The College of Management Academic Studies
Faculty of Computer Science

Canonicar

Etai Zilberman Kfir Hoftman Roee Zach Bar Ara Roey Srugo

Approved by the supervisor: Dr. Shay Horovitz

September 2025

Declaration

We, Etai Zilberman, Kfir Hoftman, Roee Zach, Bar Ara, and Roey Srugo, of the Department of Computer Science, The College of Management Academic Studies, confirm that this is our own work and figures, tables, equations, code snippets, artworks, and illustrations in this report are original and have not been taken from any other person's work, except where the works of others have been explicitly acknowledged, quoted, and referenced. We understand that failing to do so will be considered a case of plagiarism. Plagiarism is a form of academic misconduct and will be penalised accordingly.

We give consent to a copy of our report being shared with future students as an exemple.

We give consent for our work to be made available more widely to public with interest in teaching, learning and research.

Etai Zilberman, Kfir Hoftman, Roee Zach, Bar Ara, Roey Srugo
Friday 5th September, 2025

Abstract

Canonicar is an autonomous racing project in a simulation environment that trains a Proximal Policy Optimization (PPO) agent in the CARLA simulator to maintain stable control on custom-made racing tracks. The system is built as a reproducible, Dockerized stack that couples CARLA with ROS 2 nodes for real-time sensing (semantic-segmentation camera, LiDAR, IMU), a PPO policy for continuous control, and a vehicle-control node that closes the loop in simulation. To enable observability and comparison across runs, we log metrics and artifacts (e.g., rewards, action distributions, episode lengths, lap-time proxies, checkpoints) to Tensorboard and visualize them in a lightweight dashboard.

Our proof-of-concept demonstrates end-to-end data flow from sensors through a learning policy to actuation in CARLA, with the core components already implemented: simulator and random-track pipeline, ROS 2 data wiring, a baseline PPO agent, data logging, and a basic frontend. Ongoing work focuses on refining reward design, and applying curriculum learning to accelerate policy convergence and robustness. The contribution is a clean, modular framework for closed-loop autonomous racing research in simulation: easy to run, instrumented for analysis, and designed to scale from initial baselines to competitive lap-time performance on unfamiliar tracks.

Keywords: Autonomous Driving, PPO, Reinforcement Learning, CARLA, ROS 2

Acknowledgements

We would like to express our deepest gratitude to our supervisor, **Dr. Shay Horovitz**, for his continuous guidance, insightful feedback, and encouragement throughout the course of this project. His expertise and support were invaluable in helping us navigate challenges and push the project forward.

We are also grateful to the **College of Management Academic Studies (COLMAN)** for providing us with the opportunity, resources, and environment to pursue this challenging and rewarding work.

Finally, we would like to thank our families and friends for their patience, encouragement, and unwavering support during long hours of development and research. Completing this project was made even more demanding by the difficult reality and the uncertainty we are living through in Israel these days. Their strength and understanding enabled us to stay committed and focused, even in such challenging times.

Executive Summary

Canonicar is an advanced autonomous racing simulation project that leverages artificial intelligence to develop optimal racing strategies within the CARLA simulator environment. The project employs Proximal Policy Optimization (PPO), a state-of-the-art reinforcement learning algorithm, to train intelligent agents capable of navigating complex racing circuits with precision and efficiency.

The system architecture integrates multiple cutting-edge technologies including sensor fusion from camera, LiDAR, and IMU sensors to provide comprehensive environmental awareness. The AI agent processes this multi-modal sensory data through a sophisticated neural network that learns optimal steering and throttle control actions through continuous interaction with the simulated racing environment.

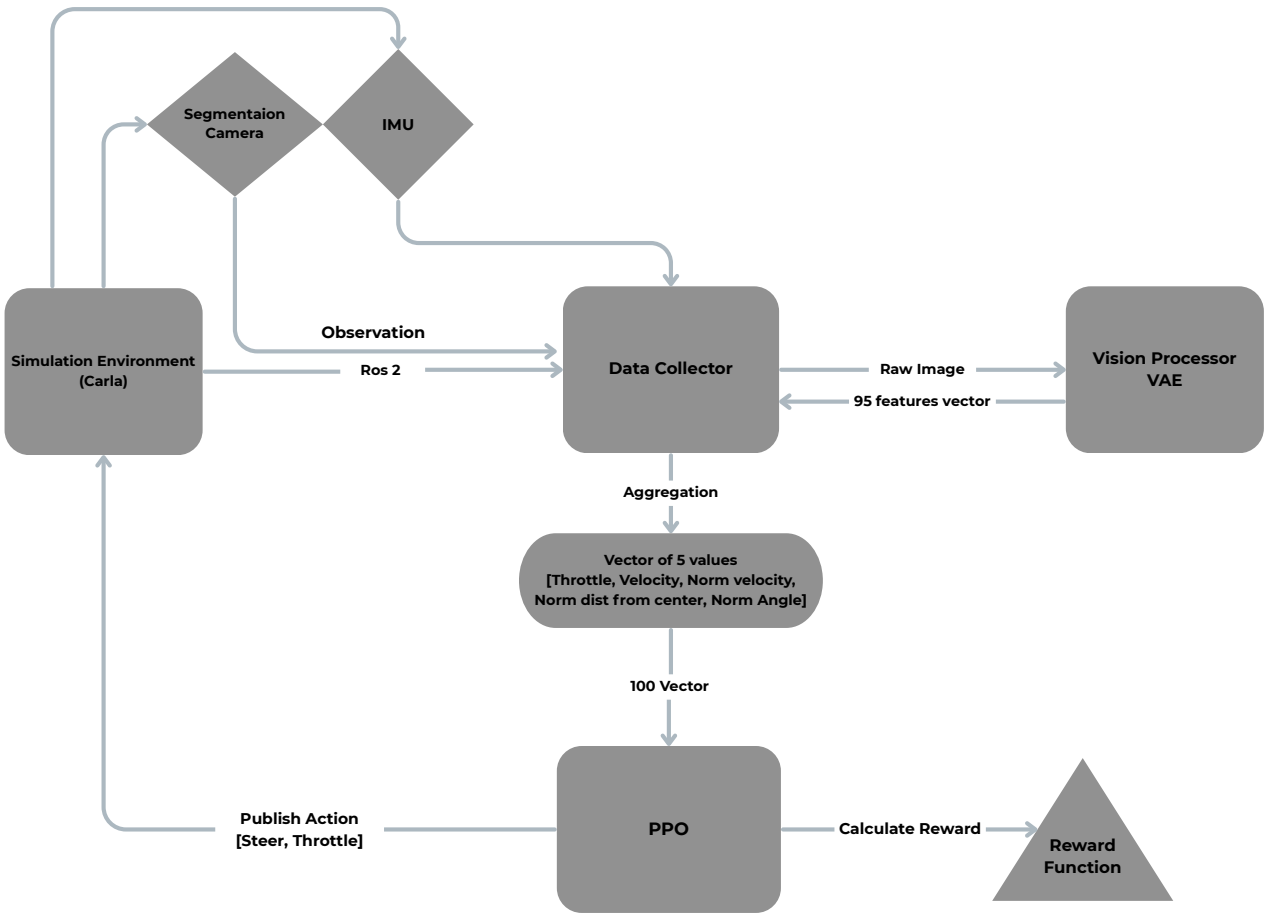
The project utilizes Docker containerization and ROS 2 middleware for scalable deployment and real-time communication between system components. The training pipeline incorporates techniques such as Monte Carlo estimates of returns for stable policy updates and custom reward functions that balance racing performance metrics including lap completion time, track adherence, and collision avoidance.

Key technical achievements include the development of a custom track generator, implementation of dynamic map loading capabilities, and integration with TensorBoard for comprehensive training visualization. The system supports both training and evaluation modes, with extensive logging capabilities for performance analysis and model checkpointing for continuous improvement.

The project demonstrates practical applications in autonomous vehicle research, racing optimization, and reinforcement learning algorithm development. Through iterative version improvements, the system has evolved to handle discrete action spaces, implement sophisticated reward mechanisms, and achieve stable training convergence across extended training sessions.

This comprehensive platform serves as both a research tool for advancing autonomous driving technologies and a demonstration of reinforcement learning capabilities in high-performance racing scenarios, contributing valuable insights to the broader autonomous vehicle development community.

Project Building Blocks Diagram



Contents

List of Figures

List of Tables

List of Abbreviations

RL	Reinforcement Learning
CARLA	Car Learning to Act (Open-source autonomous driving simulator)
PPO	Proximal Policy Optimization
ROS2	Robot Operating System 2
LiDAR	Light Detection and Ranging
IMU	Inertial Measurement Unit
GNSS	Global Navigation Satellite System
RGB	Red Green Blue (image sensor format)
VAE	Variational Autoencoder
SS	Semantic Segmentation
DB	Database
TB	TensorBoard
UI	User Interface
API	Application Programming Interface
ML	Machine Learning
NN	Neural Network
CPU	Central Processing Unit
GPU	Graphics Processing Unit
Docker	Containerization Platform for Applications
JIRA	Project Management and Issue Tracking Tool
GitHub	Version Control and Collaboration Platform
ETC	Estimated Time of Completion
POV	Point of View
RLHF	Reinforcement Learning with Human Feedback
GAE	Generalized Advantage Estimation

Chapter 1

Introduction

1.1 Background

The development of autonomous vehicle technologies has become a critical focus in modern transportation research. Racing simulations provide an ideal testing environment for autonomous driving algorithms due to their controlled yet challenging nature, requiring precise control and real-time decision making. The CARLA simulator has emerged as a leading platform for autonomous vehicle research, offering high-fidelity urban driving environments that closely replicate real-world conditions.

1.2 Problem statement

Traditional autonomous vehicle training requires extensive real-world testing, which is costly, time-consuming, and potentially dangerous. Current racing simulation approaches often lack the integration of multiple sensor modalities and sophisticated reinforcement learning algorithms needed for optimal performance. There is a need for a comprehensive autonomous racing system that can learn optimal racing strategies through reinforcement learning while incorporating multi-modal sensor fusion for enhanced environmental perception.

1.3 Objectives

The primary objectives of this project are to:
- Develop an intelligent autonomous racing agent using the Proximal Policy Optimization (PPO) Reinforcement Learning algorithm.
- Implement multi-modal sensor fusion combining Segmentation camera, LiDAR, and IMU data.
- Create a scalable training pipeline with real-time performance monitoring.
- Achieve optimal racing line on track while maintaining track adherence and collision avoidance.
- Establish a comprehensive evaluation framework for autonomous racing performance.

1.4 Scope and Limitations

This project focuses on autonomous racing within the CARLA simulator environment using custom-generated race tracks. The scope includes the development of PPO-based learning algorithms, sensor data processing, and performance optimization, as well as track generation

using Unreal Engine for custom circuits. Limitations include restriction to simulated environments, dependency on CARLA simulator capabilities, and computational requirements for neural network training and simulator running.

1.5 Methodology

The project employs a reinforcement learning approach using Proximal Policy Optimization (PPO) algorithm. The methodology includes sensor data collection from multiple sources, neural network-based policy learning, reward function optimization, and iterative training with performance evaluation. The system utilizes ROS 2 for inter-component communication and Docker for deployment scalability.

1.6 Organization of the Project Book

This project book is organized into seven main sections:
- Introduction - providing project context
- Literature Review - examining relevant research
- System Design - detailing implementation architecture
- Results - presenting experimental findings
- Conclusion - summarizing achievements and future directions
- References - citing relevant sources

Chapter 2

Literature Review

2.1 Overview of Relevant Literature

Autonomous driving has become a major research frontier in recent years, combining advances in reinforcement learning, computer vision, robotics, and high-fidelity simulation. One of the key challenges in this domain is enabling vehicles to operate robustly in highly dynamic and uncertain environments while maintaining safety, efficiency, and adaptability. Traditional rule-based and classical control approaches, although effective in structured scenarios, have shown limitations in handling the variability of real-world driving conditions. As a result, learning-based methods, particularly reinforcement learning (RL), have gained considerable attention. Proximal Policy Optimization (PPO) has emerged as one of the most widely adopted algorithms in continuous control due to its balance between performance and stability [1]. Unlike earlier policy gradient methods that were prone to instability, PPO introduces a clipped surrogate objective that constrains policy updates, preventing destructive gradient steps while maintaining sample efficiency. Numerous studies have shown PPO's effectiveness in controlling high-dimensional systems, including autonomous vehicles, robotic arms, and humanoid locomotion. In autonomous driving contexts, PPO has been applied to both urban navigation and racing environments [2, 3], demonstrating the ability to learn complex maneuvers such as overtaking, sharp turns, and collision avoidance.

Another important area of research concerns the integration of multiple sensor modalities. Multi-modal sensor fusion has consistently been shown to improve perception and decision-making by leveraging complementary strengths of different sensors [4, 5]. For instance, vision-based methods using RGB or semantic segmentation cameras provide detailed appearance and lane information, while LiDAR contributes geometric and depth information, and inertial measurement units (IMUs) capture dynamics such as acceleration and angular velocity. Studies have demonstrated that fusing these modalities produces more robust state representations, especially in environments where individual sensors may fail due to occlusions, noise, or adverse weather conditions.

The CARLA simulator [6] has established itself as a de facto standard for autonomous driving research, offering a realistic, configurable environment with support for various weather conditions, traffic participants, and sensor modalities. CARLA has been extensively used for training and benchmarking reinforcement learning agents, enabling reproducible research while avoiding the risks of real-world testing. Several competitions, such as the CARLA Autonomous Driving Challenge, have further solidified its role in pushing forward the state of the art in end-to-end driving and policy learning. Prior work has demonstrated the use of CARLA not only for urban driving but also for racing-style environments [7, 8], where the objective is to optimize lap time while minimizing collisions and off-track behavior.

Within the racing context, reward function design has been highlighted as a critical factor [9, 10]. Researchers have proposed reward functions that encourage track adherence, smooth steering, velocity optimization, and penalization of unsafe maneuvers. For example, some approaches incorporate potential-based shaping to balance progress along the track with safety constraints, while others explicitly penalize throttle–brake overlap to encourage energy-efficient driving. Curriculum learning strategies have also been investigated, where agents are progressively trained on simpler tasks (e.g., staying on track) before advancing to more complex objectives such as high-speed cornering and overtaking [11].

Recent works have also begun exploring representation learning in autonomous driving. Variational Autoencoders (VAEs) and contrastive learning methods have been employed to compress high-dimensional visual inputs into compact latent spaces that are more amenable to policy optimization [12, 13]. This reduces computational burden and improves training stability, especially when combined with PPO or other policy gradient algorithms. Other studies have leveraged transfer learning from large-scale pre-trained vision models (e.g., ImageNet-based backbones) to accelerate convergence and enhance generalization across diverse tracks [14, 15].

In summary, the literature demonstrates that (i) PPO remains a strong baseline for reinforcement learning in continuous control driving tasks, (ii) sensor fusion is essential for reliable perception in dynamic environments, (iii) CARLA has become the main testbed for reproducible research in both urban and racing contexts, and (iv) reward shaping and curriculum design are crucial for achieving competitive performance. Building on these insights, our project incorporates PPO, advanced state estimation through multi-sensor fusion, and procedurally generated racing tracks in CARLA to explore generalization to unseen driving scenarios.

Chapter 3

System Design and Implementation

3.1 System Architecture

The system is designed as a modular autonomous racing simulation stack built around the CARLA simulator. It is orchestrated using ROS 2 nodes, reinforcement learning (PPO), data services, and a monitoring user interface. The architecture is divided into several high-level components, as described below.

3.1.1 Simulation and World Interface

The CARLA simulator provides the virtual environment, including tracks and physical elements. The vehicle is spawned on a chosen map in a specific spawn point. The simulator can be configured in the Unreal Engine game engine and launched externally, enabling flexibility in experimentation and testing.

3.1.2 ROS 2 Orchestration and Launch

A central launch description coordinates all core system components. This ensures consistent initialization of the simulator connection, track configuration, sensor setup, and learning nodes.

3.1.3 Core Runtime Nodes

The runtime system is composed of several ROS 2 nodes, each with a distinct role:

- **Simulation Coordinator:** Manages the simulation lifecycle and synchronizes interactions between nodes.
- **Map Loader:** Loads track artifacts and provides map data to other components.
- **Client Node:** Spawns and configures the vehicle with the selected sensor setup.
- **Data Collector:** Captures and stores sensor data and episode information for training and evaluation, passes the data to the PPO node.
- **PPO node:** Receives data from the data collector, performs training and inference, and computes progress metrics. Sends the action vector to the vehicle control node.
- **Vehicle Control Node:** Executes actuation commands such as steering, throttle, and braking.

- **Sensor Data Module:** Provides standardized sensor data interfaces shared across nodes.
- **ROS Interfaces:** Custom service definitions enabling structured communication between nodes.

3.1.4 Learning and Model Management

The reinforcement learning workflow is managed through a checkpointing and versioning mechanism, allowing reproducibility and controlled evolution of model architectures. A dedicated performance tracking schema records metrics such as lap times, collisions, and deviations from the racing line.

3.1.5 User Interface and Monitoring

The system includes a user-facing interface for monitoring and analysis. A Streamlit-based dashboard presents system overviews and performance results, while TensorBoard enables visualization of training progress and learning curves in real time.

3.1.6 Deployment and Environment

The system supports containerized deployment for reproducibility and portability across computing environments. All major components, including the simulator bridge and learning nodes, can be launched within containers, ensuring consistent behavior across different hosts.

3.2 Data Collection and Preprocessing

Data in the system is generated continuously from the Carla simulation and ingested through a dedicated collection node. This node subscribes to multiple sensor modalities, including semantic camera imagery, and inertial measurements (IMU). A time synchronization mechanism ensures that temporally coherent triplets (image, IMU) are fused, reducing mismatches across modalities.

Acquisition Layer

- Semantic segmentation images are used for highlighting relevant classes such as road surface, boundaries, and obstacles. A pretrained VAE model takes the vector and extract features.
- IMU streams provide linear acceleration and angular velocity, (x,y,z) for each, mainly for scalar calculations as part of the input vector.

Data Integrity and Validation

- Data is published only once the vehicle and all sensors are confirmed active.
- Corrupted samples (e.g., NaN values) are filtered to avoid propagation into training.
- Shape assertions on intermediate tensors ensure consistency in data dimensionality.
- Conditional downsampling (e.g., publishing only every Nth sample) balances information density with computational efficiency.

Fusion and State Construction

The preprocessing pipeline fuses multi-modal inputs into a fixed-length state vector:

- **Vision features:** derived from semantic segmentation.
- **IMU features:** normalized kinematic signals for input vector values.

Over time, the input schema has evolved. For example, in order to reduce dimensionality for easier training, a satellite positioning modality (GNSS) was removed, requiring coordinated updates across policy and value networks, as well as LiDAR. A version log records such changes to maintain reproducibility.

Reliability and Integrity Measures

- Timestamps and readiness tracking prevent silent sensor dropouts.
- Explicit dimensional checks ensure consistency when feature definitions evolve.
- Controlled publication cadence prevents long-run numerical overflow.
- Device allocation between CPU and GPU adapts automatically depending on workload.

Challenges and Considerations

- **Modality alignment:** tolerant synchronization avoids excessive sample rejection due to small timing mismatches.
- **Evolving feature schema:** changes in input dimensionality required careful version management and checkpoint compatibility.
- **Throughput vs. fidelity:** high-resolution semantic imagery increases storage and compute demands; buffering strategies mitigated this trade-off.
- **Numerical stability:** normalization schemes required safeguards against division by zero and clipping bounds to handle rare outliers.
- **Data consistency:** strict shape contracts surfaced errors early, avoiding silent corruption during long training runs.

Outcome

The data collection and preprocessing pipeline delivers a stable, fused, and normalized state representation with strong structural guarantees. It supports reproducible reinforcement learning while remaining adaptable to architectural iterations and future dataset enrichment.

3.3 Implementation Details

3.3.1 Technology Stack

- **Language**: Python.
- **Middleware**: ROS 2 providing publish/subscribe topics, services, timed actions, and launch orchestration.

- **Simulation**: CARLA with vehicle dynamics, semantic camera, LiDAR, and IMU.
- **Learning Framework**: PyTorch supporting actor–critic networks, tensor operations, and checkpointing.
- **Monitoring**: TensorBoard for training metrics and Streamlit for the web dashboard.
- **Persistence**: Document–style episodic performance records including lap timing, collisions, deviation, and progress.
- **Containerization**: Docker / Compose enabling isolated and reproducible environments.
- **System Telemetry**: psutil reporting CPU and RAM usage and optionally GPU utilization.

3.3.2 Hardware / Runtime Requirements

- Multi-core x86 64 CPU with 4 physical cores.
- CUDA-capable GPU, optional, used to accelerate training with automatic CPU fallback.
- Memory capacity of 16GB covering simulation, learning, and logging.
- Persistent storage allocated for versioned checkpoints and run artifacts.
- Simulated sensors include semantic segmentation camera, LiDAR, and IMU with accelerometer and gyroscope.

3.3.3 Modular Components

Environment Coordination: Manages lifecycle flags such as vehicle readiness, map loading, and state transitions.

Map Loading: Imports track geometry and serves waypoint queries.

Vehicle Spawn Sensors: Instantiates the ego vehicle, the primary vehicle under observation or control, with a declarative sensor configuration.

Control Layer: Applies steering, throttle, and brake actions with safety gating.

Data Fusion / Collection: Subscribes to raw sensor streams, performs temporal alignment, feature construction, normalization, and optional archival.

Reinforcement Learning: PPO rollouts, advantage estimation, policy and value updates, checkpoint lifecycle, and geometric progress estimation.

Performance Logging: Episodic statistics and system telemetry emission.

3.3.4 Core Algorithms and Techniques

Sensor Fusion and State Assembly Semantic image features is combined with normalized IMU channels into a fixed-length state vector. Publication throttling (every Nth fused sample) balances temporal resolution and computational load. Online normalization maintains running mean and variance without storing history.

Reinforcement Learning (PPO) Proximal Policy Optimization (PPO) is a reinforcement learning algorithm designed to train agents in continuous or discrete action spaces in a stable and efficient way. It belongs to the family of policy gradient methods, which directly optimize the parameters of a policy network. Unlike naive policy gradient algorithms that can suffer from instability due to overly large parameter updates, PPO introduces a *clipped surrogate objective* that ensures each update stays within a safe trust region. This makes training more robust without requiring complex second-order optimization.
In practice, PPO alternates between two phases:

- **Trajectory Collection:** The agent interacts with the environment, generating batches of experiences $\{(s_t, a_t, r_t, \log \pi(a_t|s_t), V(s_t))\}$.
- **Policy Update:** Using these batches, the policy is updated to maximize expected rewards while constraining policy shifts between old and new policies.

Additional techniques improve learning stability:

- **Entropy Bonus:** Encourages exploration by penalizing deterministic policies too early.
- **Value Loss:** Trains the critic network to estimate state values, reducing variance in the policy gradient.
- **Gradient Clipping:** Prevents unstable updates by limiting the norm of gradients.
- **Checkpointing:** Regularly saves the model, optimizer state, and metadata for recovery and version tracking.

in our system - the PPO agent's rewards combine forward progress, penalties for lateral and heading deviation, and collision costs. This structure guides the policy to learn driving behavior that is both fast and stable.

Reward Function: A reward function is the feedback mechanism in reinforcement learning. It assigns a scalar value at each timestep to quantify how good or bad the agent's action was in a given state. The goal of the learning process is to maximize the cumulative reward, which implicitly guides the agent toward desirable behaviors.
In autonomous driving:
We use the reward function to balance **progress**, defined as moving forward along the track, with **safety**, which emphasizes staying near the centerline, minimizing heading deviation, and **avoiding collisions**.
Without a reward function, the agent has no signal to improve its policy, since raw sensory data alone provides no notion of success or failure.
Reward Function Design:

- **Progress Gain:** Positive reward proportional to forward progress along the track centerline.
- **Lateral Penalty:** Negative reward when the car deviates sideways from the track centerline.
- **Heading Penalty:** Negative reward for misalignment between car orientation and track direction.
- **Collision Penalty:** Fatal mistake - Large negative reward for collisions, terminating the episode.

This combination ensures the policy prioritizes smooth, forward driving while avoiding unsafe maneuvers.

Track Progress and Deviation A cyclic waypoint list approximates the centerline. For each timestep: (i) nearest waypoint and subsequent waypoint define an active segment; (ii) vehicle position is orthogonally projected onto the segment; (iii) cumulative arc length yields fractional lap progress; (iv) lateral deviation is the perpendicular distance; (v) heading deviation is the angular difference between vehicle heading and segment direction.

Telemetry Periodic logging of CPU, memory, GPU allocations and selected internal statistics enables correlation between resource usage and learning dynamics.

3.3.5 Data Structures

- **Waypoint Sequence**: An ordered cyclic list of x, y coordinates.
- **State Vector**: A concatenation of encoded vision features, and normalized IMU values.
- **Rollout Buffers**: Parallel arrays storing states, actions, rewards, log-probabilities, and values.
- **Performance Record**: A structured set of episodic metrics including collisions, and deviation aggregates.

3.3.6 Design Decisions and Trade-offs

Below you can see the tradeoff for the design and implementation decisions. Throughout the working process we've changed hyper-parameters, sensors, dimensionality, etc. to achieve the goal of PPO convergence.

Table 3.1: Design Decisions, Rationales, and Trade-offs

Aspect	**Decision**	**Rationale**	**Trade-off**
Sensor imagery	Semantic instead of raw RGB	Reduces ambiguity and creates a smaller feature space	Loses fine texture cues
Progress calculation	Segment projection method	Provides smooth fractional progress and stable reward shaping	Linear scan over waypoints which could be optimized
RL algorithm	PPO with clipping	Enables stable on-policy updates	Lower sample efficiency than off-policy methods
Publishing rate	Throttled fused state output	Cuts compute cost and avoids redundant frames	Coarser temporal control granularity
Normalization	Online streaming statistics	Memory efficient and allows immediate adaptation	Early distribution bias possible
Checkpointing	Versioned run directories	Ensures reproducibility, rollback, and comparative experiments	Requires disciplined version management
Entropy bonus	Fixed tunable weight	Encourages sufficient exploration	Additional tuning overhead
Action representation	Mixed discrete steering with continuous throttle in later variants	Provides more stable exploration in steering space	Reduced steering resolution compared to fully continuous control
Input reduction	Removed sensors such as GNSS ans LiDAR	Simplifies the model and enables faster inference	Potential loss of redundancy against noise

3.3.7 Reliability and Maintainability

- Assertion-guarded dimensional checks at fusion boundaries.
- Gradient clipping to mitigate exploding updates. (PPO Core by original paper)
- Atomic checkpoint writes preventing partial corruption.
- Version tagging for architectural evolution traceability.
- Modular node separation for fault isolation and parallel iteration.

3.3.8 Summary

The implementation integrates a modular ROS 2-based processing pipeline, high-fidelity simulation, and an on-policy reinforcement learning core. Deterministic geometric progress estimation, streaming normalization, and disciplined versioned checkpoints yield a reproducible and extensible platform for autonomous racing experimentation while balancing computational efficiency and model fidelity.

3.4 Evaluation Metrics

The evaluation of the reinforcement learning agent is conducted using a combination of TensorBoard logs, structured CSV exports, and explicit step counters. This section outlines the metrics recorded, their organization, and the rationale for their inclusion.

TensorBoard (Scalars/Text) During training, we report both per timestep (`/t`) and per episode (`/episode` or `/info`) aggregates. Axes are explicitly labeled: *timestep_counter* for `/t` and *episode* for `/episode, /info`. Examples to loggeed values

- **Progress (Distance Covered)**
  - `Average Distance Covered (m)/t` – mean distance progressed per timestep (m) vs. *timestep_counter*.
  - `Average Distance Covered (m)/episode` – total distance per episode (m) vs. *episode*.
- **Rewards**
  - `Average Reward/t` – average reward per timestep vs. *timestep_counter*.
  - `Average Episodic Reward/info` – total reward per episode vs. *episode*.
- **Lane Keeping (Centerline Deviation)**
  - `Average Deviation from Center/t` – mean lateral deviation (m) per timestep vs. *timestep_counter*. (Lower is better)
  - `Average Deviation from Center/episode` – episode mean lateral deviation (m) vs. *episode*. (Lower is better)
- **Episode Statistics**
  - `Episode Length/info` – timesteps per episode vs. *episode*.
  - `Episode/Episode Duration (s)` – simulated seconds per episode vs. *episode*.

CSV Logging (Per Learning Step) In parallel, training signals are exported to a CSV file named `training_every_learn_step_log.csv`. This file provides a reproducible, script-friendly trace of training, containing:
`episode`, `step` (learn_step_counter), `timestep` (timestep_counter), `actor_loss`, `critic_loss`, `entropy`, `learn_step_reward`, `cumulative_reward`.

3.4.1 Step Counters

To avoid ambiguity in plots, two distinct counters are used:

- *timestep_counter* – the x-axis for actor loss, critic loss, and entropy.
- *learn_step_counter* – the x-axis for all exploration-related metrics (`log_std`, `std`, their means).

3.4.2 Rationale

We track a focused set of signals that reflect both learning quality and driving behavior.

Reward curves at two levels—`Average Reward/t` (per-timestep) and `Average Episodic Reward/info` (per-episode)—reveal short- vs. long-horizon gains. If they diverge, check reward shaping or early terminations.

Distance covered is a reward-independent progress check. It should generally rise with reward; if not, the agent may be exploiting the reward.

Deviation from center measures lane keeping. Lower values together with rising distance suggest cleaner, faster laps.

Episode length reflects survivability (fewer crashes/timeouts). Longer episodes alongside lower deviation typically mean more stable control.

CSV exports mirror these metrics with exact per-step and per-episode traces, enabling reproducibility and post-hoc analysis. Explicit counters (*timestep_counter*, *episode*, *learn_step_counter*) remove ambiguity in plots and logs.

3.4.3 Rationale

Reward curves (`Average Reward/t` vs. *timestep_counter* and `Average Episodic Reward/info` vs. *episode*) measure performance at two levels. If they tell different stories, check reward shaping or early terminations.

Distance covered is a reward-independent sanity check. It should rise with reward. If reward rises but distance falls, you may be exploiting the reward.

Deviation from center measures lane keeping. Lower is better. Lower deviation together with higher distance means cleaner, faster laps; lower deviation without distance gains can mean the policy is too cautious.

Episode stats reflect survivability and pacing. Longer episodes with lower deviation suggest stability; sudden drops usually mean crashes or timeouts.

Chapter 4

Results and Analysis

4.1 Experimental Setup

4.1.1 Goal

Evaluate the trained autonomous racing agent on its ability to successfully make corners and drive well with stable and collision-free performance. The focus is on track curves and consistency.

4.1.2 Environment

Experiments were conducted in the CARLA simulator with deterministic settings to ensure repeatability. The ego vehicle was equipped with a semantic segmentation camera, LiDAR, and IMU sensors. Tracks were generated using waypoint-based centerlines with full road geometry.

4.1.3 Training Procedure

The policy was trained using Proximal Policy Optimization (PPO). Each rollout was followed by advantage and return estimation, multiple mini-batch epochs, and updates with a clipped surrogate objective, value loss, entropy regularization, and gradient clipping. Checkpoints were periodically saved, including model, optimizer state, and metadata.

4.1.4 Control and Actions

The learned policy outputs steering, throttle, and optional braking actions at a throttled rate, applied only once every Nth fused frame. These actions are then passed directly to the CARLA vehicle control interface.

4.1.5 Reward Signal

The reward combines forward progress along the track centerline with penalties for lateral and heading deviation. Collisions produce strong negative rewards. Fractional lap progress is used instead of discrete waypoint counting to provide smoother feedback.

4.1.6 Evaluation Metrics

Metrics such as rewards and episode length were tracked throughout training.[4.2]

4.1.7 Constraints

The experiments used a single-vehicle setup without domain randomization or replay buffers. Track progress relied on a linear waypoint search, which was adequate at current scale but may be a bottleneck at larger scales.

4.1.8 Reproducibility

To ensure consistent experiments, the setup was fully containerized, runs were version-tagged, and deterministic seeds were applied. Checkpoints were written atomically to avoid corruption.

4.1.9 Outcome

A controlled and repeatable pipeline enabling comparative evaluation of architectural and sensor configuration changes with clear, geometry-aligned performance signals.

4.2 Presentation of Results

4.2.1 Scope of Reported Results

The results are presented as a step-by-step view of how the system developed across software versions. The available records highlight changes in architecture, sensor inputs, and training parameters.
This section focuses on:

- Key design and algorithm updates
- How configurations and hyperparameters evolved
- The expected impact of these changes on training stability and driving behavior

4.2.2 Versioned Functional Progression

The main milestones observed during development include:

- Initial setup of codebase and logic. Establishment of the CARLA environment, ROS2 communication pipeline, and first attempts at integrating PPO. Early focus was on basic logic, wiring of simulation nodes, and validating that the vehicle could be controlled end-to-end, even if performance was poor or unstable (v1.x).
- Early stabilization and improved logging mechanisms (v2.x).
- Introduction of a learnable action standard deviation and refined reward shaping (v2.1.x).
- Removal of GNSS to reduce input vector size and simplify the sensor space (v3.0.0).
- Adjustments to rollout frequency and batch size handling (v3.1.x–v3.5.x).
- Experiments with activation functions and weight initialization for actor and critic networks (e.g., switching between `tanh` and leaky ReLU, Xavier and Kaiming). These experiments were aimed at improving gradient flow and avoiding vanishing/exploding gradients, since the stability of policy updates in PPO is sensitive to both activation choice and initial weight scaling.

- Upgrades to the vision model, addition of ground-point filtering, and new penalties for deviation and throttle/brake usage.
- **Integration of a VAE for feature extraction** and testing on a new track (v4.0.0).
- **Simplification of the action space and reduced input dimensionality** (v4.0.1).

4.2.3 Consolidated Hyperparameter Trajectories

Throughout development, several structural and hyperparameter adjustments were made for experimenting and to improve training stability, efficiency, and policy expressiveness. Throughout the workflow and along changes and improvements, we wrote down and tracked the changes. Table 4.1 summarizes the documented changes across versions. Only explicit modifications are listed, with emphasis on observation/action space, hyperparameters, rollout cadence, optimization constraints, and other notable structural adjustments.

Table 4.1: Version History of Model Configurations and Hyperparameter Changes

Version	Observation / Action Space	Rollout & Update Cadence	Optimization / Policy Constraints	Other Structural Adjustments
v2.0.0	Brake action reintroduced	—	PPO math review (correctness emphasis)	Logging reduction (I/O overhead cut)
v2.1.0	—	—	Switched manual decay → learnable log_std	Metadata load fix
v2.1.3	—	—	Cross-version state dict loading logic	Long-run training milestone focus
v3.0.0	Input dim 203→198 (GNSS removed)	—	—	Sensor simplification for generalization
v3.1.1	—	LEARN_EVERY_N_STEPS 128→512; EPISODE_LENGTH 640→1024; MINIBATCH 32→64	—	Scalability toward longer horizon stability
v3.1.2	—	LEARN_EVERY_N_STEPS 512→1024; EPISODE_LENGTH 1024→512	ACTOR LR 1e-4→3e-4; CRITIC LR 2.5e-4→3e-4; EPOCHS 6→4	Batch efficiency tuning
v3.2.0	—	—	Activation ReLU→tanh; Init Kaiming→Xavier	Representation smoothness trial
v3.3.0	—	—	—	Ground filtering; vision module upgrades
v3.3.1	—	EPISODE_LENGTH 512→3000	Reward normalization bug fix	Transition ordering fix; reduced debug overhead
v3.3.2	—	—	log_std upper bound log(1.5) → log(0.4); ACTOR LR 3e-4→2.5e-4	Variance control tightening
v3.3.3	—	—	Determinism disabled (True→False)	Gas/brake ratio penalty; relocation logic
v3.3.4	—	—	Reparameterized tanh-squashed raw action sampling	Policy expressiveness alteration
v3.4.0	IMU removed; ACTION_DIM=192	—	Activations tanh→leaky_relu; Init Xavier→Kaiming_uniform	Latency mitigation experiment
v3.4.1	—	LEARN_EVERY_N_STEPS 1024→2048; MINIBATCH 64→128	—	Larger effective batch stability attempt
v3.4.2	IMU reinstated; brake removed	—	Deviation penalty added	Rebalanced actuation semantics
v3.5.0	Imagenet-initialized vision (implicit feature compression)	LEARN_EVERY_N_STEPS 2048→4096; MINIBATCH 128→512; EPOCHS 3→4	CRITIC LR 3e-4→2e-4	Enhanced feature extractor
v3.5.1	Angular velocity z added; Input dim 197→198	—	PPO_INPUT_DIM 197→198	Sensor richness restoration
v4.0.0	New track; VAE added	—	—	Domain shift + latent representation
v4.0.1	Discrete actions by supervisor's advice; Action every 5 steps	LEARN_EVERY_N_STEPS 4096→1024; MINIBATCH 512→128; EPOCHS 4→5	POLICY_CLIP 0.2→0.1	Latency reduction + tighter trust region
v4.0.2	VAE features + vector of 5 values – input dim = 100	—	POLICY_CLIP 0.1→0.2	Codebase consolidation (formatting)

4.2.4 Derived Interpretive Trends

- **Sensor Economy vs. Stability:** Removal of GNSS forced reliance on remaining modalities, compensated by horizon expansion and encoder upgrades (Imagenet, VAE).
- **Variance Management:** Larger rollouts and minibatches stabilized gradients later downscaling aligned with reduced input and discrete actions.
- **Exploration vs. Exploitation:** Capping the log standard deviation reflected an emphasis on conservative policy updates following domain shifts.
- **Representation Refinement:** Experiments with activation functions and initialization schemes were conducted to evaluate network conditioning under evolving input modalities and reward structures.

4.2.5 Reproducibility Indicators

Version identifiers, before→after hyperparameter notation, and semantic labels provide deterministic provenance. Cross-version state dict loading (v2.1.3) enables fair warm starts.

4.2.6 Limitations of Reported Results

- No empirical statistics such as returns or lap times were reported.
- No confidence intervals or variance measures.
- Interaction effects between changes cannot be disentangled without ablation.

4.2.7 Summary

The result set shows systematic refinement: sensor simplification, compensatory horizon expansion, iterative optimization stabilization, and late-stage compression with discrete actions and latent representation. While quantitative endpoints are absent, the structured evolution provides a reproducible scaffold for future validation.

4.3 Data Analysis and Interpretation

4.3.1 Data Sources and Features

We started with a multisensor setup (RGB camera, LiDAR, GNSS, IMU). During training we simplified the perception stack to a single *semantic segmentation* camera with a VAE encoder, plus a small control/state vector (e.g., normalized velocity, throttle command, deviation from centerline, and related driving signals). This reduction lowered input dimensionality and made the policy updates more stable.

4.3.2 What We Measured

We analyzed the following TensorBoard scalars:

- **Rewards:** `Average Reward/t` and `Average Episodic Reward/info`.
- **Progress:** `Average Distance Covered (m)/t` and `Average Distance Covered (m)/episode`.

- **Lane Keeping:** `Average Deviation from Center/t` and `Average Deviation from Center/episode`.
- **Episode Stats:** `Episode Length/info` (timesteps per episode).

4.3.3 Observed Patterns

After reducing the state size and lowering the step publishing rate, trends improved:

- **Rewards moved up.** Both pertimestep and perepisode reward curves rose, with the perepisode curve showing clearer gains. This indicates the agent is collecting more useful reward over longer horizons.
- **Centerline deviation dropped.** Mean deviation trended down over time. Lower deviation together with higher distance points to cleaner lane keeping at higher effective speeds.
- **Episodes got longer.** `Episode Length/info` increased, which usually means fewer early terminations (crashes/timeouts) and better overall stability.

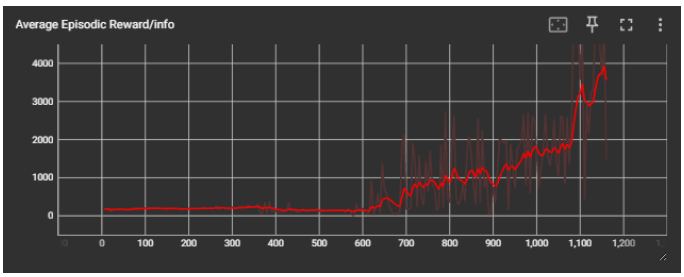


Figure 4.1: Average Episodic Reward. Upward trend indicates better outcome-level performance across episodes.

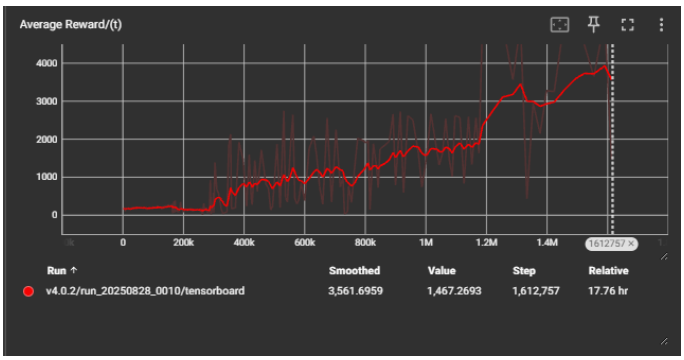


Figure 4.2: Average Reward per Timestep. Tracks update-level learning dynamics; should broadly align with episodic reward.

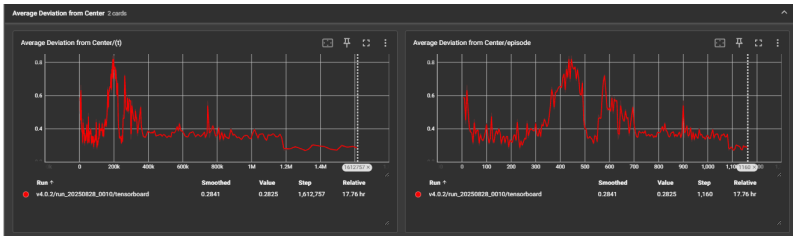


Figure 4.3: Average Deviation from Centerline. Downward trend suggests cleaner lane keeping. (Lower is better)

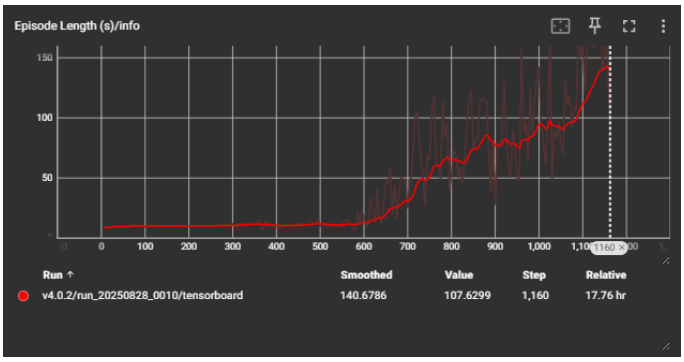


Figure 4.4: Episode Length (episode). Longer episodes usually indicate fewer early terminations and improved stability.

4.3.4 Interpretation

The combination of a simpler input (VAE latent + small state vector) and adjusted update cadence made the policy easier to optimize. Rising rewards with falling deviation indicate the policy is learning to time turns better and hold the lane with fewer corrections. The fact that distance and reward move together is very important: it supports that the reward function is aligned with the task (progress with control), not just exploiting side effects.

We did see intermittent volatility (short drops or spikes) during training. Given the setup, these are likely due to exploration bursts, environment resets, or sensitivity to starting positions. The recovery after such events suggests the policy can restabilize as training continues.

4.3.5 Against Objectives

Our short-term goal was reliable driving on Town 02 with improving lane keeping. The metrics above support that we are moving in the right direction. The longer-term goal is generalization to unseen tracks. We have not fully validated this yet; current results are mainly from Town 02. A proper generalization check will require evaluation on heldout layouts.

4.3.6 Takeaway

Simplifying perception to a VAE over segmentation images and shrinking the state vector, together with pacing the simulator updates, produced steadier learning. The agent now shows clearer progress: higher rewards, more distance, lower deviation, and longer episodes.

4.4 Comparison with Existing Approaches

A relevant project to our work is the implementation by Idrees et al.[16], which combined Proximal Policy Optimization (PPO) with a variational autoencoder (VAE). In their setup, the VAE was trained on semantically segmented images from CARLA, and its latent features were then used as input to a PPO agent. Training and evaluation were performed on the built-in CARLA maps (mainly Town 02 and Town 07), using the CARLA Python client directly for sensor management and control. Their approach provided a clear demonstration of how PPO could be applied to autonomous driving within CARLA using image-based features. Both their PPO implementation and ours rely on Monte Carlo return estimation.
In our project we follow a similar high-level direction, combining a VAE encoder with a PPO agent, with an emphasis on racing scenarios. Like Idrees et al., we currently train and evaluate primarily on Town 02, and our goal is to assess the model's ability to generalize to new and unseen tracks. While their focus was on demonstrating PPO with segmented-image features, our emphasis is on achieving robust racing performance and measuring generalization across different layouts. The overall system design relies exclusively on the VAE for perception, while maintaining modularity through ROS2 integration and Dockerized deployment for scalability and reproducibility.
In this way, our work suggests to extends earlier PPO-based approaches by shifting from general autonomous navigation to the challenge of generalization beyond the training track.

4.5 Discussion of Findings

During training we faced practical issues and challenges in the simulator setup, environment configuration, and mainly policy convergence. With guidance from the supervisor, we simplified the input state that feeds the PPO network (fewer features, lower dimensionality) and reduced the step publishing rate. After these changes, behavior improved both visually and in the logs.
Quantitatively, we observed a clear upward trend in `Average Episodic Reward` and a downward trend in `Average Deviation from Center`. These signals match what we saw in rollouts: the vehicle began to time turns more reliably and hold the lane with fewer corrections.
Qualitatively, the agent shows much better awareness of track geometry. The encoder's latent features, together with simple control signals (e.g., velocity and throttle), give the policy enough context to stabilize learning and make earlier, more deliberate steering decisions.
Overall, simplifying the state and tuning the update cadence coincided with more stable training. We will continue to monitor these metrics as we push for stronger generalization to new unseen tracks.

Chapter 5

Conclusion and Future Work

5.1 Summary of Contributions

The project delivers an integrated autonomous racing simulation stack that combines a high-fidelity vehicular simulator, modular robotic middleware, and an on-policy reinforcement learning agent. Core achievements include:

- A decoupled, service-oriented node architecture separating map ingestion, vehicle spawning, simulation state coordination, control actuation, data fusion, and learning.
- A sensor fusion pipeline unifying semantic camera, LiDAR, and inertial signals into a normalized fixed-length state representation with online statistical adaptation.
- Geometric track progress and deviation estimation enabling dense, informative reward shaping beyond coarse waypoint counting.
- A reproducible training lifecycle featuring semantic versioning, checkpoint semantics, and structured performance logging that includes lap progression, collisions, and deviation.
- Policy optimization via Proximal Policy Optimization (PPO), with clipped objective, entropy regularization, and synchronized rollout/update cadence.
- Containerized deployment enabling deterministic orchestration across simulation, middleware, learning, and monitoring layers.
- An interactive monitoring interface for presenting system status, research goals, and directing users to training analytics.

These components collectively advance a controlled experimental platform for data-driven autonomous racing under continuous improvement.

5.2 Impact

The system provides a reusable foundation for rapid iteration on perception, control, and reinforcement learning strategies in closed loop racing scenarios. Its emphasis on modularity, observability, and version traceability supports comparative studies, ablation analyses, and reproducible benchmarking. The geometric progress estimator and fused proprioceptive–exteroceptive state design contribute to shaping richer learning signals than naïve position tracking alone.

5.3 Limitations

- **Domain Restriction:** All development and evaluation occur in simulated environments; real-world transfer gaps remain unaddressed.
- **Perception Simplification:** Reliance on semantic imagery reduces raw appearance richness; absence of RGB or depth may limit emergent behaviors.
- **Algorithmic Efficiency:** On-policy PPO sacrifices sample efficiency compared to off-policy or model-based alternatives; no experience replay or auxiliary predictive tasks are used.
- **Scalability Constraints:** Waypoint proximity and projection rely on linear scans without spatial indexing-acceptable at current track scale but suboptimal for larger maps.
- **Limited Adaptivity:** No curriculum, domain randomization, or automated hyperparameter optimization pipelines are integrated.
- **Safety & Robustness:** No formal verification, fail safe supervisory logic, or shielded action filtering is implemented.
- **Underutilized Modules:** Placeholders for extended vision or generative representation learning are not yet exploited in the active training loop.

5.4 Future Research Directions

Sample Efficiency & Learning Paradigms

- Integrate off-policy methods or hybrid on/off-policy approaches.

Representation Learning

- Introduce variational or hierarchical encoders for compressing multi-modal observations.
- Evaluate latent stability under domain shifts.

Domain Randomization & Sim-to-Real

- Vary lighting, surface friction, sensor noise, and obstacle configurations to improve robustness and transferability.

Multi-Agent and Competitive Settings

- Extend environment to adversarial or cooperative multi-vehicle scenarios.

Hierarchical & Safe Control

- Layer a high-level strategic planner over a learned low-level controller.
- Integrate safety shields or control barrier functions.

Adaptive Reward and Auto-Tuning

- Apply meta-optimization or Bayesian optimization to dynamically adjust reward weights and hyperparameters.

Continuous Deployment & MLOps

- Automate experiment tracking, model lineage, and drift detection with a fully realized persistence and analytics service.

Real-Time Performance Engineering

- Profile bottlenecks in sensor fusion and waypoint search.
- Accelerate with vectorized kernels or GPU pipelines.

5.5 Concluding Remarks

The project establishes a principled, extensible baseline for autonomous racing research, emphasizing modular design, reproducibility, and geometric–sensor fusion in reinforcement learning control. While effective within its current scope, its broader potential lies in extending learning efficiency, realism, and safety. The outlined future directions chart a path toward richer perception, scalable coordination, and transferable autonomous driving competencies.

References

[1] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," in *arXiv:1707.06347*, 2017.

[2] X. Liang, T. Du, H. Xu *et al.*, "Deep reinforcement learning for autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2018.

[3] Y. Song *et al.*, "Autonomous driving using deep reinforcement learning," *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[4] D. Feng, C. Haase-Schuetz *et al.*, "Deep multi-modal object detection and semantic segmentation for autonomous driving: Datasets, methods, and challenges," *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[5] H. Caesar *et al.*, "nuscenes: A multimodal dataset for autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

[6] A. Dosovitskiy, G. Ros *et al.*, "Carla: An open urban driving simulator," in *Conference on Robot Learning*, 2017.

[7] K. Makantasis *et al.*, "Deep reinforcement learning for autonomous racing," *Robotics and Autonomous Systems*, 2022.

[8] H. Chen *et al.*, "End-to-end autonomous racing using reinforcement learning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2022.

[9] Y. Pan *et al.*, "Imitation learning for autonomous racing," in *IEEE International Conference on Robotics and Automation*, 2020.

[10] M. Jaritz *et al.*, "End-to-end race driving with deep reinforcement learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2018.

[11] H. Narasimhan *et al.*, "Curriculum learning for autonomous driving," in *NeurIPS Workshop*, 2020.

[12] D. Ha and J. Schmidhuber, "World models," in *arXiv preprint arXiv:1803.10122*, 2018.

[13] D. Hafner *et al.*, "Learning latent dynamics for planning from pixels," in *International Conference on Machine Learning*, 2019.

[14] M. Bojarski *et al.*, "End to end learning for self-driving cars," in *arXiv preprint arXiv:1604.07316*, 2016.

[15] Y. Pan *et al.*, "Adversarially guided actor-critic," in *arXiv preprint arXiv:1802.02274*, 2018.

[16] I. Shaikh, "Autonomous driving in carla using deep reinforcement learning," GitHub repository, 2023. [Online]. Available: https://github.com/idreesshaikh/Autonomous-Driving-in-Carla-using-Deep-Reinforcement-Learning