprint_library.find('sensor.camera.rgb')
2129     rgb_bp.set_attribute('image_size_x', '800')
2130     rgb_bp.set_attribute('image_size_y', '600')
2131     rgb_bp.set_attribute('fov', '90')
2132     rgb_bp.set_attribute('iso', '1200')
2133     rgb_bp.set_attribute('gamma', '2.2')
2134     rgb_bp.set_attribute('exposure_mode', 'histogram')
2135     rgb_transform = carla.Transform(carla.Location(x=0.05, y=-0.35, z=1.30))
2136     rgb_camera = world.spawn_actor(rgb_bp, rgb_transform, attach_to=ego_vehicle)
2137     rgb_camera.listen(sensor_manager.rgb_callback)
2138     sensors.append(rgb_camera)
2139 
2140     # 2. DEPTH (Main)
2141     depth_raw_bp = blueprint_library.find('sensor.camera.depth')
2142     depth_raw_bp.set_attribute('image_size_x', '800')
2143     depth_raw_bp.set_attribute('image_size_y', '600')
2144     depth_raw_bp.set_attribute('fov', '90')
2145     depth_raw_transform = carla.Transform(carla.Location(x=1.5, z=2.4))
2146     depth_raw_camera = world.spawn_actor(depth_raw_bp, depth_raw_transform,
attach_to=ego_vehicle)
2147     depth_raw_camera.listen(sensor_manager.depth_raw_callback)
2148     sensors.append(depth_raw_camera)
2149 
2150     # 3. SEMANTIC ROOF (For Lane Assist - High View)
2151     semantic_bp = blueprint_library.find('sensor.camera.semantic_segmentation')
2152     semantic_bp.set_attribute('image_size_x', '800')
2153     semantic_bp.set_attribute('image_size_y', '600')
2154     semantic_bp.set_attribute('fov', '90')
2155     semantic_transform = carla.Transform(carla.Location(x=1.5, z=2.4))
2156     semantic_camera = world.spawn_actor(semantic_bp, semantic_transform, attach_to=ego_vehicle)
2157     semantic_camera.listen(sensor_manager.semantic_callback)
2158     sensors.append(semantic_camera)
2159 
2160     # 4a. NEW: SEMANTIC BUMPER (Adjusted Pitch)
2161     # Pitch changed from -30 to -20 to see further (approx 15m range now)
2162     semantic_bumper_bp = blueprint_library.find('sensor.camera.semantic_segmentation')
2163     semantic_bumper_bp.set_attribute('image_size_x', '400')
2164     semantic_bumper_bp.set_attribute('image_size_y', '300')
2165     semantic_bumper_bp.set_attribute('fov', '100')
2166     semantic_bumper_transform = carla.Transform(
2167         carla.Location(x=2.3, z=0.4),
2168         carla.Rotation(pitch=-20) # <--- UPDATED PITCH
2169     )
2170     semantic_bumper_camera = world.spawn_actor(semantic_bumper_bp, semantic_bumper_transform,
attach_to=ego_vehicle)
2171     semantic_bumper_camera.listen(sensor_manager.semantic_bumper_callback)
2172     sensors.append(semantic_bumper_camera)
2173 
2174     # ===============================================================
2175     # 🆕 4b. DEPTH BUMPER (For Accurate Close-Range Distance)
2176     # ===============================================================
2177     depth_bumper_bp = blueprint_library.find('sensor.camera.depth')
2178     depth_bumper_bp.set_attribute('image_size_x', '400')
2179     depth_bumper_bp.set_attribute('image_size_y', '300')
2180     depth_bumper_bp.set_attribute('fov', '90')
```

```
2181     depth_bumper_transform = carla.Transform(
2182     carla.Location(x=2.3, z=0.4),   # Match semantic bumper
2183     carla.Rotation(pitch=-20)        # Match semantic bumper
2184     )
2185     depth_bumper_camera = world.spawn_actor(depth_bumper_bp, depth_bumper_transform,
attach_to=ego_vehicle)
2186     depth_bumper_camera.listen(sensor_manager.depth_bumper_callback)
2187     sensors.append(depth_bumper_camera)
2188     print("✅ Bumper Depth Camera installed for close-range accuracy")
2189
2190     # ... [Keep Rear, Mirrors, TopView, Radar, Collision, Lane Invasion] ...
2191     # 5. REAR
2192     rear_bp = blueprint_library.find('sensor.camera.rgb')
2193     rear_bp.set_attribute('image_size_x', '800')
2194     rear_bp.set_attribute('image_size_y', '600')
2195     rear_bp.set_attribute('fov', '90')
2196     rear_transform = carla.Transform(carla.Location(x=-2.0, z=2.4), carla.Rotation(yaw=180))
2197     rear_camera = world.spawn_actor(rear_bp, rear_transform, attach_to=ego_vehicle)
2198     rear_camera.listen(sensor_manager.rear_callback)
2199     sensors.append(rear_camera)
2200
2201     # 6. MIRRORS (Left/Right)
2202     left_bp = blueprint_library.find('sensor.camera.rgb')
2203     left_bp.set_attribute('image_size_x', '800')
2204     left_bp.set_attribute('image_size_y', '600')
2205     left_bp.set_attribute('fov', '90')
2206     left_transform = carla.Transform(carla.Location(x=0.325, y=-0.8, z=1.30),
carla.Rotation(yaw=-140))
2207     left_camera = world.spawn_actor(left_bp, left_transform, attach_to=ego_vehicle)
2208     left_camera.listen(sensor_manager.left_callback)
2209     sensors.append(left_camera)
2210
2211     right_bp = blueprint_library.find('sensor.camera.rgb')
2212     right_bp.set_attribute('image_size_x', '800')
2213     right_bp.set_attribute('image_size_y', '600')
2214     right_bp.set_attribute('fov', '90')
2215     right_transform = carla.Transform(carla.Location(x=0.325, y=0.8, z=1.30),
carla.Rotation(yaw=140))
2216     right_camera = world.spawn_actor(right_bp, right_transform, attach_to=ego_vehicle)
2217     right_camera.listen(sensor_manager.right_callback)
2218     sensors.append(right_camera)
2219
2220     # 7. TOP VIEW
2221     topview_bp = blueprint_library.find('sensor.camera.rgb')
2222     topview_bp.set_attribute('image_size_x', '400')
2223     topview_bp.set_attribute('image_size_y', '400')
2224     topview_bp.set_attribute('fov', '70')
2225     topview_transform = carla.Transform(carla.Location(x=0.0, y=0.0, z=10.0),
carla.Rotation(pitch=-90))
2226     topview_camera = world.spawn_actor(topview_bp, topview_transform, attach_to=ego_vehicle)
2227     topview_camera.listen(sensor_manager.topview_callback)
2228     sensors.append(topview_camera)
2229
2230     # 8. RADAR (Increased Range)
2231     radar_bp = blueprint_library.find('sensor.other.radar')
2232     radar_bp.set_attribute('horizontal_fov', '30')
2233     radar_bp.set_attribute('vertical_fov', '30')
2234     radar_bp.set_attribute('range', '150')
2235     radar_transform = carla.Transform(carla.Location(x=2.0, z=1.0))
2236     radar_sensor = world.spawn_actor(radar_bp, radar_transform, attach_to=ego_vehicle)
2237     radar_sensor.listen(radar_processor.radar_callback)
2238     sensors.append(radar_sensor)
2239
2240     # 9. COLLISION & LANE INVASION
2241     collision_bp = blueprint_library.find('sensor.other.collision')
2242     collision_sensor = world.spawn_actor(collision_bp, carla.Transform(), attach_to=ego_vehicle)
2243     collision_sensor.listen(sensor_manager.collision_callback)
2244     sensors.append(collision_sensor)
2245
2246     lane_invasion_bp = blueprint_library.find('sensor.other.lane_invasion')
2247     lane_invasion_sensor = world.spawn_actor(lane_invasion_bp, carla.Transform(),
attach_to=ego_vehicle)
```

```
2248    lane_invasion_sensor.listen(sensor_manager.lane_invasion_callback)
2249    sensors.append(lane_invasion_sensor)
2250
2251    print(f"✅ Sensors attached: {len(sensors)} sensors (Including Bumper Semantic)")
2252    return sensors
```

A-2: Radar Processor and Lane Filtering

```
277 class RadarProcessor:       # Radar data processor
278     def __init__(self):
279         self.detections = []
280         self.last_valid_distance = 100.0
281         self.current_ego_speed = 0.0
282         self.velocity_threshold = -15.6          # FIX: Initialize default threshold for HUD
283         self.tracking_active = False             # Set True when ACC is following a vehicle
284         self.current_curvature = 0.0
285         self.current_yaw_error = 0.0
286
287     def update_current_speed(self, ego_speed):      # Update current vehicle speed to help filter
static objects
288         self.current_ego_speed = ego_speed
289
290     def radar_callback(self, radar_data):       # Process radar data with DYNAMIC CURVE
ADJUSTMENT
291         self.detections = []
292         ego_velocity_ms = self.current_ego_speed / 3.6
293         self.velocity_threshold = -(ego_velocity_ms) + 0.5
294         velocity_limit = max(-35.0, self.velocity_threshold)
295
296         curvature = getattr(self, 'current_curvature', 0.0)
297         yaw_error = getattr(self, 'current_yaw_error', 0.0)
298
299         for detection in radar_data:
300             corrected_azimuth = detection.azimuth + yaw_error
301             Y_raw = detection.depth * math.sin(corrected_azimuth)
302             Y_corrected = Y_raw - (curvature * (detection.depth ** 2))
303
304             # WIDE NET: Catch everything, precise filtering happens in control loop
305             if (abs(Y_corrected) < 4.0 and
306                 detection.depth > 0.1 and
307                 detection.depth < 80.0 and
308                 detection.altitude > -0.05 and
309                 detection.altitude < 0.08):
310
311                 if detection.depth < 20.0 or detection.velocity > velocity_limit or
(self.tracking_active and detection.depth < 40.0):
312                     self.detections.append(detection)
313
314     def get_closest_vehicle_distance(self):     # Get distance to closest vehicle detected by
radar
315         if not self.detections:
316             return self.last_valid_distance
317         closest = min(self.detections, key=lambda d: d.depth)
318         self.last_valid_distance = closest.depth
319         return closest.depth
320
321     def has_vehicle_ahead(self):    # Check if any vehicle is detected ahead
322         return len(self.detections) > 0


3605    # Calculate Time Gap
3606    time_gap = 100.0
3607    if vehicle_detected:
3608        time_gap = measured_distance / max(ego_speed_ms, 0.1)
3609
3610        # Check if the "measured_distance" belongs to a car in our lane
3611        if radar_status == "RADAR_ACTIVE" and radar_processor.detections:
3612            closest = min(radar_processor.detections, key=lambda d: d.depth)
3613
3614            # PERFECT MAP-BASED LANE MATCHING FOR RADAR
3615            ego_trans = ego_vehicle.get_transform()
3616            ego_yaw_rad = math.radians(ego_trans.rotation.yaw)
3617            world_yaw = ego_yaw_rad + closest.azimuth
```

```
3618
3619            target_x = ego_trans.location.x + 2.0 * math.cos(ego_yaw_rad) + closest.depth *
math.cos(world_yaw)
3620            target_y = ego_trans.location.y + 2.0 * math.sin(ego_yaw_rad) + closest.depth *
math.sin(world_yaw)
3621            target_loc = carla.Location(x=target_x, y=target_y, z=ego_trans.location.z)
3622
3623            carla_map = world.get_map()
3624            target_wp = carla_map.get_waypoint(target_loc, project_to_road=True,
lane_type=carla.LaneType.Driving)
3625            ego_wp = carla_map.get_waypoint(ego_trans.location, project_to_road=True,
lane_type=carla.LaneType.Driving)
3626
3627            same_lane = False
3628            if target_wp and ego_wp:
3629                # Rumble strip correction for target and ego
3630                if hasattr(state_tracker, 'highway_road_ids') and
len(state_tracker.highway_road_ids) > 0:
3631                    while target_wp and target_wp.road_id not in state_tracker.highway_road_ids:
3632                        l_wp = target_wp.get_left_lane()
3633                        if l_wp and l_wp.lane_type == carla.LaneType.Driving: target_wp = l_wp
3634                        else: break
3635                    while ego_wp and ego_wp.road_id not in state_tracker.highway_road_ids:
3636                        l_wp = ego_wp.get_left_lane()
3637                        if l_wp and l_wp.lane_type == carla.LaneType.Driving: ego_wp = l_wp
3638                        else: break
3639
3640                if target_wp.road_id == ego_wp.road_id and target_wp.lane_id == ego_wp.lane_id:
3641                    same_lane = True
3642                else:
3643                    # Check ahead in case we are crossing a junction boundary
3644                    next_wps = ego_wp.next(closest.depth + 10.0)
3645                    for nwp in next_wps:
3646                        if nwp.road_id == target_wp.road_id and nwp.lane_id == target_wp.lane_id:
3647                            same_lane = True
3648                            break
3649
3650            if not same_lane:
3651                vehicle_detected = False
3652                time_gap = 100.0
```

A-3: Collision Recovery and Respawning mechanism

```
2468 def collision_recovery_control(ego_vehicle, world, state_tracker, dt):
2469     if not state_tracker.collision_recovery_active:
2470         return False, ""
2471
2472     state_tracker.collision_recovery_timer += dt
2473     phase = state_tracker.collision_recovery_phase
2474     control = carla.VehicleControl()
2475
2476     if phase == "BRAKE":
2477         control.throttle = 0.0
2478         control.brake = 1.0
2479         control.hand_brake = True
2480         control.steer = 0.0
2481         ego_vehicle.apply_control(control)
2482
2483         if state_tracker.collision_recovery_timer >= 0.5:
2484             state_tracker.collision_recovery_phase = "REVERSE"
2485             state_tracker.collision_recovery_timer = 0.0
2486             print("🔄 Collision Recovery: REVERSING...")
2487
2488         return True, "🛡 COLLISION RECOVERY (BRAKING)"
2489
2490     elif phase == "REVERSE":
2491         carla_map = world.get_map()
2492         ego_loc = ego_vehicle.get_transform().location
2493         current_wp = carla_map.get_waypoint(ego_loc, project_to_road=True,
lane_type=carla.LaneType.Driving)
2494
```

```
2495            steer_correction = 0.0
2496            if current_wp:
2497                wp_loc = current_wp.transform.location
2498                ego_fwd = ego_vehicle.get_transform().get_forward_vector()
2499                vec_to_wp = carla.Vector3D(wp_loc.x - ego_loc.x, wp_loc.y - ego_loc.y, 0.0)
2500                cross = ego_fwd.x * vec_to_wp.y - ego_fwd.y * vec_to_wp.x
2501                steer_correction = np.clip(-cross * 0.5, -0.3, 0.3)
2502
2503            control.throttle = 0.4
2504            control.brake = 0.0
2505            control.reverse = True
2506            control.steer = steer_correction
2507            ego_vehicle.apply_control(control)
2508
2509            if state_tracker.collision_recovery_timer >= 2.0:
2510                state_tracker.collision_recovery_phase = "REALIGN"
2511                state_tracker.collision_recovery_timer = 0.0
2512                print("🔄 Collision Recovery: REALIGNING with road...")
2513
2514            return True, "🛡️ COLLISION RECOVERY (REVERSING)"
2515
2516        elif phase == "REALIGN":
2517            carla_map = world.get_map()
2518            ego_loc = ego_vehicle.get_transform().location
2519            current_wp = carla_map.get_waypoint(ego_loc, project_to_road=True,
        lane_type=carla.LaneType.Driving)
2520
2521            if current_wp:
2522                # 🛡️ FORWARD CLEARANCE CHECK: Don't teleport right behind a stopped vehicle
2523                target_wp = current_wp
2524                for attempt in range(4):
2525                    if is_waypoint_clear(world, target_wp, min_clearance=15.0):
2526                        break
2527                    prev_wps = target_wp.previous(8.0)
2528                    if prev_wps:
2529                        target_wp = prev_wps[0]
2530                        print(f"    ↩️ Clearance check: vehicle ahead, moving back {(attempt+1)*8}m")
2531                    else:
2532                        break
2533
2534                new_transform = target_wp.transform
2535                new_transform.location.z = ego_loc.z + 0.5
2536                ego_vehicle.set_transform(new_transform)
2537                ego_vehicle.set_target_velocity(carla.Vector3D(0, 0, 0))
2538
2539            state_tracker.collision_recovery_active = False
2540            state_tracker.collision_recovery_phase = "NONE"
2541            state_tracker.collision_recovery_timer = 0.0
2542            state_tracker.collision_recovery_cooldown = 3.0
2543            state_tracker.post_respawn_warmup = 1.5  # Hold brakes 1.5s for sensor stabilization
2544            state_tracker.speed_pid.reset()
2545            state_tracker.distance_pid.reset()
2546            print("✅ Collision Recovery: COMPLETE - Resuming normal driving")
2547
2548            return True, "🛡️ COLLISION RECOVERY (REALIGNED)"
2549
2550        return False, ""
```

A-4: Wrong Way Detection and Respawning mechanism

```
2425    def check_wrong_way(ego_vehicle, world):
2426        """
2427        Check if the ego vehicle is driving against the road direction.
2428        Returns True if the vehicle is facing the wrong way (>120° from road direction).
2429        """
2430        ego_transform = ego_vehicle.get_transform()
2431        ego_fwd = ego_transform.get_forward_vector()
2432
2433        carla_map = world.get_map()
```

```
2434     current_wp = carla_map.get_waypoint(
2435         ego_transform.location,
2436         project_to_road=True,
2437         lane_type=carla.LaneType.Driving
2438     )
2439
2440     if current_wp is None:
2441         return False
2442
2443     # Get road direction from waypoint
2444     road_fwd = current_wp.transform.get_forward_vector()
2445
2446     # Dot product: 1.0 = same direction, -1.0 = opposite
2447     dot = ego_fwd.x * road_fwd.x + ego_fwd.y * road_fwd.y
2448
2449     # If dot < -0.5 (~120° off), vehicle is going the wrong way
2450     return dot < -0.5
2451
2452 def is_waypoint_clear(world, waypoint, min_clearance=15.0):
2453     """Check if there are any vehicles within min_clearance meters ahead of a waypoint"""
2454     wp_loc = waypoint.transform.location
2455     wp_fwd = waypoint.transform.get_forward_vector()
2456
2457     for actor in world.get_actors().filter('vehicle.*'):
2458         actor_loc = actor.get_location()
2459         dx = actor_loc.x - wp_loc.x
2460         dy = actor_loc.y - wp_loc.y
2461         forward_dist = dx * wp_fwd.x + dy * wp_fwd.y
2462         lateral_dist = abs(-dx * wp_fwd.y + dy * wp_fwd.x)
2463
2464         if 0.0 < forward_dist < min_clearance and lateral_dist < 2.5:
2465             return False
2466     return True
2467
2468 def collision_recovery_control(ego_vehicle, world, state_tracker, dt):
2469     if not state_tracker.collision_recovery_active:
2470         return False, ""
2471
2472     state_tracker.collision_recovery_timer += dt
2473     phase = state_tracker.collision_recovery_phase
2474     control = carla.VehicleControl()
2475
2476     if phase == "BRAKE":
2477         control.throttle = 0.0
2478         control.brake = 1.0
2479         control.hand_brake = True
2480         control.steer = 0.0
2481         ego_vehicle.apply_control(control)
2482
2483         if state_tracker.collision_recovery_timer >= 0.5:
2484             state_tracker.collision_recovery_phase = "REVERSE"
2485             state_tracker.collision_recovery_timer = 0.0
2486             print("🔄 Collision Recovery: REVERSING...")
2487
2488         return True, "🛡 COLLISION RECOVERY (BRAKING)"
2489
2490     elif phase == "REVERSE":
2491         carla_map = world.get_map()
2492         ego_loc = ego_vehicle.get_transform().location
2493         current_wp = carla_map.get_waypoint(ego_loc, project_to_road=True,
lane_type=carla.LaneType.Driving)
2494
2495         steer_correction = 0.0
2496         if current_wp:
2497             wp_loc = current_wp.transform.location
2498             ego_fwd = ego_vehicle.get_transform().get_forward_vector()
2499             vec_to_wp = carla.Vector3D(wp_loc.x - ego_loc.x, wp_loc.y - ego_loc.y, 0.0)
2500             cross = ego_fwd.x * vec_to_wp.y - ego_fwd.y * vec_to_wp.x
2501             steer_correction = np.clip(-cross * 0.5, -0.3, 0.3)
```

```
2502
2503            control.throttle = 0.4
2504            control.brake = 0.0
2505            control.reverse = True
2506            control.steer = steer_correction
2507            ego_vehicle.apply_control(control)
2508
2509            if state_tracker.collision_recovery_timer >= 2.0:
2510                state_tracker.collision_recovery_phase = "REALIGN"
2511                state_tracker.collision_recovery_timer = 0.0
2512                print("🔄 Collision Recovery: REALIGNING with road...")
2513
2514            return True, "🛡 COLLISION RECOVERY (REVERSING)"
2515
2516        elif phase == "REALIGN":
2517            carla_map = world.get_map()
2518            ego_loc = ego_vehicle.get_transform().location
2519            current_wp = carla_map.get_waypoint(ego_loc, project_to_road=True,
lane_type=carla.LaneType.Driving)
2520
2521            if current_wp:
2522                # 🛡 FORWARD CLEARANCE CHECK: Don't teleport right behind a stopped vehicle
2523                target_wp = current_wp
2524                for attempt in range(4):
2525                    if is_waypoint_clear(world, target_wp, min_clearance=15.0):
2526                        break
2527                    prev_wps = target_wp.previous(8.0)
2528                    if prev_wps:
2529                        target_wp = prev_wps[0]
2530                        print(f"   ↩ Clearance check: vehicle ahead, moving back {(attempt+1)*8}m")
2531                    else:
2532                        break
2533
2534                new_transform = target_wp.transform
2535                new_transform.location.z = ego_loc.z + 0.5
2536                ego_vehicle.set_transform(new_transform)
2537                ego_vehicle.set_target_velocity(carla.Vector3D(0, 0, 0))
2538
2539            state_tracker.collision_recovery_active = False
2540            state_tracker.collision_recovery_phase = "NONE"
2541            state_tracker.collision_recovery_timer = 0.0
2542            state_tracker.collision_recovery_cooldown = 3.0
2543            state_tracker.post_respawn_warmup = 1.5  # Hold brakes 1.5s for sensor stabilization
2544            state_tracker.speed_pid.reset()
2545            state_tracker.distance_pid.reset()
2546            print("✅ Collision Recovery: COMPLETE - Resuming normal driving")
2547
2548            return True, "🛡 COLLISION RECOVERY (REALIGNED)"
2549
2550        return False, ""
2551
2552    def wrong_way_correction(ego_vehicle, world, state_tracker):
2553        """
2554        Perform 180° correction when vehicle is facing the wrong way.
2555        Uses set_transform to snap to correct road direction (simulation-safe).
2556        """
2557        carla_map = world.get_map()
2558        ego_loc = ego_vehicle.get_transform().location
2559        current_wp = carla_map.get_waypoint(ego_loc, project_to_road=True,
lane_type=carla.LaneType.Driving)
2560
2561        if current_wp:
2562            print("🔄 WRONG-WAY DETECTED! Performing 180° correction...")
2563
2564            # Align with road direction at current waypoint
2565            new_transform = current_wp.transform
2566            new_transform.location.z = ego_loc.z + 0.5  # Lift slightly
2567
2568            # Stop vehicle first
```

```
2569         ego_vehicle.set_target_velocity(carla.Vector3D(0, 0, 0))
2570         ego_vehicle.set_transform(new_transform)
2571
2572         # Reset PIDs
2573         state_tracker.speed_pid.reset()
2574         state_tracker.distance_pid.reset()
2575         state_tracker.wrong_way_detected = False
2576         state_tracker.wrong_way_correction_active = False
2577
2578         print("✅ 180° Correction COMPLETE - Vehicle re-aligned with traffic flow")
2579         return True
2580
2581     return False
```

Appendix B: ADAS Implementation

B-1: PID Controller Class

```
 57 class PIDController:
 58     def __init__(self, kp, ki, kd, output_limits=(-1.0, 1.0)):
 59         """
 60         PID Controller for vehicle speed control
 61 
 62         Args:
 63             kp: Proportional gain
 64             ki: Integral gain
 65             kd: Derivative gain
 66             output_limits: Tuple of (min, max) output values
 67         """
 68         self.kp = kp
 69         self.ki = ki
 70         self.kd = kd
 71         self.output_limits = output_limits
 72 
 73         self.prev_error = 0.0
 74         self.integral = 0.0
 75 
 76     def update(self, error, dt):
 77         """
 78         Calculate PID output
 79 
 80         Args:
 81             error: Current error (setpoint - measured_value)
 82             dt: Time delta since last update
 83 
 84         Returns:
 85             Control output
 86         """
 87         # Proportional term
 88         p_term = self.kp * error
 89 
 90         # Integral term with anti-windup
 91         self.integral += error * dt
 92         self.integral = np.clip(self.integral, -50.0, 50.0)  # Prevent integral windup
 93         i_term = self.ki * self.integral
 94 
 95         # Derivative term
 96         if dt > 0:
 97             d_term = self.kd * (error - self.prev_error) / dt
 98         else:
 99             d_term = 0.0
100 
101         # Calculate total output
102         output = p_term + i_term + d_term
103 
104         # Apply output limits
105         output = np.clip(output, self.output_limits[0], self.output_limits[1])
106 
107         # Store for next iteration
108         self.prev_error = error
109 
110         return output
111 
112     def reset(self):
113         """Reset PID controller state"""
114         self.prev_error = 0.0
115         self.integral = 0.0
```

B-2: Visual Fusion Processor (Dual-Tier)

```
3196 class VisualFusionProcessor:    # GPU-ACCELERATED Sensor Fusion using 3D Point-Cloud Projection.
Converts 2D pixels into 3D meters using CuPy.
3197     def __init__(self, roof_width=800, roof_fov=90, bumper_width=400, bumper_fov=100):
3198         self.roof_cx = roof_width / 2.0
3199         self.roof_fx = roof_width / (2.0 * math.tan(math.radians(roof_fov) / 2.0))
3200
3201         self.bumper_cx = bumper_width / 2.0
3202         self.bumper_fx = bumper_width / (2.0 * math.tan(math.radians(bumper_fov) / 2.0))
3203
3204         self.distance_history = []
3205         self.history_size = 5
3206
3207     def get_bumper_fusion_distance(self, semantic_bumper_image, depth_bumper_image,
curvature=0.0, yaw_error=0.0):
3208         if semantic_bumper_image is None or depth_bumper_image is None:
3209             return 100.0
3210
3211         h, w = semantic_bumper_image.shape[:2]
3212         roi_top = int(h * 0.15)
3213         roi_bottom = int(h * 0.95)
3214
3215         # 1. Transfer ONLY the ROI to the GPU (Saves PCIe bandwidth)
3216         if USE_GPU:
3217             sem_roi = cp.asarray(semantic_bumper_image[roi_top:roi_bottom, :])
3218             depth_roi = cp.asarray(depth_bumper_image[roi_top:roi_bottom, :])
3219             lib = cp
3220         else:
3221             sem_roi = semantic_bumper_image[roi_top:roi_bottom, :]
3222             depth_roi = depth_bumper_image[roi_top:roi_bottom, :]
3223             lib = np
3224
3225         # 2. Find all vehicle pixels instantly using CUDA cores
3226         b, g, r = sem_roi[:,:,0], sem_roi[:,:,1], sem_roi[:,:,2]
3227         vehicle_mask = (
3228             ((b > 100) & (b < 160) & (g < 50) & (r < 50)) |
3229             ((b > 50) & (b < 90) & (g < 50) & (r < 50))   |
3230             ((b > 80) & (b < 120) & (g > 40) & (g < 80) & (r < 50))
3231         )
3232
3233         y_coords, x_coords = lib.where(vehicle_mask)
3234         if len(x_coords) < 100:
3235             return 100.0
3236
3237         # 3. Get depths for those specific pixels
3238         vehicle_depth_pixels = depth_roi[y_coords, x_coords]
3239         dr = vehicle_depth_pixels[:, 2].astype(lib.float32)
3240         dg = vehicle_depth_pixels[:, 1].astype(lib.float32)
3241         db = vehicle_depth_pixels[:, 0].astype(lib.float32)
3242
3243         normalized = (dr + dg * 256.0 + db * 65536.0) / 16777215.0
3244         depth_meters = 1000.0 * normalized
3245
3246         # 4. CONVERT TO 3D LATERAL OFFSET (Vectorized on GPU)
3247         lateral_distances = (x_coords - self.bumper_cx) * depth_meters / self.bumper_fx
3248
3249         # 5. Correct for ego vehicle yaw AND road curvature
3250         yaw_corrected_lateral = lateral_distances + (depth_meters * math.sin(yaw_error))
3251         path_adjusted_lateral = yaw_corrected_lateral - (curvature * (depth_meters ** 2))
3252
3253         # 6. Filter: Keep only pixels strictly inside our lane (+/- 1.8m)
3254         valid_idx = (lib.abs(path_adjusted_lateral) < 1.8) & (depth_meters > 0.1) & (depth_meters
< 30.0)
3255         valid_depths = depth_meters[valid_idx]
3256
3257         if len(valid_depths) < 50:
3258             return 100.0
3259
3260         # 7. Pull final percentile float back to CPU memory
3261         ans = lib.percentile(valid_depths, 10)
3262         return float(ans.get()) if USE_GPU else float(ans)
3263
```

```
3264     def get_roof_fusion_distance(self, semantic_image, depth_raw_image, curvature=0.0,
yaw_error=0.0):
3265         if semantic_image is None or depth_raw_image is None:
3266             return 100.0
3267 
3268         h, w = semantic_image.shape[:2]
3269         roi_top = int(h * 0.25)
3270         roi_bottom = int(h * 0.75)
3271 
3272         if USE_GPU:
3273             sem_roi = cp.asarray(semantic_image[roi_top:roi_bottom, :])
3274             depth_roi = cp.asarray(depth_raw_image[roi_top:roi_bottom, :])
3275             lib = cp
3276         else:
3277             sem_roi = semantic_image[roi_top:roi_bottom, :]
3278             depth_roi = depth_raw_image[roi_top:roi_bottom, :]
3279             lib = np
3280 
3281         b, g, r = sem_roi[:,:,0], sem_roi[:,:,1], sem_roi[:,:,2]
3282         vehicle_mask = (
3283             ((b > 100) & (b < 160) & (g < 50) & (r < 50)) |
3284             ((b > 50) & (b < 90) & (g < 50) & (r < 50))   |
3285             ((b > 80) & (b < 120) & (g > 40) & (g < 80) & (r < 50))
3286         )
3287 
3288         y_coords, x_coords = lib.where(vehicle_mask)
3289         if len(x_coords) < 30:
3290             return 100.0
3291 
3292         vehicle_depth_pixels = depth_roi[y_coords, x_coords]
3293         dr = vehicle_depth_pixels[:, 2].astype(lib.float32)
3294         dg = vehicle_depth_pixels[:, 1].astype(lib.float32)
3295         db = vehicle_depth_pixels[:, 0].astype(lib.float32)
3296 
3297         normalized = (dr + dg * 256.0 + db * 65536.0) / 16777215.0
3298         depth_meters = 1000.0 * normalized
3299 
3300         lateral_distances = (x_coords - self.roof_cx) * depth_meters / self.roof_fx
3301 
3302         yaw_corrected_lateral = lateral_distances + (depth_meters * math.sin(yaw_error))
3303         path_adjusted_lateral = yaw_corrected_lateral - (curvature * (depth_meters ** 2))
3304 
3305         # STRICT CORRIDOR: Cap camera expansion to strictly prevent adjacent lane bleeding
3306         dynamic_expansion = 1.6 + (depth_meters / 50.0) * 0.8
3307         dynamic_width = lib.minimum(dynamic_expansion, 2.6)
3308         valid_idx = (lib.abs(path_adjusted_lateral) < dynamic_width) & (depth_meters > 10.0) &
(depth_meters < 80.0)
3309         valid_depths = depth_meters[valid_idx]
3310 
3311         if len(valid_depths) < 20:
3312             return 100.0
3313 
3314         ans = lib.percentile(valid_depths, 10)
3315         return float(ans.get()) if USE_GPU else float(ans)
3316 
3317     def get_fusion_distance(self, semantic_image, depth_raw_image,
3318                             semantic_bumper_image=None, depth_bumper_image=None, curvature=0.0,
yaw_error=0.0):
3319         bumper_dist = self.get_bumper_fusion_distance(semantic_bumper_image, depth_bumper_image,
curvature, yaw_error)
3320         roof_dist = self.get_roof_fusion_distance(semantic_image, depth_raw_image, curvature,
yaw_error)
3321 
3322         if 15.0 < bumper_dist < 25.0 and roof_dist < 80.0:
3323             weight_roof = (bumper_dist - 15.0) / 10.0
3324             final_dist = (bumper_dist * (1.0 - weight_roof)) + (roof_dist * weight_roof)
3325         elif bumper_dist <= 15.0:
3326             final_dist = bumper_dist
3327         elif roof_dist < 80.0:
3328             final_dist = roof_dist
3329         else:
3330             final_dist = 100.0
3331 
3332         self.distance_history.append(final_dist)
3333         if len(self.distance_history) > self.history_size:
```

```
3334                self.distance_history.pop(0)
```

B-3: Timing + Fusion Persistence Configuration

- ## B-3a: Timing Configuration

```
 19 class TimingConfig:
 20     """All timing constants in SECONDS for FPS-independent operation"""
 21 
 22     # Detection Persistence
 23     FUSION_PERSISTENCE_TIME = 0.17          # 0.17 sec (time-based, FPS-independent)
 24     FUSION_PERSISTENCE_AT_LIGHT = 1.5       # 1.5 sec (time-based, FPS-independent)
 25     MIN_FUSION_DETECTION_TIME = 0.067       # 0.067 sec (time-based, FPS-independent)
 26     FUSION_PERSISTENCE_ACC_FOLLOW = 1.0     # 1.0 sec when ACC was actively following
 27 
 28     # Distance Filtering
 29     DISTANCE_HISTORY_WINDOW = 0.17          # 0.17 sec rolling window
 30     MAX_DISTANCE_CHANGE_RATE = 240.0        # 240 m/sec (rate-based, FPS-independent)
 31 
 32     # Steering Smoothing
 33     STEERING_SMOOTHING_TAU = 0.10           # EMA time constant (seconds)
 34     MAX_STEERING_RATE = 3.0                 # Max steering change per second
 35     LANE_CENTER_HISTORY_WINDOW = 0.17       # 0.17 sec history
 36 
 37     # Other Timers (already time-based, just centralizing)
 38     LANE_CHANGE_COOLDOWN = 12.0             # seconds
```

- ## B-3b: Fusion Persistence Configuration

```
3356     # B. Fusion Priority Logic with TIME-BASED PERSISTENCE
3357     if fusion_dist < 60.0:
3358         # Fusion sees vehicle - accumulate detection duration
3359         state_tracker.fusion_detection_duration += dt
3360         state_tracker.time_since_detection = 0.0  # Reset dropout timer
3361 
3362         # Trust fusion after minimum detection time
3363         if state_tracker.fusion_detection_duration >= TimingConfig.MIN_FUSION_DETECTION_TIME:
3364             vehicle_detected = True
3365             measured_distance = fusion_dist
3366             radar_status = "FUSION_DETECT"
3367         else:
3368             # Not enough consecutive detection time yet, check radar
3369             if radar_processor.has_vehicle_ahead():
3370                 vehicle_detected = True
3371                 measured_distance = radar_dist
3372                 radar_status = "RADAR_ACTIVE"
3373     else:
3374         # Fusion doesn't see vehicle this frame - accumulate dropout time
3375         state_tracker.time_since_detection += dt
3376 
3377         # ═══════════════════════════════════════════════════════════════
3378         # 🔧 TIME-BASED PERSISTENCE: Allow dropout for persistence time
3379         # ═══════════════════════════════════════════════════════════════
3380         if state_tracker.was_stopping_for_light:
3381             persistence_time = TimingConfig.FUSION_PERSISTENCE_AT_LIGHT
3382         elif state_tracker.was_acc_following:
3383             persistence_time = TimingConfig.FUSION_PERSISTENCE_ACC_FOLLOW
3384         else:
3385             persistence_time = TimingConfig.FUSION_PERSISTENCE_TIME
3386 
3387         if state_tracker.time_since_detection <= persistence_time and
state_tracker.fusion_detection_duration >= TimingConfig.MIN_FUSION_DETECTION_TIME:
3388             # Recently had valid fusion, keep tracking with last known distance
3389             vehicle_detected = True
3390             measured_distance = state_tracker.last_valid_distance
3391             radar_status = "FUSION_PERSIST"
```

```
3392         else:
3393             # Truly lost - reset duration
3394             state_tracker.fusion_detection_duration = 0.0
3395 
3396             # Fallback to radar
3397             if radar_processor.has_vehicle_ahead():
3398                 vehicle_detected = True
3399                 measured_distance = radar_dist
3400                 radar_status = "RADAR_ACTIVE"
3401 
3402     # Store last valid distance for persistence
3403     # 🛡️ SAFETY NET: A close vehicle doesn't just vanish
3404     # If ACC was actively following a vehicle at close range and ALL sensors lost it,
3405     # maintain detection for 3 seconds. This overrides persistence failures.
3406     if not vehicle_detected and state_tracker.was_acc_following and
state_tracker.last_valid_distance < 30.0 and state_tracker.time_since_detection < 5.0:
3407         vehicle_detected = True
3408         measured_distance = state_tracker.last_valid_distance
3409         radar_status = "SAFETY_PERSIST"
3410     if vehicle_detected and measured_distance < 80.0:
3411         state_tracker.last_valid_distance = measured_distance
```

B-4: Three Zone ACC + Emergency hierarchy (7 Priority Tiers)

```
3604     # STEP 3: CONTROL DECISION (3-STAGE LOGIC)
3605     # Calculate Time Gap
3606     time_gap = 100.0
3607     if vehicle_detected:
3608         time_gap = measured_distance / max(ego_speed_ms, 0.1)
3609 
3610         # Check if the "measured_distance" belongs to a car in our lane
3611         if radar_status == "RADAR_ACTIVE" and radar_processor.detections:
3612             closest = min(radar_processor.detections, key=lambda d: d.depth)
3613 
3614             # PERFECT MAP-BASED LANE MATCHING FOR RADAR
3615             ego_trans = ego_vehicle.get_transform()
3616             ego_yaw_rad = math.radians(ego_trans.rotation.yaw)
3617             world_yaw = ego_yaw_rad + closest.azimuth
3618 
3619             target_x = ego_trans.location.x + 2.0 * math.cos(ego_yaw_rad) + closest.depth *
math.cos(world_yaw)
3620             target_y = ego_trans.location.y + 2.0 * math.sin(ego_yaw_rad) + closest.depth *
math.sin(world_yaw)
3621             target_loc = carla.Location(x=target_x, y=target_y, z=ego_trans.location.z)
3622 
3623             carla_map = world.get_map()
3624             target_wp = carla_map.get_waypoint(target_loc, project_to_road=True,
lane_type=carla.LaneType.Driving)
3625             ego_wp = carla_map.get_waypoint(ego_trans.location, project_to_road=True,
lane_type=carla.LaneType.Driving)
3626 
3627             same_lane = False
3628             if target_wp and ego_wp:
3629                 # Rumble strip correction for target and ego
3630                 if hasattr(state_tracker, 'highway_road_ids') and
len(state_tracker.highway_road_ids) > 0:
3631                     while target_wp and target_wp.road_id not in state_tracker.highway_road_ids:
3632                         l_wp = target_wp.get_left_lane()
3633                         if l_wp and l_wp.lane_type == carla.LaneType.Driving: target_wp = l_wp
3634                         else: break
3635                     while ego_wp and ego_wp.road_id not in state_tracker.highway_road_ids:
3636                         l_wp = ego_wp.get_left_lane()
3637                         if l_wp and l_wp.lane_type == carla.LaneType.Driving: ego_wp = l_wp
3638                         else: break
3639 
3640                 if target_wp.road_id == ego_wp.road_id and target_wp.lane_id == ego_wp.lane_id:
3641                     same_lane = True
3642                 else:
3643                     # Check ahead in case we are crossing a junction boundary
3644                     next_wps = ego_wp.next(closest.depth + 10.0)
3645                     for nwp in next_wps:
3646                         if nwp.road_id == target_wp.road_id and nwp.lane_id == target_wp.lane_id:
3647                             same_lane = True
```

```
3648                              break
3649
3650                if not same_lane:
3651                    vehicle_detected = False
3652                    time_gap = 100.0
3653
3654        mode = "CRUISE"
3655
3656        # PRIORITY 1: ABSOLUTE EMERGENCY (At speed) - ALWAYS brake first regardless of lights
3657        if vehicle_detected and measured_distance < 5.0 and ego_speed > 10.0:
3658            if state_tracker.fusion_detection_duration >= TimingConfig.MIN_FUSION_DETECTION_TIME or
radar_status == "RADAR_ACTIVE":
3659                mode = f"HARD STOP ({measured_distance:.1f}m)"
3660                control.throttle = 0.0
3661                control.brake = 1.0
3662                control.hand_brake = False
3663                state_tracker.speed_pid.reset()
3664                state_tracker.distance_pid.reset()
3665
3666        # PRIORITY 2: EMERGENCY BRAKE (Low speed)
3667        elif vehicle_detected and measured_distance < 5.0 and ego_speed <= 10.0:
3668            if ego_speed > 3.0:
3669                mode = "STOPPED TOO CLOSE"
3670                control.throttle = 0.0
3671                control.brake = 1.0
3672                control.hand_brake = False
3673            else:
3674                is_vehicle, source = verify_vehicle_ahead_semantic(sensor_manager, measured_distance,
radar_status, fusion_processor)
3675                if is_vehicle:
3676                    mode = f"STOPPED TOO CLOSE [VEH:{source}]"
3677                    control.throttle = 0.0
3678                    control.brake = 1.0
3679                    control.hand_brake = False
3680                else:
3681                    vehicle_detected = False
3682                    measured_distance = 100.0
3683                    time_gap = 100.0
3684                    mode = "CRUISE"
3685
3686        # PRIORITY 3: TIME-GAP EMERGENCY (< 0.75s)
3687        elif vehicle_detected and time_gap < 0.75:
3688            if state_tracker.fusion_detection_duration >= TimingConfig.MIN_FUSION_DETECTION_TIME or
radar_status == "RADAR_ACTIVE":
3689                mode = f"EMERGENCY ({time_gap:.1f}s)"
3690                control.throttle = 0.0
3691                # Aggressive braking: closer = harder brake
3692                urgency = np.clip((1.2 - time_gap) / 0.5, 0.5, 1.0)
3693                control.brake = urgency
3694                state_tracker.speed_pid.reset()
3695                state_tracker.distance_pid.reset()
3696
3697        # PRIORITY 4: RED LIGHT WITH LEAD VEHICLE
3698        elif stop_for_light and vehicle_detected and stop_target_dist < 50.0:
3699            # The vehicle ahead is likely stopped at the light
3700            # Use the CLOSER of: vehicle distance or stop line distance
3701            effective_stop_dist = min(measured_distance - 2.5, stop_target_dist)
3702
3703            if effective_stop_dist < 2.0 or state_tracker.holding_at_line:
3704                mode = "STOPPED BEHIND CAR"
3705                control.throttle = 0.0
3706                control.brake = 1.0
3707                control.hand_brake = True
3708                state_tracker.holding_at_line = True
3709            else:
3710                # Gentle approach to car/line
3711                req_speed = math.sqrt(2 * 2.5 * effective_stop_dist)
3712                req_speed = max(req_speed, 1.5)
3713
3714                if ego_speed_ms > req_speed:
3715                    control.throttle = 0.0
3716                    control.brake = np.clip((ego_speed_ms - req_speed) / 2.0, 0.4, 1.0)
3717                else:
3718                    control.throttle = 0.2
3719                    control.brake = 0.0
```

```
3720                mode = f"APPROACH ({effective_stop_dist:.1f}m)"
3721
3722        # PRIORITY 5: ACC / FOLLOWING (2.0s - 4.0s)
3723        elif vehicle_detected and (time_gap < 5.0 or measured_distance < 40.0) and
state_tracker.acc_on:
3724            # 🎯 TARGET: 2.5 second gap (adjustable)
3725            TARGET_GAP_SECONDS = 2.5
3726            GAP_DEADBAND = 0.3  # ±0.3s tolerance (2.2s - 2.8s is "perfect")
3727
3728            target_dist = max(ego_speed_ms * TARGET_GAP_SECONDS, 5.0)
3729            error = measured_distance - target_dist
3730
3731            gap_error = time_gap - TARGET_GAP_SECONDS   # Calculate gap error in seconds
3732
3733            # THREE-ZONE CONTROL
3734            if abs(gap_error) < GAP_DEADBAND:
3735                mode = f"ACC HOLD ({time_gap:.1f}s)"    # ZONE A: HOLDING ZONE (2.2s - 2.8s)
3736
3737                # Very gentle correction only
3738                if gap_error > 0.1:  # Slightly too far
3739                    control.throttle = 0.3
3740                    control.brake = 0.0
3741                elif gap_error < -0.1:  # Slightly too close
3742                    control.throttle = 0.0
3743                    control.brake = 0.1
3744                else:  # Perfect - coast
3745                    control.throttle = 0.15  # Maintenance throttle
3746                    control.brake = 0.0
3747
3748            elif gap_error > 0:
3749                mode = f"ACC CLOSE ({time_gap:.1f}s)"   # ZONE B: TOO FAR (gap > 2.8s) - Need to
close gap
3750
3751                # Proportional closing - faster when further away
3752                closing_intensity = min(gap_error / 1.5, 1.0)  # Max at 1.5s over target
3753                control.throttle = 0.4 + (closing_intensity * 0.4)  # 0.4 to 0.8
3754                control.brake = 0.0
3755
3756            else:
3757                mode = f"ACC OPEN ({time_gap:.1f}s)"    # ZONE C: TOO CLOSE (gap < 2.2s) - Need to
open gap
3758
3759                # Proportional braking - harder when closer
3760                opening_intensity = min(abs(gap_error) / 1.0, 1.0)  # Max at 1.0s under target
3761                control.throttle = 0.0
3762                control.brake = 0.2 + (opening_intensity * 0.3)  # 0.2 to 0.5
3763
3764        # PRIORITY 6: TRAFFIC LIGHT STOP
3765        elif stop_for_light:
3766            mode = f"LIGHT STOP ({stop_target_dist:.1f}m)"
3767
3768            if stop_target_dist < 0.5:
3769                control.throttle = 0.0
3770                control.brake = 1.0
3771                control.hand_brake = True
3772                mode = "HOLDING AT LINE"
3773            elif stop_target_dist < 5.0:
3774                # Precision approach
3775                req_speed = math.sqrt(2 * 0.5 * stop_target_dist) # Very gentle approach
3776                if ego_speed_ms > req_speed:
3777                    control.throttle = 0.0
3778                    control.brake = 0.2
3779                else:
3780                    control.throttle = 0.15
3781                    control.brake = 0.0
3782            else:
3783                # Normal approach logic
3784                req_speed_ms = math.sqrt(2 * 1.5 * stop_target_dist)
3785                target_kmh = req_speed_ms * 3.6
3786                # ANTI-SURGE: Never floor the gas just to reach a stop line faster
3787                if ego_speed < target_kmh:
3788                    control.throttle = 0.35  # Minimal "creep" throttle
3789                    control.brake = 0.0
3790                else:
3791                    pid_out = state_tracker.speed_pid.update(target_kmh - ego_speed, dt)
```

```
3792                control.throttle = 0.0
3793                control.brake = abs(pid_out)
3794
3795    # PRIORITY 7: FREE CRUISE
3796    else:
3797        state_tracker.distance_pid.reset()
3798
3799        # SAFE CRUISE TRANSITION: Don't accelerate if ACC/EMERGENCY was active recently
3800        if state_tracker.ticks_since_acc < 20 and ego_speed > 15.0:
3801            mode = "CRUISE (COAST)"
3802            control.throttle = 0.0
3803            control.brake = 0.10  # Gentle deceleration, DO NOT run PID
3804        else:
3805            mode = "CRUISE"
3806
3807            if state_tracker.speed_limiter_on:
3808                target = state_tracker.target_speed
3809            else:
3810                target = 150.0
3811
3812            # POST-STOP RESUME: After red light stop, cap speed until lead vehicle reacquired
3813            if state_tracker.traffic_light_assist_on and state_tracker.time_since_light_stop <
15.0:
3814                target = min(target, 80.0)
3815
3816            pid_out = state_tracker.speed_pid.update(target - ego_speed, dt)
3817
3818            if pid_out > 0:
3819                control.throttle = np.clip(pid_out, 0.0, 1.0)
3820                control.brake = 0.0
3821            else:
3822                control.throttle = 0.0
3823                control.brake = np.clip(abs(pid_out), 0.0, 0.2)
3824
3825    # STEERING CONTROL (Respects LCA Toggle)
3826    if state_tracker.lane_assist_on:
3827        # LCA ON: Full waypoint-based lane centering using dynamic lookahead
3828        control.steer = get_waypoint_steering(ego_vehicle, world, state_tracker)
3829        mode = mode + " + LCA"
3830    else:
3831        # LAZY DRIVER (LCA OFF)
3832        # Lookahead 8.0m (shorter) so it reacts to curves very late
3833        raw_steer = get_waypoint_steering(ego_vehicle, world, state_tracker, lookahead=8.0)
3834
3835        # INCREASED Deadband: human ignores noticeable drift (< ~15cm) before correcting
3836        if abs(raw_steer) < 0.04:
3837            raw_steer = 0.0
3838
3839        # SLUGGISH gain: human steering is sluggish and poorly optimized compared to ADAS
3840        control.steer = raw_steer * 0.45
3841        mode = mode + " (NO LCA)"
3842
3843    # POST-STOP RESUME PROTOCOL: Track time since last red light stop
3844    if stop_for_light and ego_speed < 5.0:
3845        state_tracker.time_since_light_stop = 0.0
3846    else:
3847        state_tracker.time_since_light_stop += dt
3848
3849    ego_vehicle.apply_control(control)
3850    # Update ticks_since_acc: reset to 0 when in any vehicle-tracking mode, increment otherwise
3851    if "ACC" in mode or "EMERGENCY" in mode or "HARD STOP" in mode or "STOPPED" in mode:
3852        state_tracker.ticks_since_acc = 0
3853    else:
3854        state_tracker.ticks_since_acc += 1
3855
3856    state_tracker.was_stopping_for_light = stop_for_light
3857    if vehicle_detected and state_tracker.acc_on:
3858        state_tracker.was_acc_following = True
3859    elif state_tracker.time_since_detection > 5.0:
3860        state_tracker.was_acc_following = False
3861    radar_processor.tracking_active = (state_tracker.fusion_detection_duration >=
TimingConfig.MIN_FUSION_DETECTION_TIME)
3862
3863    return measured_distance, ego_speed, radar_status, vehicle_detected, mode
```

B-5: LCA Module

```
2350 def get_waypoint_steering(vehicle, world, lookahead=None, smoothing=True):
2351     """
2352     Calculate steering angle with DYNAMIC LOOKAHEAD for 60-150 km/h stability.
2353     """
2354     ego_transform = vehicle.get_transform()
2355     ego_location = ego_transform.location
2356 
2357     # 1. Get Dynamic Speed
2358     v = vehicle.get_velocity()
2359     ego_speed_kmh = 3.6 * math.sqrt(v.x**2 + v.y**2 + v.z**2)
2360 
2361     # 2. DYNAMIC LOOKAHEAD CALCULATION
2362     # If lookahead is not manually forced, calculate it based on speed.
2363     # Logic: At 60 km/h -> ~24m lookahead. At 150 km/h -> ~60m lookahead.
2364     if lookahead is None:
2365         lookahead = np.clip(ego_speed_kmh * 0.4, 12.0, 65.0)
2366 
2367     carla_map = world.get_map()
2368     current_waypoint = carla_map.get_waypoint(
2369         ego_location,
2370         project_to_road=True,
2371         lane_type=carla.LaneType.Driving
2372     )
2373 
2374     if current_waypoint is None:
2375         return 0.0
2376 
2377     # Look ahead
2378     next_waypoints = current_waypoint.next(lookahead)
2379 
2380     if next_waypoints:
2381         target_waypoint = next_waypoints[0]
2382         target_location = target_waypoint.transform.location
2383 
2384         # Calculate target vector
2385         target_vector = np.array([
2386             target_location.x - ego_location.x,
2387             target_location.y - ego_location.y
2388         ])
2389 
2390         # Normalize
2391         target_distance = np.linalg.norm(target_vector)
2392         if target_distance < 0.1: return 0.0
2393         target_vector = target_vector / target_distance
2394 
2395         # Calculate angle
2396         yaw_rad = np.radians(ego_transform.rotation.yaw)
2397         forward_vector = np.array([np.cos(yaw_rad), np.sin(yaw_rad)])
2398 
2399         cross = np.cross(forward_vector, target_vector)
2400         dot = np.dot(forward_vector, target_vector)
2401         angle = np.arctan2(cross, dot)
2402 
2403         # 3. DYNAMIC STEERING GAIN (Stiffness)
2404         # Low speed (start/city) needs aggressive steering (1.2)
2405         # High speed (highway 150kmh) needs gentle steering (0.3) to avoid spinning
2406         # Formula: Interpolate between 1.2 and 0.3 based on speed
2407         if ego_speed_kmh < 40:
2408             gain = 1.2
2409         else:
2410             # Gradually reduce gain as speed increases
2411             # At 150kmh, gain is approx 0.35
2412             gain = max(0.35, 1.2 - (ego_speed_kmh - 40) * 0.008)
2413 
2414         steering = angle * gain
2415 
2416         # Apply smoothing/damping based on angle magnitude
```

```
2417        if smoothing:
2418            damping_factor = 1.0 - (abs(angle) / np.pi) * 0.3
2419            steering = steering * damping_factor
2420
2421        return np.clip(steering, -1.0, 1.0)
2422
2423    return 0.0
```

B-6: Traffic Light Braking (Map-Based Detection & Heuristic)

- **B-6a: Traffic Light Map-based Detection**

```
3047 def detect_traffic_light_map_based(ego_vehicle, world, ego_speed):
3048     """
3049     Map-based traffic light detection with INCREASED RANGE for highway speeds
3050     """
3051     carla_map = world.get_map()
3052     ego_loc = ego_vehicle.get_location()
3053     ego_transform = ego_vehicle.get_transform()
3054     ego_fwd = ego_transform.get_forward_vector()
3055
3056     # 1. DYNAMIC SCAN DISTANCE (CRITICAL FIX)
3057     # Was 3.0s (too short). Increased to 6.0s for highway safety.
3058     # At 80 km/h, this scans ~135m ahead instead of 66m.
3059     scan_distance = max(80.0, (ego_speed / 3.6) * 6.0)
3060
3061     # 2. IMMEDIATE CHECK (Trigger Volume)
3062     if ego_vehicle.is_at_traffic_light():
3063         traffic_light = ego_vehicle.get_traffic_light()
3064         if traffic_light and traffic_light.get_state() != carla.TrafficLightState.Green:
3065             stop_waypoints = traffic_light.get_stop_waypoints()
3066             if stop_waypoints:
3067                 stop_dist = ego_loc.distance(stop_waypoints[0].transform.location)
3068                 return traffic_light, stop_dist, traffic_light.get_state(), "IMMEDIATE_TRIGGER"
3069
3070     # 3. GET CURRENT LANE
3071     current_waypoint = carla_map.get_waypoint(
3072         ego_loc,
3073         project_to_road=True,
3074         lane_type=carla.LaneType.Driving
3075     )
3076
3077     if not current_waypoint:
3078         return None, 1000.0, None, "NO_WAYPOINT"
3079
3080     # 4. SCAN FORWARD
3081     scan_waypoint = current_waypoint
3082     closest_light = None
3083     min_dist = 1000.0
3084
3085     # Optimization: Get all lights once
3086     all_lights = world.get_actors().filter('traffic.traffic_light*')
3087
3088     # Scan step size (2.0m is faster than 1.0m and accurate enough)
3089     for i in range(0, int(scan_distance), 2):
3090         next_waypoints = scan_waypoint.next(2.0)
3091         if not next_waypoints:
3092             break
3093
3094         scan_waypoint = next_waypoints[0]
3095
3096         # Check if we found a junction
3097         if scan_waypoint.is_junction:
3098             for light in all_lights:
3099                 # FILTER 1: Lane Matching
3100                 # Does this light affect the lane we are simulating ahead?
3101                 affected_waypoints = light.get_affected_lane_waypoints()
3102                 is_relevant = False
```

```
3103
3104                    for wp in affected_waypoints:
3105                        # Match Road ID and Lane ID (approximate match)
3106                        if (wp.road_id == scan_waypoint.road_id and
3107                            wp.lane_id == scan_waypoint.lane_id):
3108                            is_relevant = True
3109                            break
3110
3111                    if not is_relevant:
3112                        continue
3113
3114                    # FILTER 2: Geometric Check (Is it actually in front?)
3115                    light_loc = light.get_location()
3116                    vec_to_light = light_loc - ego_loc
3117                    vec_len = math.sqrt(vec_to_light.x**2 + vec_to_light.y**2)
3118
3119                    if vec_len > 0:
3120                        dot = (vec_to_light.x * ego_fwd.x + vec_to_light.y * ego_fwd.y) / vec_len
3121                        if dot > 0.2: # Broad 70-degree cone ahead
3122                            # FOUND RELEVANT LIGHT
3123                            stop_waypoints = light.get_stop_waypoints()
3124                            if stop_waypoints:
3125                                dist = ego_loc.distance(stop_waypoints[0].transform.location)
3126                                if dist < min_dist:
3127                                    min_dist = dist
3128                                    closest_light = light
3129
3130                # If we found a light, we can stop scanning (closest one wins)
3131                if closest_light:
3132                    break
3133
3134        if closest_light:
3135            return closest_light, min_dist, closest_light.get_state(), "MAP_SCAN"
3136
3137        return None, 1000.0, None, "NO_LIGHT"
```

- **B-6b: Traffic Light Heuristic**

```
2988 def detect_traffic_light_rgb_heuristic(rgb_image, debug=False):
2989     """
2990     Heuristic to detect Red/Yellow traffic lights using RGB Image.
2991     IMPROVED V2: Lowered area threshold to 2px for long-distance detection.
2992     """
2993     if rgb_image is None:
2994         return False, "NONE"
2995
2996     # 1. ROI: Look at the upper 60% of the screen
2997     height, width = rgb_image.shape[:2]
2998     roi = rgb_image[0:int(height*0.6), :]
2999
3000     # 2. Convert to HSV
3001     hsv = cv2.cvtColor(roi, cv2.COLOR_RGB2HSV)
3002
3003     # 3. Define Thresholds
3004     # Lowered Saturation/Value to 80 to catch dimmer/farther lights
3005     lower_red1 = np.array([0, 80, 80])
3006     upper_red1 = np.array([10, 255, 255])
3007     lower_red2 = np.array([170, 80, 80])
3008     upper_red2 = np.array([180, 255, 255])
3009
3010     lower_yellow = np.array([20, 80, 80])
3011     upper_yellow = np.array([35, 255, 255])
3012
3013     # 4. Create Masks
3014     mask_r1 = cv2.inRange(hsv, lower_red1, upper_red1)
3015     mask_r2 = cv2.inRange(hsv, lower_red2, upper_red2)
3016     mask_red = cv2.bitwise_or(mask_r1, mask_r2)
```

```
3017
3018        mask_yellow = cv2.inRange(hsv, lower_yellow, upper_yellow)
3019
3020        # 5. CONTOUR ANALYSIS
3021        def process_contours(mask, color_name):
3022            contours, _ = cv2.findContours(mask, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
3023            for contour in contours:
3024                area = cv2.contourArea(contour)
3025
3026                # FIXED: Reduced min area from 10 to 3.
3027                # At 50m, a light is barely a dot.
3028                # Upper limit 1000 avoids Sun/Glare.
3029                if area < 3 or area > 1000:
3030                    continue
3031
3032                x, y, w, h = cv2.boundingRect(contour)
3033                aspect_ratio = float(w) / h
3034
3035                # Relaxed Aspect Ratio to account for pixelation at distance
3036                if 0.3 <= aspect_ratio <= 2.0:
3037                    return True
3038            return False
3039
3040        if process_contours(mask_red, "RED"):
3041            return True, "RED"
3042        if process_contours(mask_yellow, "YELLOW"):
3043            return True, "YELLOW"
3044
3045        return False, "GREEN/NONE"
```

Appendix C: Simulation Criteria and Output Extraction

C-1: Camera Degradation due to weather

```
2278 def degrade_camera_image(image, weather_name, apply_cockpit_mask=True):
2279     """
2280     GPU-ACCELERATED Weather Degradation
2281     Generates noise on GPU to prevent CPU starvation.
2282     """
2283     height, width = image.shape[:2]
2284 
2285     # 1. Transfer to GPU
2286     if USE_GPU:
2287         img_gpu = cp.asarray(image, dtype=cp.float32)
2288     else:
2289         img_gpu = image.astype(np.float32)
2290 
2291     # 2. Handle Cockpit Mask Caching
2292     mask_key = f"cockpit_{height}_{width}_{apply_cockpit_mask}"
2293 
2294     if mask_key in MASK_CACHE:
2295         mask_3ch = MASK_CACHE[mask_key]
2296     else:
2297         if apply_cockpit_mask:
2298             mask_cpu = np.zeros((height, width), dtype=np.float32)
2299             glass_y_end = int(height * 0.85)
2300             mask_cpu[0:glass_y_end, :] = 1.0
2301 
2302             pillar_width = int(width * 0.18)
2303             pillar_height = int(height * 0.35)
2304             Y, X = np.ogrid[:height, :width]
2305             y_top = glass_y_end - pillar_height
2306             slope = pillar_height / pillar_width
2307 
2308             pillar_zone = (Y > (slope * X + y_top)) & (X < pillar_width) & (Y < glass_y_end)
2309             mask_cpu[pillar_zone] = 0.0
2310             mask_3ch_cpu = np.stack([mask_cpu] * 3, axis=2)
2311         else:
2312             mask_3ch_cpu = np.ones((height, width, 3), dtype=np.float32)
2313 
2314         if USE_GPU:
2315             mask_3ch = cp.asarray(mask_3ch_cpu)
2316         else:
2317             mask_3ch = mask_3ch_cpu
2318         MASK_CACHE[mask_key] = mask_3ch
2319 
2320     # Separate road view
2321     road_view = img_gpu * mask_3ch
2322     clean_interior = img_gpu * (1.0 - mask_3ch) if apply_cockpit_mask else 0.0
2323     degraded = road_view
2324 
2325     # 3. GPU Noise Generation (The biggest speedup)
2326     if "Heavy Rain" in weather_name:
2327         # Gaussian blur on GPU (using CuPy/OpenCV fallback) or simple box blur
2328         # Simple noise addition is much faster on GPU
2329         noise = cp.random.normal(0, 15, degraded.shape, dtype=cp.float32)
2330         degraded = degraded + noise
2331 
2332         # Simple desaturation on GPU
2333         # (Approximate HSV desaturation using linear algebra to save time)
2334         gray = degraded[:,:,0]*0.299 + degraded[:,:,1]*0.587 + degraded[:,:,2]*0.114
2335         gray = cp.stack([gray]*3, axis=2)
2336         degraded = degraded * 0.5 + gray * 0.5 # 50% desaturation
2337 
2338     elif "Night" in weather_name:
2339         degraded = degraded * 0.60 + 20.0
2340         noise = cp.random.normal(0, 20, degraded.shape, dtype=cp.float32)
```

```
2341            degraded = degraded + noise
2342
2343        # 4. Composite and Return
2344        final_image = cp.clip(degraded + clean_interior, 0, 255)
2345
2346        if USE_GPU:
2347            return cp.asnumpy(final_image).astype(np.uint8)
2348        return np.asarray(final_image).astype(np.uint8)
```

C-2: Weather Definition

```
1210 class WeatherManager:
1211     def __init__(self, world):
1212         self.world = world
1213         self.weather_presets = {
1214             # ===== CLEAR SKY WEATHER =====
1215             'Clear Sky - Sunrise': {
1216                 'cloudiness': 10.0,
1217                 'precipitation': 0.0,
1218                 'precipitation_deposits': 0.0,
1219                 'wind_intensity': 5.0,
1220                 'sun_azimuth_angle': 0.0,
1221                 'sun_altitude_angle': 15.0,  # Sunrise
1222                 'fog_density': 0.0,
1223                 'fog_distance': 0.0,
1224                 'wetness': 0.0,
1225                 'fog_falloff': 0.0,
1226                 'scattering_intensity': 0.0,
1227                 'mie_scattering_scale': 0.0,
1228                 'rayleigh_scattering_scale': 0.0331
1229             },
1230             'Clear Sky - Noon': {
1231                 'cloudiness': 10.0,
1232                 'precipitation': 0.0,
1233                 'precipitation_deposits': 0.0,
1234                 'wind_intensity': 5.0,
1235                 'sun_azimuth_angle': 0.0,
1236                 'sun_altitude_angle': 75.0,  # Noon - high sun
1237                 'fog_density': 0.0,
1238                 'fog_distance': 0.0,
1239                 'wetness': 0.0,
1240                 'fog_falloff': 0.0,
1241                 'scattering_intensity': 0.0,
1242                 'mie_scattering_scale': 0.0,
1243                 'rayleigh_scattering_scale': 0.0331
1244             },
1245             'Clear Sky - Night': {
1246                 'cloudiness': 10.0,
1247                 'precipitation': 0.0,
1248                 'precipitation_deposits': 0.0,
1249                 'wind_intensity': 5.0,
1250                 'sun_azimuth_angle': 0.0,
1251                 'sun_altitude_angle': -30.0,  # Night
1252                 'fog_density': 0.0,
1253                 'fog_distance': 0.0,
1254                 'wetness': 0.0,
1255                 'fog_falloff': 0.0,
1256                 'scattering_intensity': 0.0,
1257                 'mie_scattering_scale': 0.0,
1258                 'rayleigh_scattering_scale': 0.0331
1259             },
1260             # ===== HEAVY RAIN WEATHER =====
1261             'Heavy Rain - Sunrise': {
1262                 'cloudiness': 100.0,
1263                 'precipitation': 100.0,
1264                 'precipitation_deposits': 100.0,
1265                 'wind_intensity': 80.0,
1266                 'sun_azimuth_angle': 0.0,
1267                 'sun_altitude_angle': 15.0,  # Sunrise
1268                 'fog_density': 10.0,
1269                 'fog_distance': 20.0,
1270                 'wetness': 100.0,
1271                 'fog_falloff': 2.0,
```

```
1272                 'scattering_intensity': 1.0,
1273                 'mie_scattering_scale': 0.03,
1274                 'rayleigh_scattering_scale': 0.0331
1275             },
1276             'Heavy Rain - Noon': {
1277                 'cloudiness': 100.0,
1278                 'precipitation': 100.0,
1279                 'precipitation_deposits': 100.0,
1280                 'wind_intensity': 80.0,
1281                 'sun_azimuth_angle': 0.0,
1282                 'sun_altitude_angle': 45.0,  # Noon (darker due to clouds)
1283                 'fog_density': 10.0,
1284                 'fog_distance': 20.0,
1285                 'wetness': 100.0,
1286                 'fog_falloff': 2.0,
1287                 'scattering_intensity': 1.0,
1288                 'mie_scattering_scale': 0.03,
1289                 'rayleigh_scattering_scale': 0.0331
1290             },
1291             'Heavy Rain - Night': {
1292                 'cloudiness': 100.0,
1293                 'precipitation': 100.0,
1294                 'precipitation_deposits': 100.0,
1295                 'wind_intensity': 80.0,
1296                 'sun_azimuth_angle': 0.0,
1297                 'sun_altitude_angle': -30.0,  # Night
1298                 'fog_density': 10.0,
1299                 'fog_distance': 20.0,
1300                 'wetness': 100.0,
1301                 'fog_falloff': 2.0,
1302                 'scattering_intensity': 1.0,
1303                 'mie_scattering_scale': 0.03,
1304                 'rayleigh_scattering_scale': 0.0331
1305             },
1306         }
1307 
1308         self.weather_names = list(self.weather_presets.keys())
1309         self.current_weather_index = 0
1310 
1311     def apply_weather(self, weather_name):
1312         """Apply a weather preset"""
1313         if weather_name in self.weather_presets:
1314             weather = carla.WeatherParameters(**self.weather_presets[weather_name])
1315             self.world.set_weather(weather)
1316             print(f"🌦  Weather changed to: {weather_name}")
1317 
1318     def cycle_weather(self):
1319         """Cycle to next weather preset"""
1320         self.current_weather_index = (self.current_weather_index + 1) % len(self.weather_names)
1321         next_weather = self.weather_names[self.current_weather_index]
1322         self.apply_weather(next_weather)
1323 
1324     def get_current_weather_name(self):
1325         """Get current weather name"""
1326         return self.weather_names[self.current_weather_index]
1327 
1328     def get_current_weather_params(self):
1329         """Get current weather parameters"""
1330         weather_name = self.get_current_weather_name()
1331         return self.weather_presets.get(weather_name, {})
```

C-3: Road ID Tracing

```
2040 def trace_highway_road_ids(world, start_location, trace_distance=5000.0, step=5.0):
2041     """
2042     Trace waypoints forward from spawn point, collecting all road_ids.
2043     Covers the entire figure-8 highway loop in Town04 (~3-4km).
2044     """
2045     carla_map = world.get_map()
2046     wp = carla_map.get_waypoint(
2047         carla.Location(x=start_location[0], y=start_location[1], z=0.5),
2048         project_to_road=True,
```

```
2049            lane_type=carla.LaneType.Driving
2050        )
2051        if not wp:
2052            print("⚠️ Could not find starting waypoint for highway tracing")
2053            return set()
2054
2055        road_ids = set()
2056        road_ids.add(wp.road_id)
2057        traced = 0.0
2058
2059        while traced < trace_distance:
2060            next_wps = wp.next(step)
2061            if not next_wps:
2062                break
2063            wp = next_wps[0]
2064            road_ids.add(wp.road_id)
2065            traced += step
2066
2067        print(f"🛣️ Highway traced: {len(road_ids)} road IDs over {traced:.0f}m")
2068        return road_ids
```

C-4: Fixed Traffic

```
1079    def spawn_traffic_fixed(self, num_vehicles):
1080        """
1081        🆕 FIXED TRAFFIC SPAWNING (v5-4) - WAYPOINT-BASED
1082        Spawns vehicles ahead of ego using waypoints for 100% correct lane placement.
1083        """
1084        if num_vehicles <= 0:
1085            return
1086
1087        if not self.ego_vehicle:
1088            print("⚠️  No ego vehicle set, cannot spawn fixed traffic")
1089            return
1090
1091        carla_map = self.world.get_map()
1092        ego_loc = self.ego_vehicle.get_location()
1093        ego_wp = carla_map.get_waypoint(ego_loc, project_to_road=True)
1094
1095        if not ego_wp:
1096            print("⚠️  Could not find ego waypoint")
1097            return
1098
1099        print(f"📍 FIXED spawn mode: Waypoint-based (100% reproducible)")
1100        print(f"   Ego location: ({ego_loc.x:.1f}, {ego_loc.y:.1f}, {ego_loc.z:.1f})")
1101        print(f"   Ego lane: {ego_wp.lane_id}, Road: {ego_wp.road_id}")
1102
1103        vehicle_bps = self.blueprint_library.filter('vehicle.*')
1104        vehicle_bps = [x for x in vehicle_bps if int(x.get_attribute('number_of_wheels')) == 4]
1105        vehicle_bps = [x for x in vehicle_bps if not x.id.endswith('microlino')]
1106        vehicle_bps = [x for x in vehicle_bps if not x.id.endswith('carlacola')]
1107        vehicle_bps = [x for x in vehicle_bps if not x.id.endswith('cybertruck')]
1108        vehicle_bps = [x for x in vehicle_bps if not x.id.endswith('t2')]
1109        vehicle_bps = [x for x in vehicle_bps if not x.id.endswith('sprinter')]
1110
1111        FRONT_SPAWN_CONFIG = [
1112            (60.0, 1), (100.0, -1), (150.0, 0), (200.0, 1),
1113            (250.0, 0), (300.0, -1), (350.0, 1), (400.0, 0),
1114            (460.0, -1), (520.0, 0), (580.0, 1), (650.0, 0), (720.0, -1), (800.0, 1), (900.0, 0),
1115        ]
1116
1117        REAR_SPAWN_CONFIG = [
1118            (-50.0, 1), (-120.0, 0), (-200.0, -1), (-300.0, 1), (-400.0, 0),
1119        ]
1120
1121        num_front = min(int(num_vehicles * 0.75), len(FRONT_SPAWN_CONFIG))
1122        num_rear = min(num_vehicles - num_front, len(REAR_SPAWN_CONFIG))
1123
1124        spawned_front = 0
```

```
1125        spawned_rear = 0
1126
1127        def get_lane_waypoint(base_wp, lane_offset):
1128            target_wp = base_wp
1129            if lane_offset > 0:
1130                for _ in range(abs(lane_offset)):
1131                    right = target_wp.get_right_lane()
1132                    if right and right.lane_type == carla.LaneType.Driving:
1133                        target_wp = right
1134                    else:
1135                        return None
1136            elif lane_offset < 0:
1137                for _ in range(abs(lane_offset)):
1138                    left = target_wp.get_left_lane()
1139                    if left and left.lane_type == carla.LaneType.Driving:
1140                        target_wp = left
1141                    else:
1142                        return None
1143            return target_wp
1144
1145        for i, (distance, lane_offset) in enumerate(FRONT_SPAWN_CONFIG[:num_front]):
1146            ahead_wps = ego_wp.next(distance)
1147            if not ahead_wps:
1148                continue
1149            target_wp = get_lane_waypoint(ahead_wps[0], lane_offset)
1150            if not target_wp:
1151                continue
1152            spawn_transform = target_wp.transform
1153            spawn_transform.location.z += 0.5
1154            vehicle_bp = vehicle_bps[i % len(vehicle_bps)]
1155            if vehicle_bp.has_attribute('color'):
1156                colors = vehicle_bp.get_attribute('color').recommended_values
1157                vehicle_bp.set_attribute('color', colors[i % len(colors)])
1158            vehicle = self.world.try_spawn_actor(vehicle_bp, spawn_transform)
1159            if vehicle:
1160                vehicle.set_autopilot(True, self.tm_port)
1161                self.tm_backend.vehicle_percentage_speed_difference(vehicle, 10.0)
1162                self.tm_backend.update_vehicle_lights(vehicle, True)
1163                self.vehicles.append(vehicle)
1164                spawned_front += 1
1165
1166        for i, (distance, lane_offset) in enumerate(REAR_SPAWN_CONFIG[:num_rear]):
1167            behind_wps = ego_wp.previous(abs(distance))
1168            if not behind_wps:
1169                continue
1170            target_wp = get_lane_waypoint(behind_wps[0], lane_offset)
1171            if not target_wp:
1172                continue
1173            spawn_transform = target_wp.transform
1174            spawn_transform.location.z += 0.5
1175            vehicle_bp = vehicle_bps[(num_front + i) % len(vehicle_bps)]
1176            if vehicle_bp.has_attribute('color'):
1177                colors = vehicle_bp.get_attribute('color').recommended_values
1178                vehicle_bp.set_attribute('color', colors[(num_front + i) % len(colors)])
1179            vehicle = self.world.try_spawn_actor(vehicle_bp, spawn_transform)
1180            if vehicle:
1181                vehicle.set_autopilot(True, self.tm_port)
1182                self.tm_backend.vehicle_percentage_speed_difference(vehicle, -5.0)
1183                self.tm_backend.update_vehicle_lights(vehicle, True)
1184                self.vehicles.append(vehicle)
1185                spawned_rear += 1
1186
1187        total_count = len(self.vehicles)
1188        print(f"🚙 Spawned {spawned_front + spawned_rear} vehicles: {spawned_front} FRONT + 
{spawned_rear} REAR (Total: {total_count})")
```

C-5: Degradation Metrics Tracker

```
399 class DegradationMetrics:
400     def __init__(self):
401         self.distance_errors = []
402         self.dropout_count = 0
403         self.total_samples = 0
404         self.weather_stats = {}
405 
406         self.ttc_history = []          # Time To Collision tracking
407         self.min_ttc = float('inf')    # Minimum TTC recorded
408         self.headway_errors = []       # For variance calculation
409 
410         # Radar FPR/FNR tracking
411         self.radar_true_positives = 0   # Radar detected, ground truth confirms vehicle
412         self.radar_false_positives = 0  # Radar detected, but no actual vehicle
413         self.radar_true_negatives = 0   # Radar clear, ground truth confirms no vehicle
414         self.radar_false_negatives = 0
415         self.current_weather = "Unknown"
416         self.fatal_collisions = 0       # Respawn count (collision recovery + wrong-way)
417 
418     def _ensure_weather(self, weather_name):
419         """Initialize per-weather tracking with ALL metrics"""
420         if weather_name not in self.weather_stats:
421             self.weather_stats[weather_name] = {
422                 'errors': [],
423                 'dropouts': 0,
424                 'samples': 0,
425                 'headway_errors': [],
426                 'ttc_history': [],
427                 'min_ttc': float('inf'),
428                 'radar_tp': 0,
429                 'radar_fp': 0,
430                 'radar_tn': 0,
431                 'radar_fn': 0,
432                 'rtf_history': [],
433                 'fatal_collisions': 0,
434             }
435 
436     def update(self, gt_distance, measured_distance, radar_status, weather_name):
437         self.total_samples += 1
438         self.current_weather = weather_name
439         self._ensure_weather(weather_name)
440 
441         if radar_status in ["NO_TARGET", "FUSION_PERSIST"]:
442             self.dropout_count += 1
443             self.weather_stats[weather_name]['dropouts'] += 1
444 
445         # Distance error tracking
446         if 0.1 < gt_distance < 99.0 and 0.1 < measured_distance < 99.0:
447             error = abs(gt_distance - measured_distance)
448             self.distance_errors.append(error)
449             self.headway_errors.append(error)
450             self.weather_stats[weather_name]['errors'].append(error)
451             self.weather_stats[weather_name]['headway_errors'].append(error)
452 
453         self.weather_stats[weather_name]['samples'] += 1
454 
455     def update_ttc(self, distance, ego_speed_ms, relative_velocity=None):
456         """
457         Update Time To Collision metric
458         TTC = distance / closing_speed
459         """
460         closing_speed = relative_velocity if relative_velocity is not None else ego_speed_ms
461 
462         if closing_speed > 0.5:  # Only calculate if actually moving towards
463             ttc = distance / closing_speed
464             if ttc > 0 and ttc < 30.0:  # Reasonable TTC range
465                 self.ttc_history.append(ttc)
466                 if ttc < self.min_ttc:
```

```
467                     self.min_ttc = ttc
468                 # Per-weather TTC tracking
469                 if self.current_weather in self.weather_stats:
470                     self.weather_stats[self.current_weather]['ttc_history'].append(ttc)
471                     if ttc < self.weather_stats[self.current_weather]['min_ttc']:
472                         self.weather_stats[self.current_weather]['min_ttc'] = ttc
473 
474     def update_radar_accuracy(self, radar_detected, gt_distance, weather_name=None):
475         """
476         Track radar detection accuracy for FPR/FNR calculation
477 
478         Args:
479             radar_detected: bool - Did radar report a vehicle?
480             gt_distance: float - Ground truth distance
481             weather_name: str - Current weather condition
482         """
483         if weather_name:
484             self.current_weather = weather_name
485             self._ensure_weather(weather_name)
486 
487         actual_vehicle_present = 2.0 < gt_distance < 80.0
488 
489         if radar_detected and actual_vehicle_present:
490             self.radar_true_positives += 1
491             if self.current_weather in self.weather_stats:
492                 self.weather_stats[self.current_weather]['radar_tp'] += 1
493         elif radar_detected and not actual_vehicle_present:
494             self.radar_false_positives += 1
495             if self.current_weather in self.weather_stats:
496                 self.weather_stats[self.current_weather]['radar_fp'] += 1
497         elif not radar_detected and not actual_vehicle_present:
498             self.radar_true_negatives += 1
499             if self.current_weather in self.weather_stats:
500                  self.weather_stats[self.current_weather]['radar_tn'] += 1
501         elif not radar_detected and actual_vehicle_present:
502             self.radar_false_negatives += 1
503             if self.current_weather in self.weather_stats:
504                 self.weather_stats[self.current_weather]['radar_fn'] += 1
505 
506     def get_stats(self):
507         if not self.distance_errors:
508             return {}
509 
510         # Calculate headway error variance
511         headway_variance = 0.0
512         if len(self.headway_errors) > 1:
513             headway_variance = np.var(self.headway_errors)
514 
515         # Calculate FPR and FNR
516         total_negatives = self.radar_true_negatives + self.radar_false_positives
517         total_positives = self.radar_true_positives + self.radar_false_negatives
518 
519         fpr = (self.radar_false_positives / total_negatives * 100) if total_negatives > 0 else
0.0
520         fnr = (self.radar_false_negatives / total_positives * 100) if total_positives > 0 else
0.0
521 
522         return {
523             'mean_error': np.mean(self.distance_errors) if len(self.distance_errors) > 0 else 0,
524             'max_error': np.max(self.distance_errors) if len(self.distance_errors) > 0 else 0,
525             'dropout_rate': self.dropout_count / self.total_samples if self.total_samples > 0
else 0,
526             'samples': self.total_samples,
527             'min_ttc': self.min_ttc if self.min_ttc != float('inf') else 0.0,
528             'headway_variance': headway_variance,
529             'radar_fpr': fpr,
530             'radar_fnr': fnr,
531             'fatal_collisions': self.fatal_collisions
532         }
533 
534     def update_rtf(self, rtf, weather_name):
```

```
535         """Track RTF per weather condition"""
536         self._ensure_weather(weather_name)
537         self.weather_stats[weather_name]['rtf_history'].append(rtf)
538
539     def update_fatal_collision(self, weather_name):
540         """Track vehicle respawn as a fatal collision"""
541         self.fatal_collisions += 1
542         self._ensure_weather(weather_name)
543         self.weather_stats[weather_name]['fatal_collisions'] += 1
```

C-6: Vehicle State Tracker Initialization

```
677 class VehicleStateTracker:  # Vehicle State Tracker
678     def __init__(self):
679         self.target_speed = 90.0
680         self.emergency_brake = False
681         self.manual_mode = False
682
683         # TIME-BASED DETECTION TRACKING (FPS-Independent)
684         self.time_since_detection = 0.0            # seconds (was frames_since_detection)
685         self.fusion_detection_duration = 0.0       # seconds (was
consecutive_fusion_detections)
686
687         # DISTANCE FILTERING (Prevents sensor thrashing)
688         self.smoothed_distance = 100.0             # Filtered distance value
689         self.last_valid_distance = 100.0           # Last confirmed reading
690         self.distance_history = []                 # Rolling buffer (time-stamped)
691         self.max_distance_change_rate = TimingConfig.MAX_DISTANCE_CHANGE_RATE  # m/sec
692
693         # ADAS features toggle
694         self.traffic_light_assist_on = True
695         self.active_red_light_latch = False
696         self.lane_assist_on = True
697         self.speed_limiter_on = True
698         self.acc_on = True
699         self.holding_at_line = False
700         self.was_stopping_for_light = False
701         self.was_acc_following = False
702         self.time_since_light_stop = 999.0  # Timer for post-stop resume protocol
703         self.ticks_since_acc = 999  # Tick counter: 0 when ACC/EMERGENCY active, increments in
CRUISE
704
705         # Manual driving control - reverse functionality
706         self.brake_key_press_time = 0.0
707         self.brake_key_pressed = False
708         self.reverse_engaged = False  # Track if reverse is currently engaged
709
710         # Input method tracking
711         self.using_wheel = False  # Track if wheel is being used
712
713         # Camera toggle states (for performance optimization)
714         self.show_depth_camera = True      # Toggle with T key
715         self.show_semantic_camera = True   # Toggle with Y key
716         self.show_topview_camera = True    # Toggle with U key
717
718         # ENHANCED LIGHTING SYSTEM WITH VISIBILITY SENSOR
719
720         # Main headlights (low/high beam)
721         self.headlights_on = False
722         self.manual_lights_mode = False      # Direct manual control (L key)
723         self.manual_lights_state = False     # User-set light state when manual
724         self.high_beam_mode = False          # False = low beam, True = high beam
725
726         # Visibility sensor data
727         self.current_visibility = 1000.0     # meters
728         self.low_visibility_threshold = 200.0  # meters
729
730         # Speed PID for cruise control (when no vehicle ahead)
731         self.speed_pid = PIDController(kp=0.4, ki=0.0132, kd=0.225, output_limits=(-1.0, 1.0))
#Tuned for dt=0.05s @ 20 FPS
732
733         # Distance PID for following control (when vehicle ahead)
734         self.distance_pid = PIDController(kp=0.2, ki=0.005, kd=1.2, output_limits=(-1.0, 1.0))
```

```
735         # NEW: STOP LINE LOCKING VARIABLES
736         self.stop_line_locked = False       # True when we have committed to a specific line
737         self.stop_line_true_dist = 0.0      # The absolute distance we are trying to close
738 
739         # COLLISION RECOVERY & WRONG-WAY DETECTION (Always Enabled) - Covers: Guardrails,
barriers, walls, AND vehicle-to-vehicle
740         self.collision_recovery_active = False
741         self.collision_recovery_phase = "NONE"   # NONE -> BRAKE -> REVERSE -> REALIGN -> DONE
742         self.collision_recovery_timer = 0.0
743         self.collision_recovery_count = 0         # Tracks ALL collisions (guardrails + vehicles)
744         self.last_recovery_collision_index = 0    # Index into collision_history
745         self.collision_recovery_cooldown = 0.0    # Seconds to ignore collisions after recovery
746         self.post_respawn_warmup = 0.0            # Seconds to hold brakes after teleport (sensor
stabilization)
747         self.post_respawn_grace = 0.0             # Seconds to exempt from stuck timer after any
respawn
748         self.wrong_way_detected = False
749         self.wrong_way_correction_active = False
```

Bibliography

Ali Alheeti, K. M., & Mc Donald-Maier, K. (2018). Intelligent intrusion detection in external communication systems for autonomous vehicles. *Systems Science and Control Engineering*, *6*(1), 48–56. https://doi.org/10.1080/21642583.2018.1440260

Anand, S., & Ohol, S. S. (2023). Modelling and simulation of adaptive cruise control and overtake assist system. *Materials Today: Proceedings*, *72*, 1353–1360. https://doi.org/10.1016/j.matpr.2022.09.330

Autocrypt. (2023). *The State of Level 3 Autonomous Driving in 2023 | AUTOCRYPT*. https://autocrypt.io/the-state-of-level-3-autonomous-driving-in-2023/

Bijelic, M., Gruber, T., Mannan, F., Kraus, F., Ritter, W., Dietmayer, K., & Heide, F. (2020). Seeing Through Fog Without Seeing Fog: Deep Multimodal Sensor Fusion in Unseen Adverse Weather. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 11679–11689. http://arxiv.org/abs/1902.08913

Borase, R. P., Maghade, D. K., Sondkar, S. Y., & Pawar, S. N. (2020). A review of PID control, tuning methods and applications. *International Journal of Dynamics and Control 2020 9:2*, *9*(2), 818–827. https://doi.org/10.1007/s40435-020-00665-4

CARLA Documentation. (2024). *CARLA Simulator*. https://carla.readthedocs.io/en/latest/

CARLA Vehicle Catalogue. (2024). *Catalogue vehicles - CARLA Simulator*. https://carla.readthedocs.io/en/latest/catalogue_vehicles/#mini-cooper-s

Coulter, & R.C. (1992). *Implementation of the Pure Pursuit Path 'hcking Algorithm*.

Dai, D., & Gool, L. Van. (2018). Dark Model Adaptation: Semantic Image Segmentation from Daytime to Nighttime. *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, *2018-November*, 3819–3824. https://doi.org/10.1109/ITSC.2018.8569387

Idrees Shaikh. (2024). *Autonomous Driving in Carla using Deep Reinforcement Learning · GitHub*. https://github.com/idreesshaikh/Autonomous-Driving-in-Carla-using-Deep-Reinforcement-Learning/blob/main/continuous_driver.py

Jaimin K. (2021). *Longitudinal & Lateral Control for Autonomous Vehicles — CARLA Simulator | Medium*. https://medium.com/@jaimin-k/longitudinal-lateral-control-for-autonomous-vehicles-carla-simulator-c045918816bd

Kenk, M. A., & Hassaballah, M. (2020). *DAWN: Vehicle Detection in Adverse Weather Nature Dataset*. https://doi.org/10.17632/766ygrbt8y.3

Kim, J., Park, B. J., & Kim, J. (2023). Empirical Analysis of Autonomous Vehicle's LiDAR Detection Performance Degradation for Actual Road Driving in Rain and Fog. *Sensors*, *23*(6). https://doi.org/10.3390/s23062972

Kumar, D., & Muhammad, N. (2023). Object Detection in Adverse Weather for Autonomous Driving through Data Merging and YOLOv8. *Sensors (Basel, Switzerland)*, *23*(20). https://doi.org/10.3390/s23208471

Li, D., & Okhrin, O. (2023). Modified DDPG car-following model with a real-world human driving experience with CARLA simulator. *Transportation Research Part C: Emerging Technologies*, *147*, 103987. https://doi.org/10.1016/j.trc.2022.103987

Li, P., Kusari, A., & LeBlanc, D. J. (2021). *A Novel Traffic Simulation Framework for Testing Autonomous Vehicles Using SUMO and CARLA*. http://arxiv.org/abs/2110.07111

Luca Venturi, & Krishtof Korda. (2020). *Hands-On Vision and Behavior for Self-Driving Cars | Data | Paperback*. https://www.packtpub.com/en-pt/product/hands-on-vision-and-behavior-for-self-driving-cars-9781800203587

Maciek Dziubinski. (2019a). *Introduction to the CARLA simulator: training a neural network to control a car (Part 1) | by Maciek Dziubiński | Acta Schola Automata Polonica | Medium*. https://medium.com/asap-report/introduction-to-the-carla-simulator-training-a-neural-network-to-control-a-car-part-1-e1c2c9a056a5

Maciek Dziubinski. (2019b). *Introduction to the CARLA simulator: training a neural network to control a car (Part 2) | by Maciek Dziubiński | Acta Schola Automata Polonica | Medium*. https://medium.com/asap-report/introduction-to-the-carla-simulator-training-a-neural-network-to-control-a-car-part-2-6a71115f2940

Maddern, W., Pascoe, G., Gadd, M., Barnes, D., Yeomans, B., & Newman, P. (2020). *Real-time Kinematic Ground Truth for the Oxford RobotCar Dataset*. http://arxiv.org/abs/2002.10152

MathWorks. (2024). *Adaptive Cruise Control with Sensor Fusion - MATLAB & Simulink*. https://www.mathworks.com/help/driving/ug/adaptive-cruise-control-with-sensor-fusion.html

Mercedes-Benz Group. (2022). *Mercedes-Benz – the front runner in automated driving and safety technologies | Mercedes-Benz Group > Technology > Autonomous Driving > Driving*. https://group.mercedes-benz.com/technology/autonomous-driving/driving/drive-pilot.html

Michael Wu. (2020). *PID Control Integration into ROAR Autonomous System in Carla | Medium*. https://medium.com/@wuxiaohua1011/pid-control-integration-into-roar-autonomous-system-in-carla-539724f98f1a

Olsson Kth, M., Tekniska, K., Skolan, H., Datavetenskap, F., & Kommunikation, O. (2016). *Behavior Trees for decision-making in Autonomous Driving*.

Pérez-Gil, Ó., Barea, R., López-Guillén, E., Bergasa, L. M., Gómez-Huélamo, C., Gutiérrez, R., & Díaz-Díaz, A. (2022). Deep reinforcement learning based control for Autonomous Vehicles in CARLA. *Multimedia Tools and Applications 2021 81:3*, *81*(3), 3553–3576. https://doi.org/10.1007/s11042-021-11437-3

Royo, S., & Ballesta-Garcia, M. (2019). An Overview of Lidar Imaging Systems for Autonomous Vehicles. *Applied Sciences 2019, Vol. 9, 9*(19). https://doi.org/10.3390/app9194093

SAE International. (2021). *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*. https://doi.org/10.4271/J3016_202104

Shrivastava, S., Somthankar, A., Pandya, V., & Patil, M. (2023). Implementation of a PID Controller for Autonomous Vehicles with Traffic Light Detection in CARLA. *Lecture Notes in Networks and Systems*, *632*, 1–13. https://doi.org/10.1007/978-981-99-0071-8_1

Sun, L., Wang, K., Yang, K., & Xiang, K. (2019). *See Clearer at Night: Towards Robust Nighttime Semantic Segmentation through Day-Night Image Conversion*. 8. http://arxiv.org/abs/1908.05868

Vargas, J., Alsweiss, S., Toker, O., Razdan, R., & Santos, J. (2021). An overview of autonomous vehicles sensors and their vulnerability to weather conditions. *Sensors*, *21*(16). https://doi.org/10.3390/s21165397

Vector. (2024). *ADAS Testing with Virtual Test Drives | Vector*. https://www.vector.com/int/en/products/products-a-z/software/dyna4/adas-testing-with-virtual-test-drives/

Wang, X., Ye, C., Quddus, M., & Morris, A. (2023). Pedestrian safety in an automated driving environment: Calibrating and evaluating the responsibility-sensitive safety model. *Accident Analysis & Prevention*, *192*, 107265. https://doi.org/10.1016/j.aap.2023.107265

Yan, X., Feng, S., Leblanc, D. J., Flannagan, C., Liu, H. X., & Svenson, A. L. (2023). *SAFETY METRICS ASSESSMENT USING LOGGED VEHICLE TRAJECTORY DATA*.

Yeong, D. J., Velasco-hernandez, G., Barry, J., & Walsh, J. (2021). Sensor and Sensor Fusion Technology in Autonomous Vehicles: A Review. *Sensors 2021, Vol. 21, 21*(6), 1–37. https://doi.org/10.3390/s21062140

Yoneda, K., Suganuma, N., Yanase, R., & Aldibaja, M. (2019). Automated driving recognition technologies for adverse weather conditions. *IATSS Research*, *43*(4), 253–262. https://doi.org/10.1016/j.iatssr.2019.11.005

Yu, L., & Wang, R. (2022). Researches on Adaptive Cruise Control system: A state of the art review. *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, *236*(2–3), 211–240. https://doi.org/10.1177/09544070211019254

Zhou, J., Schmied, R., Sandalek, A., Kokal, H., & del Re, L. (2016). A Framework for Virtual Testing of ADAS. *SAE International Journal of Passenger Cars - Electronic and Electrical Systems*, *9*(1), 66–73. https://doi.org/10.4271/2016-01-0049