Multi-Agent Reinforcement Learning for Autonomous Micro-Scale Driving

ENPH 479

By,

Felipe Garavelli, Sasan Ghasaei

William Gibbs, Mahdi Shakouri, Itai Boss

Project Sponsors:

Wayve Technologies Ltd.

Dr. Daniele Reda, Dr. Matthew Brown

Project 2561

Engineering Physics 479

The University of British Columbia

12 April, 2026

Executive Summary

This project established a cost-effective, micro-scale platform for multi-agent reinforcement learning in autonomous driving, addressing the high cost and safety constraints of full-scale real-world testing. The primary goals were to implement a multi-agent platform, build a track environment with an overhead camera that in real time detects and tracks 1:64 scale RC cars, and to create a low-latency control architecture to validate simulation-trained policies. Using the Proximal Policy Optimization algorithm, agents were first trained in an NVIDIA Isaac Lab digital twin for core tasks like line following and obstacle avoidance. The system successfully deployed these policies to the physical world via a low-latency ESP-NOW communication architecture and ArUco-based localization. The main outcome is a complete hardware and software pipeline that successfully generated and fine-tuned driving policies, proving the viability of this platform for rapid, economical research into simulation-to-real transfer, and accelerating the development of autonomous driving agents.

Table of Contents

Table of Figures

1 Introduction

This project focuses on developing a multi-agent reinforcement learning platform for autonomous driving using 1:64-scale micro RC cars. By creating a physical "playground" integrated with an overhead vision system and digital simulation, the system provides a low-stakes environment to implement and test complex Reinforcement Learning (RL) algorithms.

1.1 Sponsor Context and Motivation

The project is sponsored by Wayve Technologies Ltd., a company that builds and trains autonomous driving agents. A key motivator for the sponsors was the research paper "Robust Autonomy Emerges from Self-Play" (Cusumano-Towner et al., 2025). Testing trained models in full-scale real-world environments is expensive, time-consuming, and heavily constrained by safety regulations and maintenance requirements. Wayve sought a miniature testing suite to experiment with sim-to-real transfer and validate simulation-trained policies in a physical environment much quicker and more cost-effectively than using full-sized vehicles.

1.2 Background and Significance

Reinforcement Learning allows agents to learn optimal driving habits through observations and rewards. While simulation is valuable for rapid testing and vectorization, real-world deployment introduces unmodeled dynamics such as tire slip, motor nonlinearity, and sensor noise (e.g., motion blur and lighting variations). Developing a successful sim-to-real transition is essential to scale these models safely and reduce reliance on extensive full-scale real-world data collection.

1.3 Problem Statement and Project Objectives

Training driving agents directly in real-life settings is costly and impractical for high-stakes testing. The core problem is creating a reliable sim-to-real training platform that can bridge the performance gap between perfect simulation physics and the "noise" of reality.

Project Objectives:

- Implement a multi-agent autonomous driving platform powered by RL.
- Build a physical 1:64 scale model environment (approx. 2.5m x 2.5m) capable of hosting multiple vehicles.
- Develop an overhead computer vision system for real-time state tracking of all vehicles.
- Create a scalable infrastructure to remotely control multiple micro RC cars simultaneously with low latency.
- Successfully train agents in simulation for tasks such as line following, obstacle avoidance, and goal-oriented positioning for real-world validation.

1.4 Scope and Limitations

The project scope encompasses the entire pipeline from system architecture design to physical deployment, including hardware modification of RC cars, digital twin creation in Isaac Lab, and real-world policy inference.

Limitations:

- Physical Space: The system is constrained to a 2.5m x 2.5m track area.
- Hardware: Agents are limited by the physical capabilities of 1:64 scale RC cars (e.g., turning radius and battery life).
- Sensors: Localization relies entirely on a centralized overhead camera rather than on-board sensors.

2 Reinforcement Learning

2.1 What is RL?

Reinforcement learning is a paradigm of machine learning in which an agent learns to make sequential decisions by interacting with an environment without a ground truth. At each timestep, the agent observes a state, selects an action

according to its current policy, and receives a scalar reward signal. The objective is to learn a policy that maximizes the expected cumulative discounted reward over time. This framework is grounded in the theory of Markov Decision Processes, which assumes that the transition dynamics and reward function depend only on the current state and action, not on prior history. This is known as the Markov property.

RL algorithms broadly divide into model-free and model-based approaches. Model-free methods, such as Q-learning and policy gradient algorithms, learn a value function or policy directly from sampled experience without explicitly representing the environment's transition dynamics. Model-based methods, by contrast, learn or exploit a world model to plan ahead, offering greater sample efficiency at the cost of model bias. Further distinction exists between value-based methods, which derive policies implicitly by estimating action-value functions, and policy-based methods, which directly parameterize and optimize the policy, often via gradient ascent on the expected return.

2.2 Proximal Policy Optimization

Proximal Policy Optimization (PPO) is one of the most widely adopted policy gradient algorithms in modern RL, favored for its strong empirical performance, relative simplicity, and stable training behavior. Standard policy gradient methods suffer from a fundamental instability: large parameter updates can drastically change the policy, leading to performance collapse that is difficult to recover from. PPO addresses this by constraining how much the policy is allowed to change at each update step. It does so by clipping the probability ratio between the new and old policy within a narrow band, effectively preventing any single gradient step from moving the policy too far. This removes the need for the more complex constraint machinery of its predecessor, Trust Region Policy Optimization (TRPO), while achieving comparable or better results in practice.

In implementation, PPO is an actor-critic algorithm that collects a fixed number of environment interactions using the current policy, then performs multiple epochs of minibatch gradient updates over that experience before

discarding it, making it on-policy. The clipped surrogate objective encourages the policy to improve on advantageous actions while ignoring updates that would push the probability ratio outside the clipped range, yielding a pessimistic lower bound on policy improvement. Advantage estimates are computed using Generalized Advantage Estimation, which interpolates between high-bias/low-variance and low-bias/high-variance return estimates via a parameter lambda.

PPO was the chosen RL algorithm for this project due to robustness to hyperparameter choices and stable learning. While PPO was used, many other algorithms could be explored, particularly in the case of multi-agent training. One interesting direction worth investigating is self-play, where the agent competes with different versions of itself. In general, other approaches were not heavily explored up to this point, and is absolutely worth looking into.

2.3 Using RL to Train Agents

The observations of our agent come from an overhead camera. Generally, this includes a picture of the track aligned to the frame of the agent, in addition to a velocity vector. These provide our agent with a context of its own position and trajectory in the environment. The rewards are designed to teach good driving behaviours. Typically, rewards are given for staying close to the line and driving at high speeds. The actions are a steering angle and a throttle command. Specific observations and rewards are task-dependent and are discussed in further detail in Section 5.

3 Simulations

All training tasks are developed and validated in NVIDIA Isaac Lab before deployment on the physical platform. The simulation environment serves as a digital twin of the physical system, replicating the track geometry, vehicle dynamics, and overhead camera perspective to enable rapid policy iteration at a fraction of the cost of real-world training.

3.1 Simulation in Isaac Lab

The environment models an Ackermann-steered vehicle on a flat track with an overhead camera. The vehicle's four-joint articulation is abstracted to a two-dimensional action interface (throttle and steering) matching the control scheme from our real environment. The virtual camera is configured to replicate the physical camera's optical parameters and produces 512 x 384 pixel frames, which is ¼ of the actual camera's resolution. This choice was made to save memory usage. The camera frames are then processed into cropped and binarized agent-aligned observations. A coordinate conversion utility ensures consistent mapping between the simulation world frame and image pixel space across both domains. This conversion is necessary to ensure that all information is passed. Figure 3.1 shows the simulated environment.

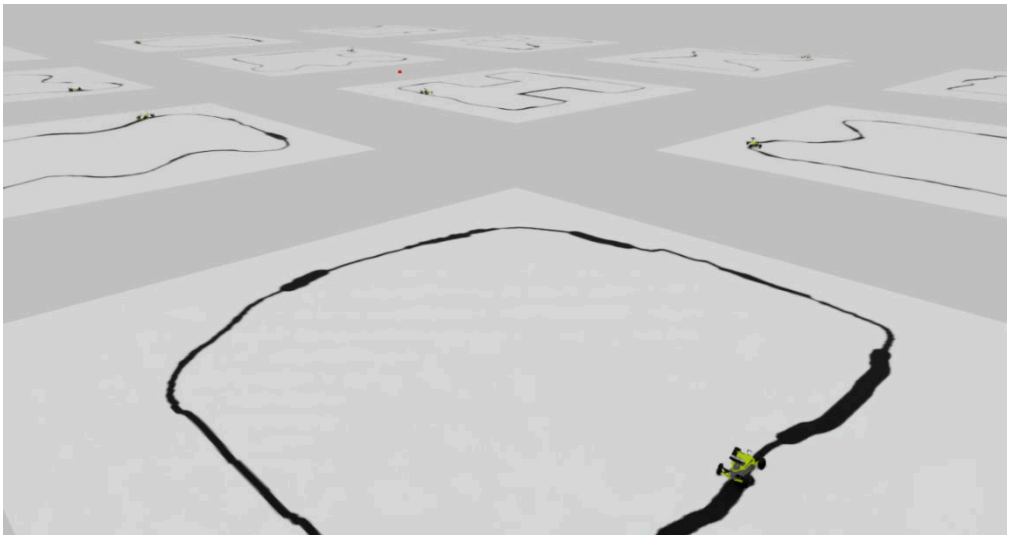

Figure 3.1: Isaac Lab environment with parallel tracks and the Ackermann vehicle model.

3.2 Parallelization

Training runs up to 12 environments simultaneously, each assigned a different track layout from a library of 12 variants. This diversity was introduced to avoid overfit in policies trained on a single track. To scale further without GPU memory bottlenecks from camera rendering, static track images are captured once at

initialization and cached as tensors, eliminating per-step rendering overhead in subsequent runs. All observations, rewards, actions, and termination functions used in the training loop are vectorized, in order to eliminate CPU usage as much as possible and greatly speed up training.

Training uses Stable Baselines 3 (SB3) PPO and converges in approximately 15 minutes for line following on a single NVIDIA RTX 5070. The simulation control period (100 ms) approximates the physical system's 60 Hz update rate. The control period is higher than that of the real track setup, which is about 20 ms; however, running at a higher control period allows for frequent and rapid training runs to evaluate the effectiveness of reward and observation structure.

3.3 Sim-to-Real Considerations

The simulation deliberately omits hard-to-model dynamics such as tire slip, motor nonlinearity, battery voltage sag, and actuator latency. Zero-shot transfers showed that simulation-trained policies capture basic driving competence but degrade under these unmodeled effects. Rather than pursuing high-fidelity system identification, the platform supports continued PPO training on real-world rollouts using the simulation policy as a warm start, as described in Section 6.

4 System Hardware and Software

The physical platform is composed of three subsystems: the vehicle agents, the wireless communication architecture, and the track environment. Figure 4.1 provides a system-level overview of how these subsystems interact with each other as well as the simulation during a training loop.

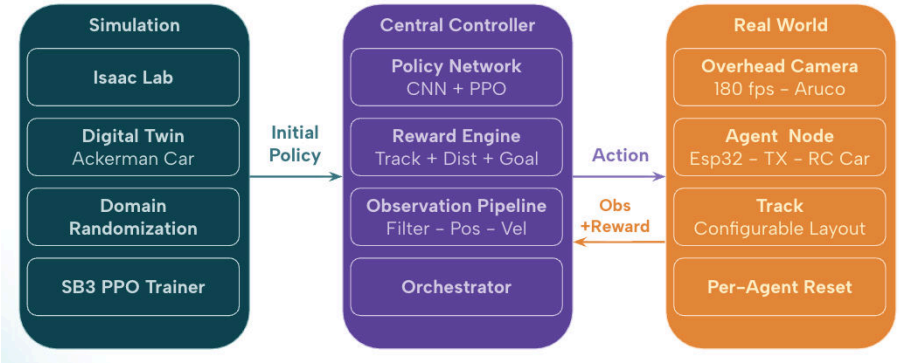

Figure 4.1: System-level diagram

4.1 Vehicle Platform

The project uses commercially available TTR Nano 1:64-scale RC cars rather than custom-built vehicles, prioritizing reproducibility and development speed over full mechanical customization. Using commercial hardware required reverse-engineering the control interface. Each car's receiver accepts analog voltage signals for steering and throttle. To enable gain digital control, the stock transmitter's potentiometers were replaced with Digital-to-Analog Converter (DAC) outputs from an ESP32 microcontroller, which writes 8-bit voltage values directly to the control lines. A custom PCB and a mount were designed for the updated controllers so that they would all be housed together, allowing 3-5 car controllers to each be connected to their own ESP32. Since each ESP32 only contains two DACs, each controller requires its own dedicated ESP. This modular approach made it easy to connect the controller PCBs: we disassembled the stock transmitters, added JST connectors to the wires originally connected to the potentiometers and throttle, and plugged them directly into the ESP-based PCB interface.

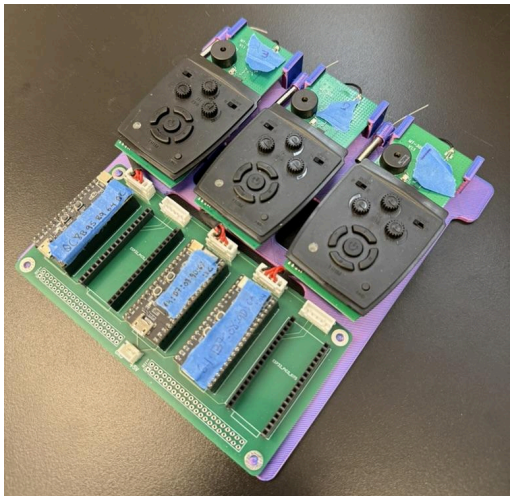

Figure 4.2: Modified control unit capable of controlling three TTR Nano cars.

The stock batteries provided approximately 40 minutes of runtime. To ensure longer uptimes during training sessions, these were upgraded with higher-capacity cells. This upgrade shifted the vehicle's center of mass upward, which increased the risk of rollovers at high speeds on high-grip surfaces. To mitigate this effect, a software-level throttle limit was implemented during the training process. Figure 4.3 illustrates the revised chassis design, which accommodates the larger battery and features a 3D-printed mount for the ArUco marker on the top surface.

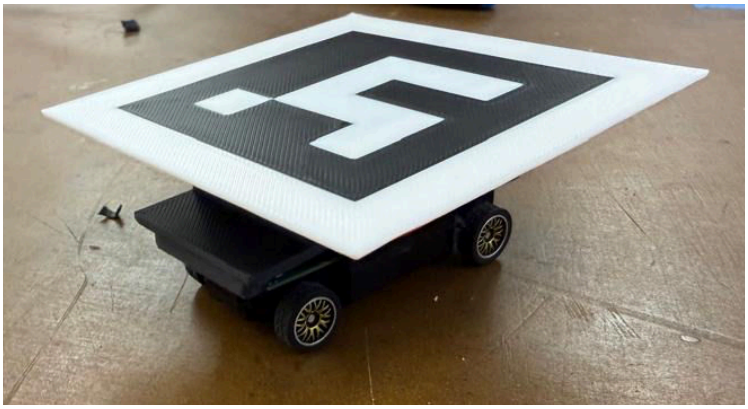

Figure 4.3: Upgraded TTR Nano with a bigger battery and mounted AruCo

4.2 Communication Architecture

The communication architecture evolved through two major iterations, primarily driven by a bandwidth target of 60 Hz.

The initial implementation used MQTT over Wi-Fi with a Mosquitto broker. While MQTT offered convenient topic-based multi-device addressing, round-trip latency measured approximately 60 ms under ideal conditions and degraded to 200–500 ms during PS4 controller teleoperation testing.

The final architecture replaced MQTT with ESP-NOW, which transmits directly between radios using MAC-level addressing, bypassing both the TCP stack and the access point.

Multi-agent control is achieved through a serial-to-ESP-NOW bridge: the host computer sends car-ID-prefixed commands over USB serial to a master ESP32, which routes each command via ESP-NOW to the correct receiver. This architecture was validated with three simultaneous vehicles at 60 Hz with no cross-talk or perceptible lag.

4.3 Track Environment

The track is a 2.5 x 2.5 m modular surface with reconfigurable lane markings painted on a white projector surface. An overhead FLIR camera provides fiducial tracking at up to 118 fps. The modular layout supports different training scenarios, including line following, obstacle avoidance, and multi-agent interaction tasks. The track also includes boundary walls to contain the agents, which was necessary for allowing for training without human intervention. An agent that was contained by the boundary is simply reset using the reset agent, which is further discussed in section 5.4.

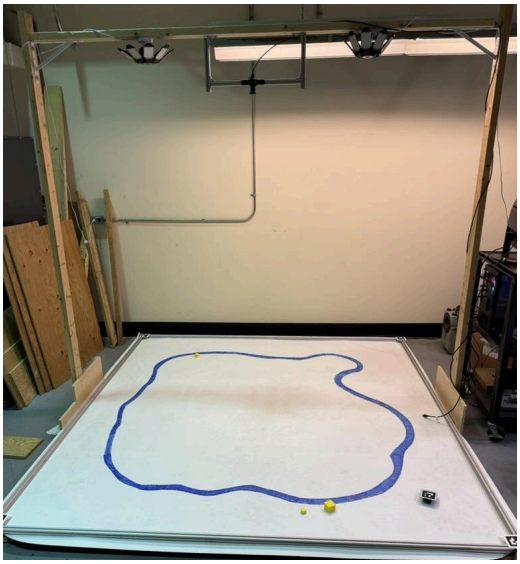

Figure 4.4: Built track with overhead camera and lights.

4.4 Computer Vision and Tracking

As discussed in the introduction, the successful transfer to the real, physical world and implementation of the tasks for which agents were trained in simulation require a modular and adaptable computer vision system and pipeline for real-time state tracking of all vehicles, objects, and agents within the real environment.

The computer vision real-time tracking system was developed with the following key considerations and specifications.

- Modularity
- Varsitility
- Low Latency

4.4.1 Camera Selection

Given that the tracking system in our project would be responsible for the capture of high-speed moving agents with low latency and high responsiveness, the camera must have a high enough frame rate and resolution. The speed of an RC car was measured to be approximately 55 cm/s and was used to calculate the

approximate change in pixels per frame, given the camera specifications for a series of models. The selected camera is a 118 fps Flir BFS-U3-32S4C.

4.4.2 Localization and Tracking

The system uses a set of 2D fiducial markers for localization of the environment and tracking agents while using HSV value filters for tracking coloured obstacles and the road drawn on the track. To determine the appropriate fiducial for tracking the agent, the latency and efficacy performance of several fiducials, including AprilTag, ChromoTag, and ArUco, were explored. Based on the requirement for consistent and reliable detection of moving agents and the need for low latency in the tracking system, ArUco tags were chosen, as seen in Figure 4.5.

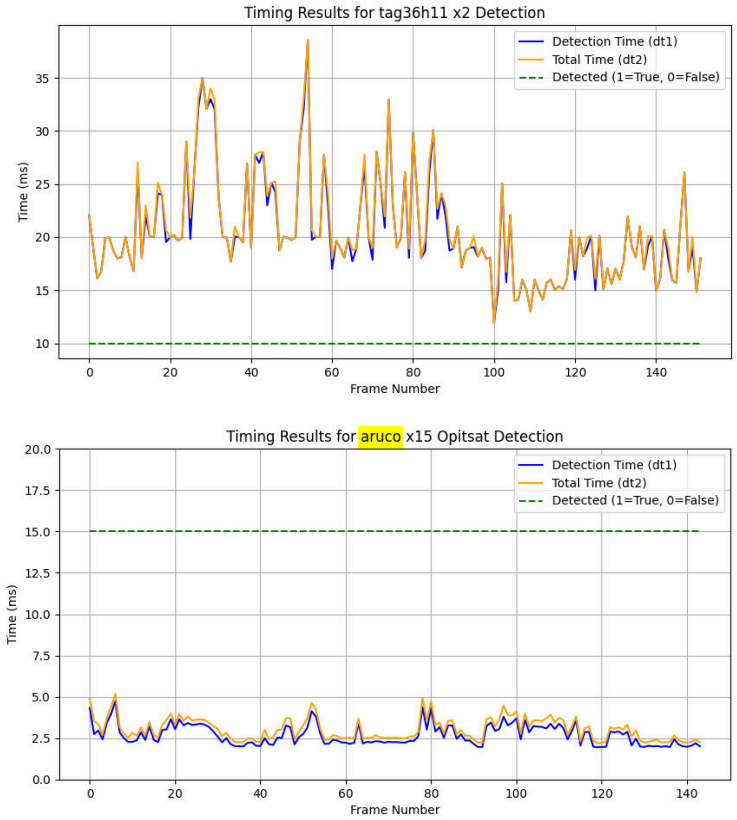

Figure 4.5: Latency and detection results for 2x AprilTags running on Lenovo Yoga and 15x ArUco tags running on Project Lab Optisat desktop.

4.4.3 Software Pipeline

The software system consists of multiple classes that together localize features in the environment and track agents using a set of ArUco tags and object HSV values. The following classes were developed for modularity.

- FiducialDetector: This class initiates and finds all fiducials within the given frame
- HSV Filter: Uses HSV values for segmentation
- Track: Localizes the environment boundaries using fiducials and segments the track using HSV values of track colours.
- Agent: Localizes and tracks agent position, speed, and orientation within the environment.
- Obstacle: Segmented based on HSV values.
- RealObserver: Main class that extracts real observations using the above classes.

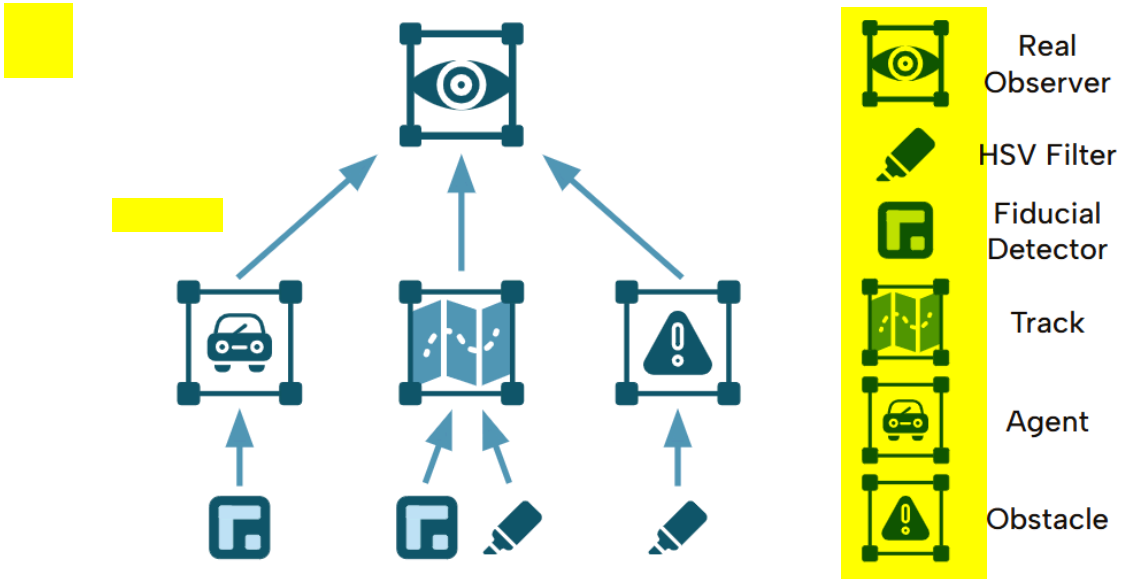

Figure 4.6: Computer vision pipeline for extracting observations and features for inference and training tasks in the real environment.

The tracking pipeline consists of the RealObserver initiating a Track instance and forming the boundaries of the physical environment through the fiducials placed at the four corners of the environment. Then one or many instances of Agent are initiated, which track the agents' position, orientation, and speed in real time. Obstacles can be added and localized using their colour HSV values. The RealObserver puts together information from these objects and returns requested observations depending on the training strategy and the agent's RL model inputs, as will be discussed.

5 Training Tasks

The platform supports a progression of training tasks of increasing complexity: line following, obstacle avoidance, and multi-agent driving. Each task is first developed and validated in the Isaac Lab simulation environment before deployment on the physical platform. A dedicated position reset agent enables autonomous episode resets in the real world, which is a prerequisite for uninterrupted real-world training.

5.1 Line Following

Line following is the foundational training task, where a single agent learns to track a path on the track surface. The agent receives a 48 × 48 pixel overhead image observation, cropped from the original 512 x 384 px camera image and oriented relative to its own position and heading, along with scalar velocity and orientation readings. The policy network processes the image through three convolutional layers before concatenating with the scalar observations and passing through fully connected layers to produce steering and throttle commands.

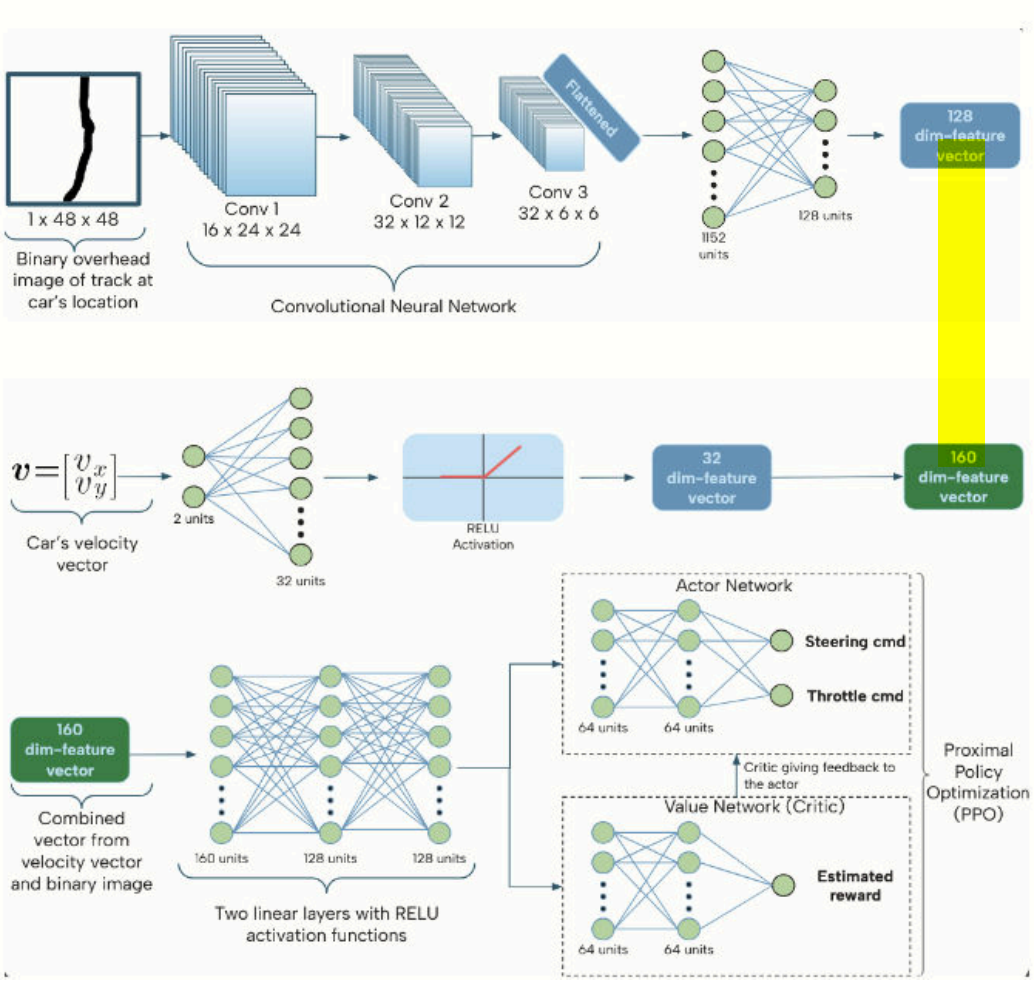

Figure 5.1: A CNN extracts features from the track image, which are combined with the vehicle's velocity and passed through shared layers. An actor–critic structure then outputs steering and throttle commands, trained using PPO.

The reward function combines four terms:

1. Distance-from-line penalty that encourages the agent to stay centered on the track,
2. Velocity reward that promotes forward progress
3. Alive bonus for remaining on the track,
4. Steering jerk penalty that discourages erratic control inputs. The jerk penalty is simply a steering penalty normalized by the max steer angle. This encourages

the agent to make steering action only when necessary, hence removing any jerk from steering.

Training runs in simulation with 12 parallel environments and converges in approximately 15 minutes. Figure 5.2 shows a trained agent following the line in simulation.

Figure 5.2: Screenshot of the line-following agent in Isaac Lab, with the agent-oriented observation overlay.

5.2 Obstacle Avoidance

Obstacle avoidance extends the line-following task by introducing static obstacles on the track. The observation space adds a second image channel encoding obstacle locations, giving the agent a two-channel input: track and obstacles. The reward function retains the line-following rewards, and the only change is that the episode terminates as the agent hits the obstacle. The agent learns to deviate from the optimal line to avoid obstacles and return to the path afterward. Figure 5.3 shows the two-channel observation and a trained avoidance maneuver.

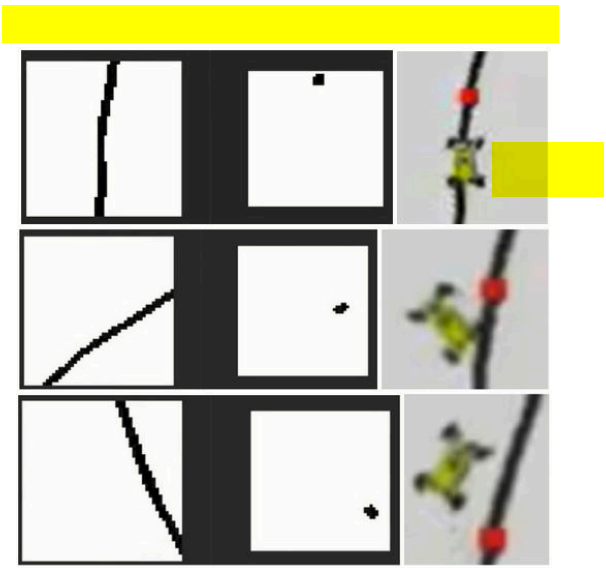

Figure 5.3: Agent-oriented observation showing track channel and obstacle channel side by side, plus a simulation screenshot of the agent navigating around an obstacle. Obstacles are shown in red. Three snapshots are shown in chronological order from top to bottom.

5.3 Multi-Agent Driving

Multi-agent training introduces additional vehicles into the environment. Each agent receives observations where other cars appear in the obstacle channel, and frame stacking encodes the velocity of neighboring agents. While there are multiple agents on the track, all of their experiences are used to train a single model. This means that the model is forced to adapt its weights and biases to more complex scenarios where two or more cars are on a head-on collision course, or a car is stuck behind another car and might have to overtake.

In this multi-agent scenario, the reward structure is similar to the obstacle avoidance task, with the addition of a penalty in case of collisions. The termination is slightly more complex and needs extra attention to avoid spawning terminated agents close to other agents that are still collecting data. To do this, the termination function spawns the agents randomly at a location on the track while ensuring that they are within some safe neighborhood of the other agents.

Interesting behaviours emerge from this multi-agent setup. The agents automatically learn that they can maximize the reward by driving on one side of the

line, which allows them to avoid head-on collisions. Furthermore, in the case of two agents driving back-to-back, the one at the front tries to speed up while the one at the back slows down.

5.4 Reset Agent

In simulation, resetting an agent to a spawn location is trivial, as the vehicle is simply teleported. In the physical system, the car must drive itself back to a valid starting position after each episode termination. The reset agent addresses this by training a policy that navigates to a specified goal position and orientation from anywhere on the track.

Unlike the vision-based driving tasks, the reset agent uses a compact vector observation: the displacement vector to the goal, the vector to the nearest wall, the heading error relative to the goal orientation, and the current speed. Figure 5.4 illustrates the reset agent's observation space.

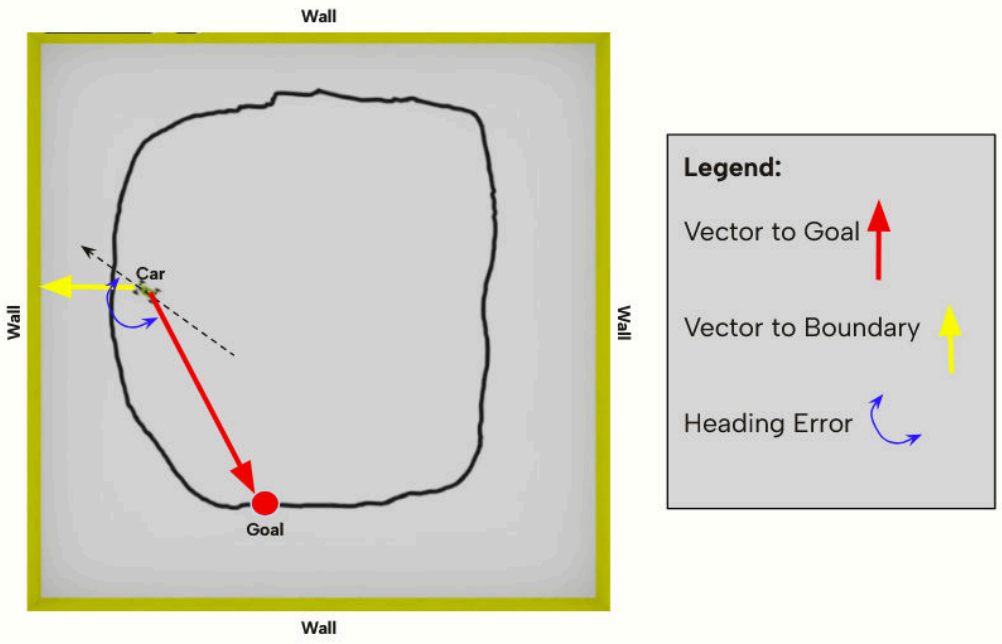

Figure 5.4: Diagram showing the reset agent's vector observation — Vector to goal point and to closest wall point, and heading error in the car's local frame. In addition, the car speed is part of the observation (not shown above)

The reward function for this training task encourages reducing the distance to the goal and provides a bonus upon arrival. The bonus is only given if the car stays at the goal for some set number of steps. This ensures that the model learns to stop at the goal and not just simply arrive at it. The velocity reward is also defined piecewise to encourage the same kind of behaviour. When the agent is outside some radius of the goal, a higher speed is encouraged, whereas inside the radius, a lower speed is encouraged.

Upon termination, which can happen either due to a timeout or the agent achieving the objective, the robot is spawned 50% of the time on a random location on the track and 50% of the time at a location close and facing to the boundary. This choice of resetting allows the agent to learn to also drive backward when necessary. The new goal point is also chosen randomly at the beginning of a new episode.

Please refer to Appendix B to view the training results of the reset agent model.

6 Training in Real

Trained simulation policies are deployed to the physical platform via zero-shot transfer. This transfer works well in some cases, but unmodeled dynamics such as tire slip, battery voltage sag, and actuator latency degrade zero-shot performance. Rather than attempting to close the sim-to-real gap entirely through simulation improvements, the platform supports continued PPO training on real-world rollouts. When an episode terminates, the reset agent autonomously returns the car to a valid position, enabling continuous training without human intervention.

To ensure proper transfer from simulation to the real track, we ensure that the core components of the training task are transferred properly. The two main components are observation and action space.

6.1 Observation Space Matching

At each time step, the RealObserver object discussed in Sec 4.4.2 localizes the Agents and crops the track and obstacle channels around the position of each Agent according to an input field of view (FOV) parameter that determines the proportion of this cropped image to the entire track frame. Using the orientation of the Agents, the cropped images are rotated to match the frame of reference of each of the Agents independently. The results can be seen in Figure 6.1.1.

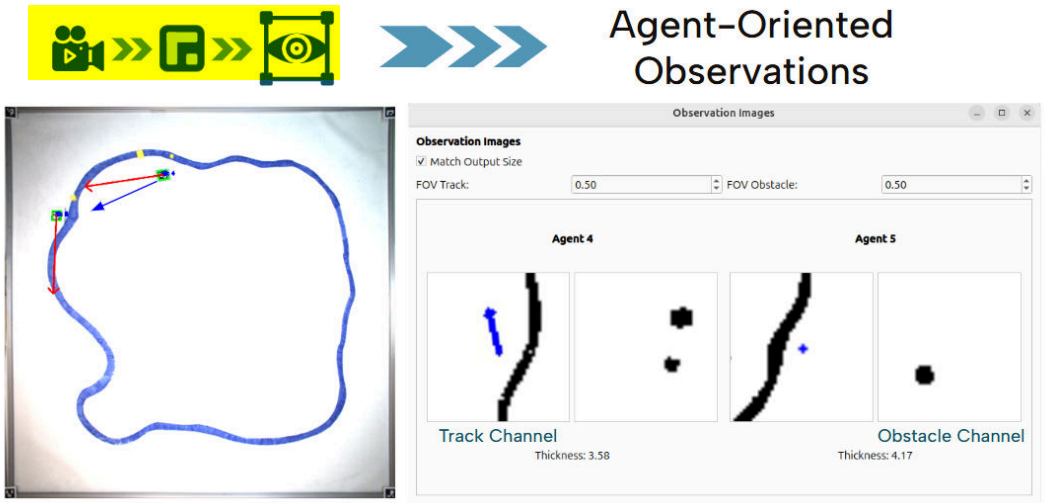

Figure 6.1.1: Sample Outputs of the computer vision pipeline.

The car's velocity vector was normalized in both simulation and real-world settings by dividing it by its maximum possible magnitude. In the real system, the raw velocity was computed from the pixel displacement between consecutive frames. Since both representations are ultimately normalized by their respective maximum magnitudes, the transition between simulation and real-world inputs remains consistent.

6.2 Action Space Matching

In simulation, the vehicle joints are configured to receive position-based commands for steering (in radians) and velocity-based commands for throttle (in radians per second). Conversely, the physical system requires 8-bit integer values

(0–255) transmitted to the ESP32, where the midpoint corresponds to a neutral steering angle and zero throttle.

To allow for easy transfer between sim to real, the model's output passes through a tanh function, which ensures that its outputs are bounded to -1 and 1

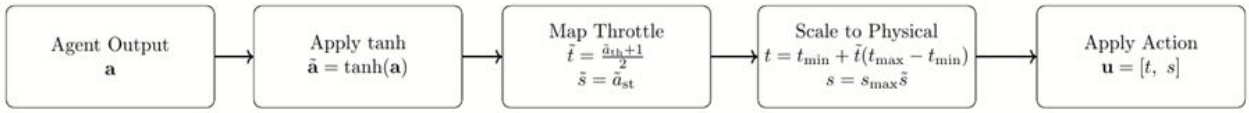

Figure 6.1: Flowchart showing models output normalization and mapping — The models' output after passing through tanh can be easily mapped to any range with some max and min value.

By doing this, when the model is running in simulation, the max and min values of the simulation must be used, and when taken to real, by just changing the max and min values of steering and throttle, the model can output the actions that correspond to the same dynamics in the simulation.

6.3 Zero-Shot and Fine-Tuning

We used our simulation-trained model in the real environment both in zero-shot transfer (no additional training) and as a starting point for fine-tuning with further reinforcement learning. Zero-shot transfer of the line-following model showed adequate driving performance, able to consistently follow the track. While able to complete the basic task, the agent struggled with oscillations and at higher speeds, could not follow the line as closely as desired. The need for fine-tuning stems from the differences in both observations and physics between our real track and simulated environments.

During fine-tuning, the line follower model was trained for about 3-4 hours on the real track to achieve better performance. The learning rate was lowered from 0.0003 to 0.0001 to avoid excessive updates on the model that would make it too different from the original simulation version. The fine-tuning was performed for about 170000 steps, and the reset agent model was also used to return the car back

to the track. The performance of the line follower was significantly enhanced after fine-tuning.

7 Conclusion

The objective of establishing a cost-effective, micro-scale platform for multi-agent reinforcement learning in autonomous driving was successfully achieved, providing a novel environment for real-world validation and sim-to-real transfer. The project delivered a complete hardware and software pipeline, including a 2.5m x 2.5m modular track environment and modified 1:64 scale RC vehicles capable of precise digital control. A key technical success was the implementation of a scalable, low-latency ESP-NOW communication architecture that enabled simultaneous command transmission to multiple vehicles at a stable 60 Hz rate. This, combined with the real-time ArUco-based computer vision localization pipeline and an Ackermann vehicle digital twin in Isaac Lab, enables a complete RL training and deployment loop. The platform successfully generated and deployed policies for core training tasks such as line-following, obstacle-avoidance, and multi-agent scenarios using the SB3 PPO algorithm. By bridging the gap between perfect simulation physics and the noise of reality, this robust system offers sponsors Wayve Technologies Ltd. a rapid and economical testing suite, fulfilling the project's core motivation to accelerate the development of autonomous driving agents.

8 Recommendations

Future work should prioritize the development of a high-fidelity URDF model for the 1:64 scale vehicles to enhance sim-to-real transfer accuracy. Additionally, further fine-tuning of obstacle avoidance policies and the validation of multi-agent capabilities in the physical track environment (which is currently limited to simulation) are essential next steps. With the complete hardware and software infrastructure now established, the platform is fully prepared for rigorous, research-focused exploration into complex multi-agent reinforcement learning behaviors.

9 Deliverables

1. **Multi-agent RC car platform**: modified TTR Nano vehicles with custom PCB, DAC control interface, 3D-printed ArUco/battery mounts, and TP4056 charging station
2. **ESP-NOW communication system**: ESP32 firmware (master bridge + receivers) and Python control library supporting teleop & simultaneous multi-car command at 60 Hz
3. **Modular track environment**: 2.5 x 2.5 m reconfigurable track with overhead FLIR camera and boundary walls
4. **Computer vision pipeline**: real-time ArUco-based localization, track segmentation, and agent-oriented observation generation
5. **Isaac Lab digital twin**: simulation environment with Ackermann vehicle model, overhead camera used to train simulation model behavior.
6. **RL training and deployment pipeline**: SB3 PPO with CNN policy, trained policies for line following, obstacle avoidance, position reset, and multi-agent scenarios, plus a real-world training loop for in-real policy fine-tuning.
7. **User manual and documentation**: setup guide (See appendix A), GitHub repository, and project logbooks, including team and individual logbooks.

References

Cusumano-Towner, M., Hafner, D., Hertzberg, A., Huval, B., Petrenko, A., Vinitsky, E., Wijmans, E., Killian, T., Bowers, S., Sener, O., Krähenbühl, P., & Koltun, V. (2025). *Robust Autonomy Emerges from Self-Play* (arXiv:2502.03349). arXiv. https://doi.org/10.48550/arXiv.2502.03349

Appendix

Appendix A - Project Manual

In addition to this report, we have prepared a manual for our project where we dive deeper into technical details. The manual goes more in-depth than this report in terms of the overall operation of the system and specific choices and decisions made in not only software, but electrical and mechanical aspects of the project. The manual can be accessed here.

Appendix B - Training Metrics for Reset Agent

The reset agent, trained with the SB3 PPO algorithm, demonstrates stable and rapid convergence, achieving optimal performance within approximately one minute of training across 1,000 parallel environments. This performance was realized through refinements to the observation space and the application of conditional rewards that incentivize high-speed navigation when distant from the target, coupled with precise deceleration upon arrival. Furthermore, the integration of generalized State-Dependent Exploration (gSDE) and a dedicated turning penalty proved instrumental in establishing smooth trajectories and robust goal-oriented behavior.

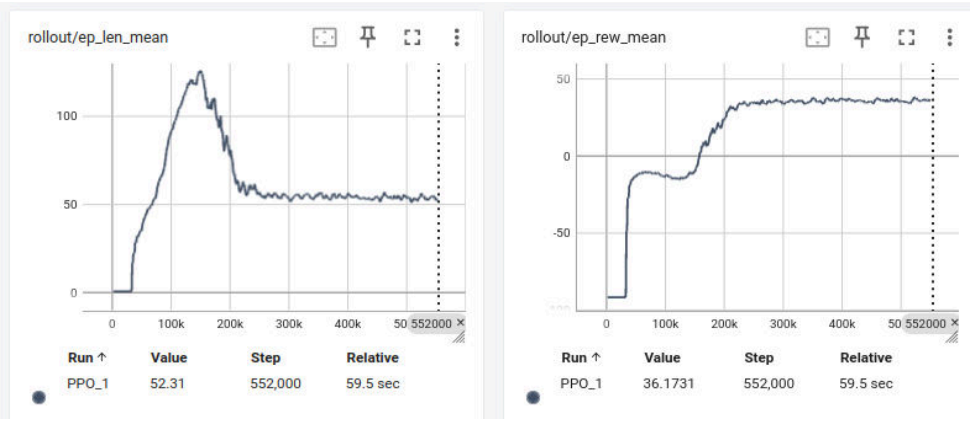

Figure B.1: Training curves for the PPO-trained Reset Agent, illustrating the stable convergence of the mean episode reward (Right) and the corresponding mean episode length (Left). The x axis is the number of steps.

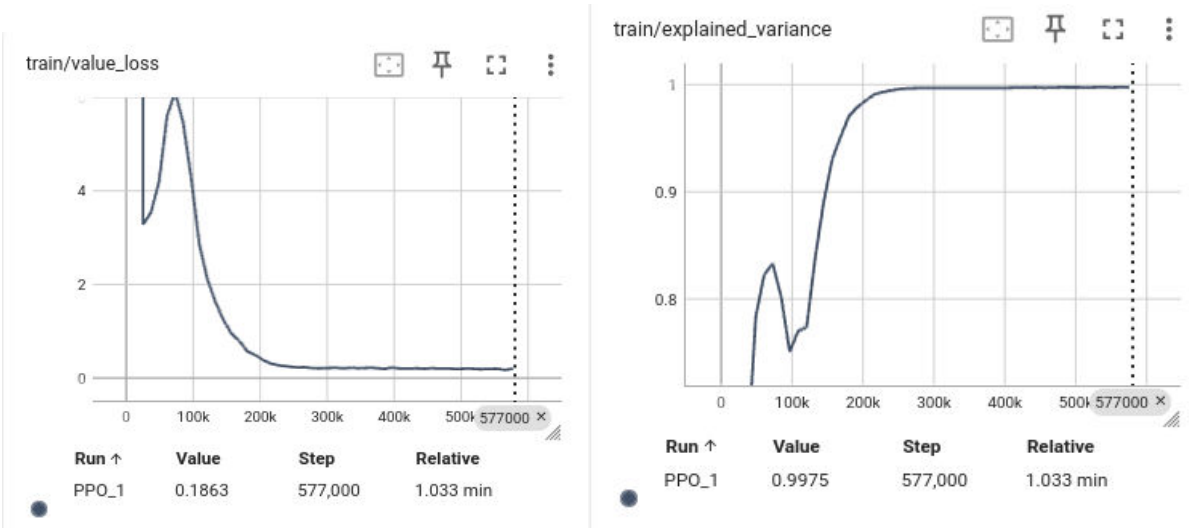

Figure B.2: Value Loss (Left) demonstrates stable convergence, and Explained Variance (Right) approaching 1.0 confirms the Critic network's high-fidelity prediction of expected returns. The x-axis is the number of steps