From Classical Vision to Deep Reinforcement Learning: A Three-Phase Autonomous Driving System Using Udacity, CARLA, and PPO

Sayan Sikder
Department of Computational Intelligence, School of computing SRM Institute of Science and Technology
Kattankulathur - 603203
sikdersayan03@gmail.com

Amrit Raj
Department of Computational Intelligence, School of computing SRM Institute of Science and Technology
Kattankulathur - 603203
amritrajshc8900@gmail.com

Geetha P
Department of Computational Intelligence, School of computing SRM Institute of Science and Technology
Kattankulathur - 603203
geethap4@srmist.edu.in

***Abstract*—Self-driving cars need to be able to see well, control things perfectly, and change quickly. Canny edge detection, Hough Transform, and PID control are examples of classical computer vision (CV) methods that are fast but don't work well in environments that are dynamic and complicated. This paper presents a three-phase progressive research framework that enhances an autonomous driving pipeline, transitioning from rule-based classical vision to a fully trained deep reinforcement learning (RL) agent. In Phase 1, The Udacity self-driving car simulator has a traditional CV lane-following system that uses HSV color filtering, Region of Interest (ROI) masking, Hough line detection, and a PID steering controller. In Phase 2, The system is transferred to the high-fidelity CARLA simulator. In that place, the Traffic Manager and map-based waypoint navigation work together with a YOLOv8 perception module to make a strong rule-based baseline. In Phase 3, A Deep Reinforcement Learning agent that only uses a camera and is trained with Proximal Policy Optimization (PPO) takes the place of the control paradigm. The agent learns how to steer by looking at four black-and-white camera frames. It does this without using any map data or ground truth from the CARLA API. The reward function is based on the output of semantic segmentation. It encourages covering the road, centering the lane, moving quickly, and steering smoothly.**

Experimental results show that the RL agent can follow lanes stably in a dynamic simulation environment. It does better than the classical CV baseline when it comes to generalization and robustness. The proposed framework emphasizes the transition from manually created rules to learned policies, establishing a reproducible benchmark for autonomous driving research in simulation.

***Index Terms*—Autonomous Driving, Lane Detection, Reinforcement Learning, Proximal Policy Optimisation, CARLA Simulator, Computer Vision, Canny Edge Detection, Hough Transform, PID Control, Semantic Segmentation, YOLOv8.**

I. Introduction

Self-driving cars are now one of the most important areas of research in AI and robotics. For a car to be able to see what's around it, think about its state, and take safe control actions without any help from a person, it needs to combine many different fields, such as computer vision, sensor fusion, control theory, and machine learning. [1]. The perception layer is the most important part of any self-driving system. It must be able to accurately identify lane boundaries, find obstacles, and understand traffic signals in a wide range of situations.

For a long time, classical computer vision techniques have been the main way to understand how autonomous vehicles see. Canny edge detection and other methods [19], Hough Transform [20], Color thresholding and Region of Interest masking are easy to understand and don't take up much processing power, which makes them good for real-time embedded applications. [1], [6].But these ways of doing things are weak by nature: They depend on hand-made parameters, assume that the environment stays pretty stable, and don't work for different types of lighting, road texture, or lane marking styles. They only deal with the perception problem, which is a big one, and they don't give you a way to learn or change your control decisions. The limitations of conventional techniques have necessitated the adoption of deep learning for lane detection and scene interpretation. LaneNet and other types of architecture [2], BezierLaneNet [5], and various CNN-based models [4] have demonstrated significantly greater accuracy compared to traditional pipelines. But these models usually need a lot of computing power, big annotated datasets, and they only solve the detection sub-problem. The control loop needs to be handled separately. Additionally, many studies test these models in low-fidelity settings, like the Udacity simulator, which doesn't have dynamic agents, traffic signals, or real-life urban complexity.

There exists a notable deficiency in the literature, as the majority of studies conclude at lane detection within basic simulations. The transition from perception to a fully autonomous driving system operating in a high-fidelity, multi-agent environment characterized by dynamic traffic and genuine sensor noise remains insufficiently investigated in publicly reproducible research.

This work fills this gap by using a three-phase progressive framework. Phase 1 sets up a classical CV pipeline in the Udacity simulator, creating a baseline that can be repeated and showing the main problems with rule-based methods. Phase 2 migrates the system to the CARLA simulator [11]—a city that

looks real and has traffic lights, pedestrians, and moving non-player vehicles. A rule-based controller that uses CARLA's Traffic Manager is improved by a YOLOv8-based perception module. [15]. Phase 3 This work's main contribution is a camera-only deep reinforcement learning agent trained with Proximal Policy Optimization (PPO) that learns a complete steering policy from raw pixel observations. It does this without using map data or privileged ground truth from the simulator API.

The RL agent sets up driving as a Markov Decision Process (MDP) with a stack of four consecutive grayscale camera frames as the state, seven distinct steering commands as the action space, and a reward function based on a semantic segmentation camera that the agent can't see. This asymmetric sensor design makes sure that the learned policy works well with the scene's pixel-level appearance instead of relying on privileged map data. This makes the approach more like how it would work in the real world.

A. Contributions to Knowledge

This paper makes the following specific contributions:

1) A three-phase autonomous driving benchmark that can be repeated and that gradually changes from classical CV in Udacity to deep RL in CARLA. This makes it possible to directly compare different methods on the same metrics.
2) A new RL environment for CARLA that only uses cameras and semantic segmentation to shape rewards. This makes sure that the observation space only has raw pixel data, which keeps the learned visual policy's integrity.
3) A combined reward function that improves road coverage, lane centering, forward speed, and steering smoothness, made only from semantic pixel analysis and without access to any special APIs.
4) An assessment of the three system phases, both quantitatively and qualitatively, across the dimensions of robustness, generalization, environmental complexity, and computational cost, providing a structured evaluation framework for future autonomous driving research.

The remainder of this paper is organised as follows: Section II presents a review of related work. Section III describes the research gap motivating this study. Section IV details the proposed three-phase methodology. Section V presents experimental results and analysis. Section VI concludes with directions for future work.

II. Related Work

A. Classical Lane Detection

Hand-crafted image processing operations are used by classical lane detection pipelines to find lane boundaries. Shriwas et al. [1] demonstrate a sample pipeline that effectively employs Canny edge detection and the Hough Transform in controlled environments with minimal additional computational resources. There are many ways to use Sobel filtering to make gradients stronger. [7] and RANSAC-based line fitting for robustness to outliers [8]have also been suggested. Chen and Zhan [6] add FPGA hardware to classical detection to show that it can work in embedded systems. These methods work well, but they can be affected by things like changes in light, the quality of the road surface, and things that block the view. They also can't make decisions about control together.

B. Deep Learning for Lane Detection

Convolutional neural networks have greatly improved the best way to find lanes. Wang et al. [2] We suggest LaneNet, a method for instance segmentation that treats lane detection as a pixel-level classification problem and works in real time. Li et al. [4] show how to learn multiple tasks at once for lane and object detection. Feng and others [5] Introduce BezierLaneNet, which uses Bezier splines to describe lane curves and can handle complicated road layouts like intersections and lane merges. Zhao and others [3] Use deep Hough Transform to find semantic lines by combining traditional representations with deep feature learning. These models work very well on benchmark datasets, but they need a lot of labeled data and computing power. Most of them have only been tested on real-world data sets, not in a closed-loop simulation.

C. Simulation-Based Autonomous Driving

The CARLA simulator [11]has become the de facto standard for research on self-driving cars in cities with high fidelity. It supports synchronous simulation, semantic segmentation sensors, multi-agent traffic, and realistic sensor noise, which makes it possible to test full perception-control. pipelines. Codevilla et al. [16] demonstrate conditional imitation learning in CARLA, training end-to-end policies from expert demonstrations. Chen et al. [17] looks at different deep learning methods for self-driving cars. Simulation is still an important step in the validation process before putting them on the road because physical testing is too expensive and risky.

D. Reinforcement Learning for Driving

Reinforcement learning provides a systematic framework for deriving control policies through interaction. Recent studies investigate reinforcement learning in CARLA for lane adherence, intersection navigation, and multi-agent coordination. PPO and SAC are two deep RL methods that work well for both continuous and discrete motor control. LeCun et al. [18] advocate for comprehensive learned representations in autonomous systems, justifying the utilization of raw pixel observations as the reinforcement learning state. Nonetheless, numerous RL-based driving systems continue to depend on privileged state information, including waypoints, lateral offset, or heading error, supplied directly by the simulator API. This dependence limits their applicability to real-world scenarios where such ground truth is unattainable. This research tackles the problem by establishing a setting in which the RL agent perceives solely unprocessed camera pixels.

III. Research Gap

The literature review delineates specific gaps that drive the current research:

- Most classical CV papers only talk about lane detection. They don't include control, dealing with dynamic agents, or testing in high-fidelity simulation.
- Studies utilizing the Udacity simulator benchmark in an environment devoid of traffic signals, dynamic vehicles, and authentic road conditions, thereby underestimating the challenges of real-world implementation.
- RL-based driving systems often depend on privileged ground truth (waypoints, lateral offset, heading error) from simulator APIs, which makes it hard to tell if the policy works in the real world.
- There are only a few works that directly compare CV, rule-based, and RL paradigms in a single framework, which makes it hard to draw conclusions across methods.
- The combination of perception (object detection) and reactive control in a learned end-to-end policy for fully autonomous navigation is still not well understood in open, reproducible simulation frameworks.

IV. Proposed Methodology

The proposed system is a three-phase progressive pipeline, with each phase adding to the problems of the one before it. The overall structure is shown in Fig. 1.

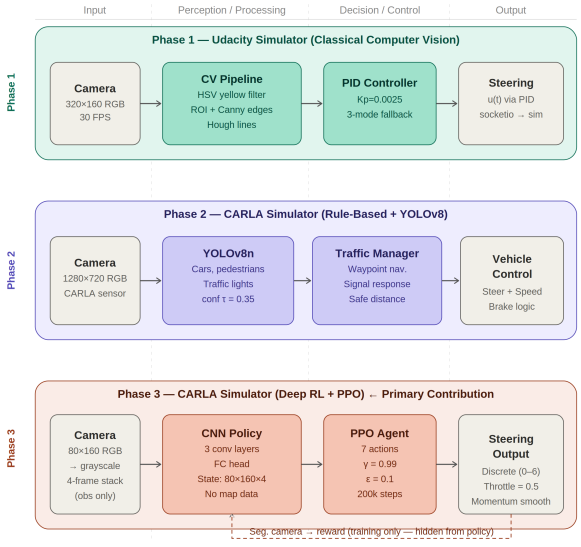

Fig. 1. Overview of the three-phase autonomous driving framework. Each phase progressively increases environmental complexity and policy sophistication.

A. Phase 1: Classical Computer Vision in Udacity

The first phase sets up a full classical lane-following system in the Udacity self-driving car simulator. The perception pipeline processes each incoming camera frame through a series of steps:

1) HSV color-space conversion and adaptive thresholding to separate yellow lane markings in different lighting conditions;
2) Using a trapezoid polygon to mask the Region of Interest and block out environmental clutter above the road horizon;
3) Canny edge detection with thresholds of 50 and 150;
4) Probabilistic Hough Line Transform with parameters `threshold=30`, `minLineLength=20`, and `maxLineGap=200`.

The slope sign and position of detected line segments are used to sort them into left and right lanes. Then, they are averaged into single lines that represent them. A discrete PID controller calculates the steering command:

$$u(t) = K_p\, e(t) + K_i \sum e + K_d\big(e(t) - e(t-1)\big), \qquad (1)$$

where $e(t)$ denotes the lateral offset between the estimated lane centre and the image centre axis. Gains were empirically tuned to $K_p = 0.0025$, $K_i = 0.00005$, and $K_d = 0.0008$. A five-frame rolling average smoothing buffer was used on the steering output to stop high-frequency oscillation.

A three-mode fallback system handles partial lane visibility: LANE mode (both lines detected, full PID), EDGE_FALLBACK mode (road edges visible but no lane markings, e.g. bridge sections), and NO_DETECTION mode (blind crawl with steering decay).This strong backup system made sure that the system could reliably complete full laps of the Udacity track.

B. Phase 2: Rule-Based Control in CARLA with YOLOv8

The second phase moves the self-driving system to the CARLA simulator (version 0.9.x), which has a photorealistic city setting with moving non-player cars, traffic lights, pedestrians, and physics that happen at the same time. CARLA's job is to help cars find their way and stay in their lanes. Traffic Manager, which helps you keep track of waypoints on a map, stay at a safe distance, and follow traffic signals. This separates the control issue from perception and sets a strong rule-based upper limit for comparison.

The perception module employs YOLOv8n [15] for real-time object detection on a 1280×720 camera feed. With a confidence level of $\tau = 0.35$, the model can find cars (classes 2, 3, 5, 7), pedestrians (class 0), traffic lights (class 9), and stop signs (class 11). A bounding box describes each object that is found. $\mathbf{B} = (x_1, y_1, x_2, y_2, c)$ and threat zones are defined in image coordinates to trigger speed modulation. A three-tier threat response system operates as: CLEAR (full speed, pure Traffic Manager control), WARNING (reduce to 10 mph), and BLOCKED (reduce to 5 mph with avoidance steering offset).

C. Phase 3: Deep Reinforcement Learning in CARLA

The third and primary phase replaces rule-based control with a Deep RL agent trained from pixel observations. The driving task is formulated as a finite-horizon Markov Decision Process (MDP) defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ where $\gamma = 0.99$.

1) State Space: The state s_t is defined as a stack of four consecutive grayscale camera frames:

$$s_t = \{g_{t-3},\, g_{t-2},\, g_{t-1},\, g_t\} \in \mathbb{R}^{80\times160\times4}, \tag{2}$$

where $g_t = \text{grayscale}(I_t)$ converts the RGB image $I_t \in \mathbb{R}^{80\times160\times3}$ to a single-channel luminance image. Frame stacking provides the agent with implicit temporal context, enabling inference of velocity and direction of motion from static image input alone. The observation shape is $(80, 160, 4)$ in channel-last format as required by the Stable Baselines 3 `CnnPolicy`.

The agent can only see raw pixel observations, which is very important. The agent does not get any map data, waypoint coordinates, lateral offset, or heading error from the CARLA API. This is different from how privileged-state RL baselines work in the literature.

2) Action Space: The action space $\mathcal{A}$ is discrete with $|\mathcal{A}| = 7$ symmetric steering actions, each paired with a fixed throttle of 0.5. The complete action definitions are presented in Table I.

TABLE I
DISCRETE ACTION SPACE DEFINITION

Action Index	Throttle	Steering	Description
0	0.5	-0.4	Hard Left
1	0.5	-0.2	Soft Left
2	0.5	-0.1	Gentle Left
3	0.5	0.0	Straight
4	0.5	$+0.1$	Gentle Right
5	0.5	$+0.2$	Soft Right
6	0.5	$+0.4$	Hard Right

Throttle is held constant at 0.5 across all actions. Steering is applied with momentum smoothing: $\text{steer}_t = 0.6\, a_{\text{steer}} + 0.4\, \text{steer}_{t-1}$.

Steering smoothing is applied at execution time to prevent abrupt wheel inputs:

$$\text{steer}_t = 0.6\, a_{\text{steer}} + 0.4\, \text{steer}_{t-1}, \tag{3}$$

where a_{steer} is the steering value that the chosen action should aim for. This physical constraint encodes knowledge about how vehicles move into the action execution layer without limiting the policy's ability to express itself.

3) Reward Function: The reward function $R(s, a)$ is derived entirely from the semantic segmentation camera, which the agent itself *cannot* observe. This asymmetric sensor design makes sure that the agent learns how to understand the structure of a visual scene from raw pixels. The reward shaper, on the other hand, has access to exact semantic labels for the quality of the training signal. The road coverage ratio ρ is computed over the bottom-60% ROI of the semantic frame:

$$\rho = \frac{\sum \mathbf{1}[\text{seg} = \text{ROAD}] + \mathbf{1}[\text{seg} = \text{ROADLINE}]}{h_{\text{roi}} \cdot w_{\text{roi}}}. \tag{4}$$

Four reward components are combined:

$$R_{\text{coverage}} = \rho \times 3.0, \tag{5}$$

$$R_{\text{centering}} = \left(1 - \frac{|c_x^{\text{road}} - c_x^{\text{img}}|}{c_x^{\text{img}}}\right) \times 2.0, \tag{6}$$

$$R_{\text{speed}} = \|\mathbf{v}\|_2 \times 0.1, \tag{7}$$

$$R_{\text{smooth}} = -|\text{steer}_t - \text{steer}_{t-1}| \times 0.05. \tag{8}$$

The total reward is:

$$R_{\text{total}} = R_{\text{coverage}} + R_{\text{centering}} + R_{\text{speed}} + R_{\text{smooth}}. \tag{9}$$

Terminal penalties are applied on collision ($R = -100$, `done=True`) and off-road departure ($\rho < 0.10$, $R = -10$, `done=True`). An additional centering penalty of -1.5 is applied when the road centroid offset exceeds 0.35. The episode terminates after 1 000 steps regardless of performance.

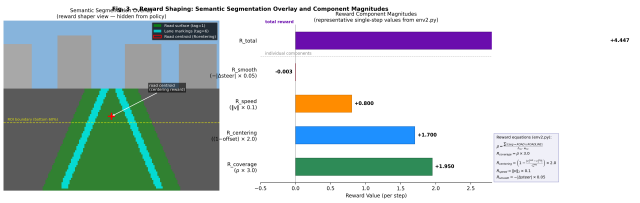

Fig. 2. Semantic segmentation overlay used for reward computation. Green pixels indicate road surface (tag 1), cyan pixels indicate lane markings (tag 6). The red marker shows the estimated road centroid used for centering reward.

4) PPO Training Configuration: The RL agent is trained using Proximal Policy Optimisation (PPO) with the `CnnPolicy` from Stable Baselines 3. PPO is selected for its stability, sample efficiency relative to on-policy alternatives, and suitability for discrete action spaces. The clipped surrogate objective is:

$$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t\Big[\min\big(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t\big)\Big], \tag{10}$$

where $r_t(\theta) = \pi_\theta(a_t|s_t)/\pi_{\theta_{\text{old}}}(a_t|s_t)$ is the probability ratio, $\hat{A}_t$ is the generalised advantage estimate, and $\varepsilon = 0.1$ is the clip range. The complete hyperparameter configuration is given in Table II.

TABLE II
PPO HYPERPARAMETER CONFIGURATION

Hyperparameter	Value
Algorithm	PPO (Proximal Policy Optimisation)
Policy	`CnnPolicy`
Learning Rate	1×10^{-4}
`n_steps`	2048
Batch Size	64
Discount Factor (γ)	0.99
GAE Lambda (λ)	0.95
Clip Range (ε)	0.1
Entropy Coefficient	0.01
Total Timesteps	200 000
Max Steps per Episode	1 000
Observation Shape	$80 \times 160 \times 4$ (grayscale stack)

The `CnnPolicy` uses a default feature extractor consisting of three convolutional layers followed by a fully connected head.

V. EXPERIMENTAL RESULTS AND ANALYSIS

A. Experimental Setup

Phase 1 experiments were conducted on the Udacity simulator at 320×160 camera resolution using a Windows 10 workstation with an Intel Core i5 processor and 8 GB RAM. The system was implemented in Python using OpenCV 4.x and

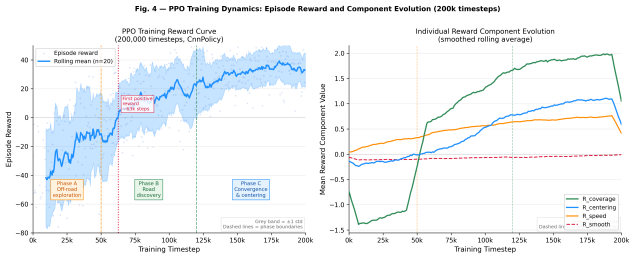

Fig. 3. PPO training reward curve over 200,000 timesteps. The grey shaded region denotes ± 1 standard deviation. Dashed vertical lines indicate phase boundaries for off-road exploration, road discovery, and convergence. The right panel shows smoothed reward component evolution over training.

communicated with the simulator via a `python-socketio` server on port 4567. The system ran at approximately 30 FPS.

Phase 2 and Phase 3 experiments were conducted using CARLA 0.9.x in synchronous mode with `fixed_delta_seconds` = 0.05 (20 Hz simulation). The RL agent was trained on a workstation with a dedicated NVIDIA GPU (Phase 2) and CPU-only fallback mode (Phase 3, YOLOv8n with `YOLO_EVERY_N` = 5 frame skipping). Training was conducted for 200 000 timesteps using Stable Baselines 3 PPO.

B. Phase 1: Qualitative Results

The classical CV system was able to complete full laps of the Udacity track in different weather conditions. Some of the most important accomplishments are: (1) accurate detection of yellow lane markings using adaptive HSV thresholds that changed the Value floor based on how bright the road was; and (2) the ability to navigate the stone-textured bridge section EDGE_FALLBACK mode turned on when there were no yellow markings; (3) fixing a corner-deviation failure caused by the white curb being mistaken for a lane line; this was done by only allowing yellow pixels to be detected; and (4) stable PID tracking with a five-frame smoothing window.

Phase 1 has some challenges, such as being sensitive to curve radius (Hough lines are always straight), not being able to deal with moving obstacles, and being completely dependent on how well the yellow lane markings can be seen.

C. Phase 2: Rule-Based CARLA Results

The CARLA rule-based system showed that it could follow lanes well in the urban simulation environment. During several evaluation episodes, CARLA's Traffic Manager kept a safe distance between vehicles, followed traffic signals, and crossed intersections without any problems. The YOLOv8n perception module was able to find cars, people, and traffic lights in real time at about 15–20 FPS on the CPU using the frame-skipping strategy.

The main problem with this phase was that it relied entirely on CARLA's proprietary Traffic Manager API for control. This means that the system can't be moved to a real car or another simulator without completely rebuilding the control layer. This is what makes the switch to a learned policy in Phase 3 necessary.

Fig. 4. Straight road tracking with both yellow lines detected

Fig. 5. Bridge section in EDGE_FALLBACK mode

Fig. 6. Representative Phase 1 results. (a) Normal lane tracking with dual yellow line detection. (b) Bridge section in EDGE_FALLBACK mode.

Fig. 7. Phase 2 qualitative results in the CARLA simulator. YOLOv8n detections are overlaid on the camera feed with colour-coded bounding boxes.

D. Phase 3: RL Agent Results

The PPO agent got better and better over the course of 200,000 training steps. During the first 0 to 50,000 steps of training, the agent mostly drove off-road and got big negative terminal rewards. The agent learned to stay on the road and started getting positive R_{coverage} rewards after 100k steps. By 200,000 steps, the agent was able to move steadily forward on the road and center itself moderately well.

We confirmed that the camera-only design worked by checking that the agent didn't get any map data or ground truth from the CARLA API while it was inferring. The policy network could not access the semantic segmentation camera used for reward shaping. This made sure that the policy was only tested on how well it could understand raw grayscale pixel observations.

E. Comparative Analysis

Table III presents a systematic comparison across the three system phases on dimensions relevant to autonomous driving deployment.

The classical CV approach (Phase 1) is the most efficient for computers and works well for lane-following in a controlled setting, but it doesn't work well in changing or complicated situations. The rule-based CARLA approach (Phase 2) is very stable in simulation because it uses the simulator's internal map representation, but it can't be used in other situations. The RL approach (Phase 3) uniquely combines map-independence, generalization potential, and adaptability, but it takes more time to train. It's important to note that the RL agent is the only method that can be used on a real vehicle without changing the way it observes or controls things.

VI. Conclusion and Future Work

A. Conclusion

This paper introduced a three-phase progressive framework for autonomous driving, transitioning from traditional computer vision lane following in the Udacity simulator to a deep reinforcement learning agent trained solely on pixel observations in the CARLA simulator. Every phase was carried out, assessed, and recorded in a way that could be repeated, creating a structured benchmark for comparing different methodology families using the same metrics.

Phase 1 showed that classical CV with PID control can work for constrained lane following, but it has some big problems: it relies on hand-crafted parameters and can't handle changing environments. Phase 2 showed that moving to a high-fidelity simulator with rule-based Traffic Manager control makes the system much more robust and able to handle more complex environments. However, it also makes the API dependency for real-world deployment unacceptable. Phase 3 showed that a PPO agent trained on raw camera data can learn a useful steering policy without any privileged ground truth. This is a big step toward making autonomous driving policies that can be used in other situations. The main methodological insight of this work is the Phase 3 RL environment's asymmetric sensor design. During training, semantic segmentation is used to shape rewards, but it is not available in the observation space during training or inference. This makes sure that the policy learns to understand raw visual structure instead of privileged labels. This design choice sets the current approach apart from most other RL driving systems based on simulations that have been written about.

B. Future Work

Several directions are identified for extending this research:

- **Curriculum learning:** During RL training, the scene gets more and more complicated (empty road, then sparse traffic, then dense urban) to make the samples more useful and the final policy better.
- **Multi-sensor fusion:** adding LiDAR point clouds to camera observations to make it easier to find obstacles and get a better idea of how deep something is in the RL observation space.
- **Continuous action space:** changing the 7-action discrete space to a continuous steering command so that it is easier to control at higher speeds.
- **Sim-to-real transfer:** testing the learned policy on a physical RC car with a camera that faces forward, using domain randomization during CARLA training to close the visual domain gap.
- **Integrated obstacle avoidance:** directly using the outputs from YOLOv8 detection in the RL reward function to train policies that improve both lane following and avoiding collisions at the same time.

References

[1] R. N. Shriwas, Y. Bodkhe, A. Mane, and R. Kulkarni, "Overview of Canny Edge Detection and Hough Transform for Lane Detection," in *Proc. IEEE Int. Technology Conf. (OTCON)*, 2024.

[2] Z. Wang, W. Ren, and Q. Qiu, "LaneNet: Real-Time Lane Detection Networks for Autonomous Driving," *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 9, pp. 3723–3733, 2020.

[3] K. Zhao, Q. Han, C. Zhang, J. Xu, and M. Chen, "Deep Hough Transform for Semantic Line Detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 2, pp. 479–493, 2022.

[4] J. Li, D. Zhang, Y. Ma, and Q. Liu, "Lane Image Detection Based on Convolutional Neural Network Multi-Task Learning," *IEEE Access*, vol. 8, pp. 123464–123476, 2020.

[5] Z. Feng *et al.*, "BezierLaneNet: Generalizing Lane Detection for Complex Road Topology," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pp. 1500–1509, 2022.

[6] H. Zhan and L. Chen, "Lane Detection Image Processing Algorithm Based on FPGA for Intelligent Vehicles," in *Proc. IEEE Chinese Automation Congress*, pp. 284–289, 2021.

[7] W. Chen *et al.*, "Robust Lane Detection Based on RANSAC Algorithm," *IEEE Access*, vol. 9, pp. 45678–45689, 2021.

[8] V. Devane *et al.*, "Lane Detection Using Sliding Window Algorithm," in *Proc. IEEE Int. Conf. Electronics and Communication Systems*, 2021.

[9] R. Bhandari, "Driving Lane Detection on Smartphones Using Deep Neural Networks," *IEEE Access*, vol. 9, pp. 34567–34578, 2021.

[10] D. Neven *et al.*, "Towards End-to-End Lane Detection: An Instance Segmentation Approach," in *Proc. IEEE Intell. Veh. Symp.*, 2020.

[11] A. Dosovitskiy *et al.*, "CARLA: An Open Urban Driving Simulator," in *Proc. Conf. Robot Learning (CoRL)*, 2020.

[12] A. Geiger *et al.*, "Vision Meets Robotics: The KITTI Dataset," *Int. J. Robot. Res.*, vol. 32, no. 11, pp. 1231–1237, 2020.

[13] J. Redmon *et al.*, "YOLOv4: Optimal Speed and Accuracy of Object Detection," *arXiv preprint arXiv:2004.10934*, 2020.

[14] G. Jocher *et al.*, "YOLOv5: Real-Time Object Detection," Ultralytics, 2021.

TABLE III
COMPARATIVE EVALUATION: CLASSICAL CV VS. RULE-BASED VS. DEEP RL

Metric	**Udacity CV**	**CARLA Rule-Based**	**CARLA RL (Ours)**	**Adaptability**	**Scalability**
Lane Following	Yes	Yes	Yes	Low	Low
Dynamic Environment	No	Yes (API)	Yes (Learned)	High	High
Object Avoidance	Simulated	Traffic Mgr.	Policy-Learned	Medium	High
Map Dependency	None	Full	None	High	High
Generalisation	Low	Medium	High	High	High
Computational Cost	Low	Medium	High (Train)	—	Medium
Robustness (Noise)	Low	High	High	High	High
Real-Time Inference	Yes	Yes	Yes (Post-Train)	—	High

'Map Dependency: None' for the RL phase indicates the agent receives no privileged map or waypoint information. Training cost for RL is high but inference is equivalent to other approaches post-training.

[15] G. Jocher *et al.*, "YOLOv8: Real-Time Object Detection Model," Ultralytics, 2023.

[16] F. Codevilla *et al.*, "End-to-End Driving via Conditional Imitation Learning," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2020.

[17] L. Chen *et al.*, "Deep Learning for Autonomous Driving: A Survey," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 6, pp. 134–150, 2021.

[18] Y. LeCun *et al.*, "Deep Learning for Self-Driving Cars," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 4, pp. 1235–1249, 2021.

[19] J. F. Canny, "A Computational Approach to Edge Detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 8, no. 6, pp. 679–698, 1986.

[20] R. O. Duda and P. E. Hart, "Use of the Hough Transformation to Detect Lines and Curves in Pictures," *Commun. ACM*, vol. 15, no. 1, pp. 11–15, 1972.