Deep Reinforcement Learning Agents for Robust Decision Making in Automated Driving

Università di Genova

Alessandro Pighetti

Supervisors: Prof. Riccardo Berta
Prof. Francesco Bellotti
Prof. Changjae Oh
Prof. Andrea Cavallaro

Department of Electrical, Electronic, Telecommunications Engineering and Naval Architecture (DITEN)
University of Genoa

This dissertation is submitted for the degree of
Doctor of Philosophy

Joint Doctorate in Interactive and
Cognitive Environments - Cycle 38

May 2026

Deep Reinforcement Learning Agents for Robust Decision Making in Automated Driving

Alessandro Pighetti

Joint Doctorate in Interactive and Cognitive Environments
JD-ICE

XXXVIII cycle

Acknowledgements

This PhD Thesis has been developed in the framework of, and according to, the rules of the Joint Doctorate in Interactive and Cognitive Environments JD-ICE with the cooperation of the following Universities:

Università degli Studi di Genova (UNIGE)

DITEN - Dept. of Electrical, Electronic, Telecommunications Engineering and Naval Architecture

ISIP40 - Information and Signal Processing for Cognitive Telecommunications

Primary Supervisor: Prof. Riccardo Berta
Secondary Supervisor: Prof. Francesco Bellotti

UNIVERSITÀ DEGLI STUDI
DI GENOVA

Queen Mary University of London (QMUL)

EECS - School of Electronic Engineering and Computer Science

CSI - Centre for Intelligent Sensing

Primary Supervisor: Prof. Changjae Oh
Secondary Supervisor: Prof. Andrea Cavallaro

Abstract

Robust decision making is a central bottleneck for automated driving (AD). Even rare policy failures can translate into safety-critical outcomes, and therefore learning-based approaches must deliver not only high nominal performance but also stability and reliability under perturbations. Deep reinforcement learning (DRL) provides a principled framework to learn driving policies from interaction, yet state-of-the-art methods still face recurring gaps in practice. They are often brittle under distribution shift, hard to train reproducibly in safety-constrained tasks, and vulnerable to atypical or adversarial behaviors by other agents. This thesis advances the state of the art by introducing and validating a coherent training stack for robust DRL-based decision making that treats stability and safety as design objectives. The core claim is that robustness emerges from structured problem formulation, training processes that control difficulty over time, systematic evaluation under operational design domain (ODD) shifts, and systematic exposure to failure modes through adversarial interactions. Concretely, the proposed pipeline integrates maneuver-level action abstraction, curriculum learning (CL), scenario diversification and benchmarking for ODD coverage, and adversarial fine-tuning with explicit realism constraints. Across highway driving and automated parking (AP), results show that posing the decision problem at the maneuver level reduces unsafe exploration, improves learning efficiency, and yields more interpretable policies than flat low-level control. In low-speed maneuvering, staged curricula mitigate common local minima and enable stable convergence, while a transfer study in CARLA confirms that the training logic generalizes to higher-fidelity simulation when timing and perception are tuned appropriately. To quantify generalization beyond single scenarios, the thesis introduces a unified CARLA parking benchmark that exposes the performance impact of operational design domain (ODD) shifts across perpendicular, skewed, and parallel geometries and supports targeted fine-tuning for ODD expansion. For robustness under strategic interactions, the thesis studies attacker–victim training in AP and proposes Regularized Adversarial Fine-Tuning (RAFT) for CARLA parking. RAFT addresses a key failure mode of naive zero-sum formulations, namely degenerate collision-seeking or region-blocking opponents, by regularizing the adversarial objective to favor challenging yet feasible interactions. Empirically, RAFT preserves baseline success in the

static parking ODD while improving trajectory-quality indicators such as alignment error and reverse actions, without catastrophic forgetting. Taken together, the presented methods and the accompanying environments, interfaces, and benchmarks provide a practical basis to train, evaluate, and harden DRL policies for AD beyond nominal conditions. They also make it easier for other researchers to reproduce results and to build on this work, including automated curriculum design, multi-agent robustness with data-driven behavior priors, tighter integration with safety guarantees and verification, and sim-to-real validation.

Table of contents

List of figures

List of tables

Nomenclature

Roman Symbols

$\mathscr{A}$	Set of actions
a	Longitudinal acceleration command
$A^{\pi}(s,a)$	Advantage function
β	Slip angle
d	LiDAR ray distances
δ	Steering angle
ε	Clipping coefficient (PPO)
γ	Discount factor
G_t	Discounted return at time step t
l	Wheelbase
$L_{\mathrm{CLIP}}(\theta)$	PPO clipped surrogate objective
n	Number of LiDAR rays
π	Policy
ψ	Heading angle
$Q^{\pi}(s,a)$	Action-value function
R	Reward / reward function (context-dependent)
$r_t(\theta)$	Probability ratio (new vs. old policy)

$\mathscr{S}$	Set of states
s	Stacked frames / stacking index
$\mathscr{T}$	State-transition probability function
v	Vehicle speed
$V^{\pi}(s)$	State-value function
x	Vehicle longitudinal position
y	Vehicle lateral position

Subscripts

coll	Collision-related term
hs	High-speed term
max	Upper bound of a range
min	Lower bound of a range
old	Old policy (PPO)
rml	Right-most-lane term
r	Collision penalty component (RAFT)
rs	Regularization term (RAFT)
s	Blocking/obstruction penalty component (RAFT)
t	Time step

Acronyms / Abbreviations

ACC	Adaptive Cruise Control
AD	Automated Driving
ADAS	Advanced Driver Assistance Systems
AP	Automated Parking
API	Application Programming Interface

CARLA Car Learning to Act (simulator)

CL Curriculum Learning

DNN Deep Neural Network

DQN Deep Q-Network

DRL Deep Reinforcement Learning

EV Ego Vehicle

Gym OpenAI Gym / Gymnasium (RL interface)

LiDAR Light Detection and Ranging

MDP Markov Decision Process

MLP Multi-Layer Perceptron

MPC Model Predictive Control

NN Neural Network

NPV Non-Player Vehicle

PID Proportional–Integral–Derivative

PPO Proximal Policy Optimization

RAFT Reinforcement-learning Adversarial Framework (for Testing)

RF Reward Function

RL Reinforcement Learning

SAC Soft Actor–Critic

SB3 Stable-Baselines3

TRPO Trust Region Policy Optimization

Chapter 1

Introduction

1.1 Motivation

Automated driving (AD) represents one of the most transformative applications of artificial intelligence, with the potential to reduce accidents, alleviate congestion, and expand mobility access. The World Health Organization identifies traffic accidents as a leading cause of death and injury, with human error responsible for the majority of cases [92]. By mitigating human limitations such as distraction, fatigue, and impaired judgment, automation can directly contribute to safer and more efficient transportation [54, 82]. Yet, unlike other domains of machine learning where occasional failures are acceptable, failures in driving systems may have catastrophic consequences [27].

One of the central challenges of AD lies in robust decision making. While perception systems for object detection, lane recognition, and sensor fusion have advanced significantly, perception alone does not ensure safety [62, 15]. What ultimately matters is how an automated vehicle (AV) chooses to act under uncertainty, sensor noise, or adversarial disturbances [123, 75]. Decision making in traffic involves navigating multi-agent interactions where the intentions of others are uncertain, balancing safety, efficiency, and compliance with implicit social norms [120]. Traditional optimization-based planning approaches provide interpretability and guarantees in structured conditions but struggle to scale to the combinatorial complexity of real-world traffic [74].

Deep reinforcement learning (DRL) has emerged as a compelling alternative. By combining reinforcement learning (RL) [119] with deep neural networks (DNNs), DRL agents can directly map high-dimensional sensory observations to actions, learning through interaction with their environment. DRL has demonstrated strong performance in sequential decision problems such as Atari [84], Go [116], and robotic control [65, 71], and its adoption in AD

is steadily growing [39, 56]. However, translating these results from simulation benchmarks to safety-critical driving contexts exposes several persistent gaps.

Despite this progress, current DRL-based decision-making approaches still face major barriers when deployed in safety-critical driving contexts. In particular, reward maximization alone does not guarantee safe behavior, and policies often remain brittle when conditions deviate from those seen during training [5, 47, 30]. In addition, training can be unstable and difficult to reproduce, with outcomes that vary across random seeds, hyperparameters, and implementation choices [7]. Multi-agent interactions further amplify uncertainty, since the intentions and reactions of other road users cannot be fully controlled or predicted [121].

From a practical AD perspective, these limitations translate into open questions about how to develop policies that are both effective and verifiably reliable, how to evaluate them beyond nominal test cases, and how to harden them against disturbances and adversarial behaviors [100, 36]. The motivation of this thesis is therefore to treat robustness and policy stability in AD as primary development objectives. The research follows a staged trajectory, beginning with decision making in simplified driving environments, advancing toward complex parking in high-fidelity simulators, and culminating in adversarial learning frameworks for robustness evaluation and improvement. This structured approach enables both the generation of effective DRL-based decision-making agents and the development of methodologies for their testing, evaluation, and fine-tuning. The ultimate goal is to contribute to AD systems that are not only capable but also demonstrably robust in the face of uncertainty and adversarial conditions.

1.2 Research gaps and problem statement

AD is often conceptualized as a pipeline spanning perception, pre-processing, decision making, and control [40] (Fig. 1.1). Perception gathers raw data from sensors such as cameras, LiDAR, radar, or ultrasonic devices. Pre-processing transforms these signals into structured representations such as semantic maps or object lists [139]. Decision making constitutes the central reasoning module, translating processed perception into tactical or operational maneuvers [131]. Control then executes steering, throttle, and braking commands to realize the chosen actions [91].

From a learning perspective, the decision-making module operates under partial observability, multi-agent interactions, and uncertain dynamics. These properties motivate model-free DRL, but they also sharpen three gaps that this thesis explicitly targets.

First, standard objective design provides weak safety guarantees, and reward misspecification can induce unsafe or brittle strategies, which is particularly problematic in safety-critical

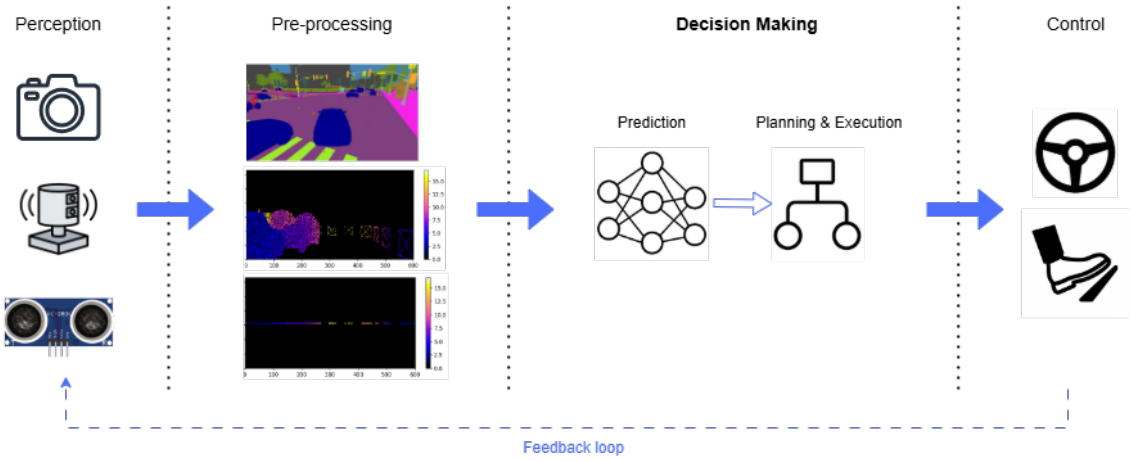


Fig. 1.1 Automated driving pipeline, simplified.

settings [5]. Second, policies often generalize poorly under distribution shift; for instance, changes in traffic density, road geometry, sensing noise, or weather can substantially degrade performance [47, 30]. Third, training stability and reproducibility remain open issues, with results that can vary widely across seeds and implementation details [7].

In this thesis, robustness is treated both as resistance to such scenario variations and as the ability to withstand intentional disturbances from other agents. Adversarial learning studies show that DRL policies can be sensitive to carefully crafted perturbations or adversarial behaviors, motivating systematic stress testing and adversarial fine-tuning as complementary tools for evaluation and improvement [100, 36].

At the strategic level, route planning determines global paths through road networks [8, 97, 113]. At the tactical level, the agent must decide maneuvers such as overtaking, yielding, or selecting a parking maneuver [38, 115, 113]. At the operational level, it must generate and execute feasible trajectories, controlling continuous variables such as steering, throttle, and braking [93, 38]. Classical modular architectures separate these levels into prediction, behavior planning, and motion planning [93, 113], while DRL-based systems may collapse tactical and operational decision making into a unified policy.

In the presented work, we address this hierarchy in different AD contexts. In highway-env, tactical decision making is modeled with discrete actions corresponding to high-level maneuvers such as “keep lane,” “overtake,” or “move to rightmost lane.” In Unity, the focus shifts to low-speed maneuvering, where the agent must navigate around obstacles in a 3D environment, emphasizing CL to promote stable training and generalization. In CARLA, the challenge becomes automated parking (AP), where realistic physics and photorealistic rendering demand continuous control and reward functions (RFs) that combine dense shaping with sparse safety constraints.

Beyond stable policy generation, the thesis addresses robustness through adversarial RL, formulated as a two-player Markov game [72]. Here, the AV acts as the victim agent, while an adversary is trained to expose weaknesses. The adversary may pursue a purely competitive reward or a regularized objective that discourages unrealistic strategies. This adversarial setup provides a systematic framework for stress-testing policies and for adversarial fine-tuning, allowing robustness to be treated not as an emergent property but as an explicit target.

The problem formulation is thus twofold: first, to design DRL agents driven by safe and effective decision-making policies across tasks of increasing fidelity and abstraction; second, to develop frameworks for evaluating and enhancing robustness under environmental variability and adversarial interaction.

1.3 Contribution

This thesis is organized around the gaps identified in the previous section. In particular, it addresses (i) safety-oriented decision making under reward-based learning, (ii) generalization under scenario variation, (iii) training stability and reproducibility, and (iv) robustness assessment and improvement under disturbances and adversarial interactions.

To address the safety and decision-making gap, we study high-level DRL decision making in highway scenarios using highway-env, where discrete action abstraction supports safety-oriented reward shaping and improves sample efficiency compared to direct low-level control. To address training stability and generalization, we introduce a CL setup for low-speed maneuvering in Unity and analyze how curriculum design affects learning dynamics and out-of-configuration performance. To address generalization and evaluation at higher fidelity, we develop DRL agents for AP in CARLA and propose scenario diversification mechanisms to test performance across geometric variations under realistic dynamics and continuous-control constraints. Finally, to address robustness evaluation and improvement, we introduce adversarial policy learning to systematically stress-test highway policies and propose RAFT, a regularized adversarial fine-tuning framework for parking tasks, designed to expose and mitigate failure modes that arise under disturbances or adversarial agent behaviors.

1.4 Organization of the thesis

The remainder of this thesis is organized as follows. Chapter 2 reviews the current state-of-the-art for both decision making for AD and adversarial policies, as well as providing theoretical background on RL, DRL algorithms such as PPO, CL, and robustness. Chapter 3 presents high-level decision making in highway-env, focusing on the generation of different

driving styles through reward shaping. Chapter 4 addresses decision making for AP in high-fidelity simulation environments such as Unity and CARLA, examining CL and scenario diversity as a method to improve generalization. Chapter 5 introduces adversarial policy learning, presenting its application to highway driving and the RAFT framework for AP. Chapter 6 concludes by synthesizing insights and discussing implications for robust AD.

Chapter 2

Background

2.1 Related work

2.1.1 DRL for decision making in automated driving

Decision making represents one of the central challenges in AD, as it connects the perception and control layers by determining how and when maneuvers such as lane changes, merging, overtaking, or intersection crossing should be executed [76]. Unlike trajectory tracking or vehicle stabilization, which can often be addressed with classical controllers such as Proportional–integral–derivative controller (PID) [34] or Model Predictive Control (MPC) [108], tactical decisions require reasoning about dynamic interactions with other vehicles, long-term planning, and compliance with safety and comfort requirements [50]. DRL has been increasingly studied for this purpose, since it allows agents to learn decision policies directly from interactions with traffic environments, without requiring hand-crafted rules or exhaustive enumeration of scenarios [132, 48].

One of the earliest contributions in this area is by Mirchevska et al. [83], who frames lane-change decisions as a discrete action problem, where the agent could choose to keep its lane or initiate a left or right lane change. Low-level motion control was decoupled from this process and handled by a separate planner, ensuring feasibility of the selected maneuver. The reward design incorporated safety margins, legality constraints, and comfort metrics such as jerk and acceleration bounds. Evaluation in highway-like scenarios showed that the DRL agent learned policies that were not only collision-free but also smoother than rule-based baselines.

Following this, attention shifted to modeling inter-vehicle interactions more explicitly. Leurent et al. [64] present an architecture that integrates an attention mechanism into the policy network. Rather than concatenating the states of surrounding vehicles into a fixed-size

input, the attention module dynamically weighs the influence of each neighbor, ensuring permutation invariance and allowing the policy to scale to varying traffic densities. This design enabled the agent to focus on the most critical vehicles, such as those in adjacent lanes during merging. The framework was evaluated in dense multi-lane traffic environments, where it outperformed standard DRL approaches in terms of both safety and efficiency, producing decisions that appeared more socially compliant. The introduction of social attention marked a step forward by showing how relational reasoning can be incorporated into decision-making agents.

The problem of reliability under unfamiliar conditions was addressed by Hoel et al. [49]. Recognizing that DRL agents can behave unpredictably in out-of-distribution states, the authors augmented decision policies with uncertainty quantification. They employed ensembles with randomized prior functions to estimate epistemic uncertainty in Q-value predictions. This mechanism allowed the system to detect when its decisions were unreliable and switch to fallback strategies that favored safety. Experiments in highway driving scenarios showed that uncertainty-aware agents achieved lower collision rates than standard Deep Q-Network (DQN) [84] agents, especially in corner cases where the distribution of training data was sparse. By explicitly addressing confidence, this work bridged the gap between pure learning approaches and the safety requirements of real-world deployment.

While uncertainty quantification provides a safeguard, other studies emphasized improving scene representation to enhance policy learning. Liu et al. [73] proposes a method incorporating a Transformer encoder with a Soft Actor–Critic (SAC) [43] agent. The Transformer captures spatial and temporal relations among traffic participants, enabling the policy to infer intentions and interactions over time. In comparison to convolutional or fully connected encoders, the Transformer-based representation significantly improved sample efficiency and policy robustness. Evaluations in urban driving environments confirmed smoother maneuvers, fewer abrupt lane changes, and higher route completion rates. This contribution highlighted the benefits of cross-pollination between advances in sequence modeling and DRL-based decision making.

More recent works further extend this idea by explicitly modeling spatio-temporal interactions among traffic participants within the decision-making process. Zhang et al. in [3] proposed a DRL framework that integrates Long Short-Term Memory (LSTM) networks with Graph Attention Networks (GAT), enabling the agent to jointly reason over temporal evolution and relational dependencies among surrounding vehicles. By embedding this spatio-temporal fusion module into the policy network, the agent demonstrated improved decision consistency and robustness in dense traffic scenarios, outperforming conventional DRL architectures relying on static state encodings.

Safety-oriented formulations constitute another important line of research. He et al. [45] introduce adversarial perturbations that are deliberately injected into the agent's observations during training. The actor–critic framework was constrained to maintain stable behavior under these disturbances, thereby producing policies robust to sensor noise and partial observability. In controlled lane-change tasks, the observation-robust agents exhibited significantly fewer unsafe maneuvers compared to standard baselines. Complementing this, Mohammadhasani et al. [85] develop a method in which unsafe trajectories are excluded from the replay buffer by design. This mechanism prevents catastrophic exploration, accelerates convergence, and improves the overall safety of learned policies. Together, these works demonstrate how safety constraints and robustness considerations can be embedded directly into the DRL training process.

Building on safety-aware formulations, recent studies have explored robustness through the integration of expert knowledge into DRL decision policies. Li et al. [66] proposed a framework where expert guidance is incorporated during training to constrain policy updates in critical situations, reducing unsafe exploration and improving generalization to unseen traffic conditions. Experimental results showed that expert-guided agents achieved lower collision rates and more stable decision behaviors compared to purely data-driven DRL baselines.

Similarly, Wang et al. [130] address the limitations of single-expert imitation by introducing a DRL framework trained with multiple expert demonstrations. By aggregating heterogeneous driving strategies within a PPO-based policy, the proposed approach mitigated expert bias and improved policy robustness across diverse driving styles. This line of work highlights a growing trend toward hybrid learning paradigms, where DRL is combined with structured prior knowledge to enhance reliability in complex decision-making tasks.

Beyond general tactical decision frameworks, some research has targeted scenario-specific challenges. Wu et al. [137] proposed a pipeline where an auxiliary critic provided additional gradient information to stabilize the actor–critic training loop. This approach reduced variance in gradient estimates and improved policy robustness across diverse traffic scenarios, illustrating how architectural modifications can strengthen DRL decision-making systems.

Most recently, emerging research has investigated the integration of large language models (LLMs) into DRL-based decision-making pipelines. Xu et al. [141] introduced a teacher–student framework in which an LLM provides high-level guidance to a DRL agent during training. The LLM acts as a semantic reasoning module, offering corrective feedback and shaping rewards in complex scenarios. This hybrid approach demonstrated accelerated

Table 2.1 Summary of state-of-the-art DRL approaches for decision making in automated driving.

Ref	Approach	Simulator	Scenario
[83]	High-level Decision Making	PELOPS [1]	Highway, Lane-change
[64]	Transformer [129]	highway-env [63]	Highway, Traffic
[49]	Randomized Prior Functions (RPF)	SUMO [59]	Highway
[73]	Transformer [129] + SAC [43]	CARLA [28] + SMARTS[144]	Urban
[3]	LSTM-GAT Spatio-temporal DRL	highway-env [63]	Highway, Dense traffic
[45]	Observation-adversarial DRL	SUMO [59]	Lane-change
[85]	DQN [84]	highway-env [63]	Highway
[66]	Expert-guided DRL	SMARTS [144]	Highway, Lane-change
[130]	Multi-expert DRL (PPO) [112]	CARLA [28]	Urban driving
[137]	Random network distillation (RND) [14] + PPO [112]	TORCHS [138]	Race track
[141]	LLM-guided DRL (Teacher-Student)	TESSNG [53]	Urban, Multi-scenario
[98]	Deep Deterministic Policy Gradient (DDPG) [70]	CARLA [28]	Urban navigation
[42]	PPO [112]	CARLA [28]	Intersection

learning and improved robustness in challenging traffic situations, suggesting a promising direction for combining symbolic reasoning capabilities with data-driven decision policies.

The application of DRL to high-fidelity simulators has further validated its potential. Pérez-Gil et al. [98] implements an end-to-end DRL controller that combined sensor inputs with continuous control actions. The agent successfully navigated urban layouts with traffic and obstacles, confirming the feasibility of DRL in realistic simulation environments, but also revealing challenges in training stability and generalization. Gutiérrez-Moreno et al. [42] focuses on the highly interactive scenario of unsignalized intersections. Agents learned gap acceptance and priority handling strategies, reducing deadlocks and collisions compared to heuristic approaches. These results demonstrate how DRL can autonomously acquire cooperative behaviors in complex multi-agent traffic contexts, an essential step toward broader applicability in urban driving.

A summary of the presented state-of-the-art works for DRL decision making in AD is reported in Table 2.1.

Driving simulation frameworks

The study of decision making with DRL relies heavily on simulation, since training directly on real vehicles is unsafe, expensive, and impractical due to the vast number of interactions required. Simulation frameworks provide controllable, repeatable, and safe environments in which to prototype and evaluate learning-based agents. Among the most widely used in the AD research community are highway-env [63], Unity ML-Agents [2], and CARLA [28], each offering different levels of fidelity, flexibility, and computational efficiency. Table 2.2 offers a general overview of the compared simulation frameworks.

Unity is a multi-industry graphic engine that can be flexibly modified in terms of its simulation environment and settings, with a variety of easily pluggable automotive sensors, such as lidars, radars, cameras, and semantic cameras. Additionally, the generation of different observations is straightforward. The graphical user interface makes the interaction pleasant and efficient, without the need to explore the source code. A Unity simulation implies physics computation and rendering, at least when the input to the agent includes a camera, leading to much higher computational demand. This issue (which is common to all environments) can be efficiently handled because Unity offers the possibility of instantiating several environments in a single process (scene) and/or executing several scenes simultaneously, which significantly reduces the training time by exploiting the different cores of the CPU. As a major limitation, the types of selectable NN architectures (and tunable parameters) are very limited (e.g., dense and convolutional NNs, which do not implement state-of-the-art mechanics such as attention). Unity can be interfaced through the ML-agents Python library as an Open AI Gym environment, thus allowing the exploitation of any NN architecture or custom. However, the available documentation is limited, and we are not aware of reports on this topic in the literature.

Highway-env, on the other hand, is an automotive-specific DRL environment that directly supports high-level concepts such as vehicles and lanes. The framework offers several predefined simulation environments representing different driving scenarios, such as driving on a highway, navigating a roundabout, and parking. In addition to the learning agent (i.e., the ego vehicle), such environments can also be easily populated by concurrent nonplayer vehicles (NPVs). The built-in hook to OpenAI's Gym API enables very efficient and fast model training and policy tuning. All environments are indeed rendered in 2D with a bird's-eye view and can be easily interfaced through the Pygame Python library. For cars, highway-env employs the bicycle kinematic model [101], a linear acceleration model based on the intelligent driver model (IDM) [127], and lane-changing behavior based on the MOBIL [55] model. The behavioral decisions of the nonego cars are made through heuristics. One of the main advantages of highway-enhanced methods is their simplicity and low computational cost. This makes it a great tool for developing new and benchmarking DRL algorithms without the added complexity of more sophisticated environments [11]. The goal, RFs, and observation space for the learning agent can be easily set and adjusted at the user's discretion to best fit the problem at hand. Modifying the environment (e.g., the number of lanes, the topology of roads, and intersections) and implementing variations of the tasks (e.g., overtaking, reaching a goal in a narrow space) is possible by operating at the source code level. On the other hand, the realism and variety of scenarios and objects are

limited (e.g., there are no pedestrians or motorbikes/bicycles), which are necessary for proper training and testing of a realistic agent.

SUMO [59] is a microscopic traffic simulator widely adopted for learning and evaluating tactical driving policies, particularly in highway and lane-change scenarios. Its lightweight design and efficient execution enable large-scale experimentation and rapid policy iteration, making it especially suitable for curriculum learning, safety analysis, and ablation studies. However, SUMO relies on simplified vehicle dynamics and abstract representations, limiting its applicability for perception-driven or low-level control tasks.

SMARTS [144] has emerged as a modern multi-agent simulation platform explicitly designed for learning-based autonomous driving. Built on top of SUMO, SMARTS extends traditional traffic simulation by supporting heterogeneous agents, social driving behaviors, and standardized evaluation benchmarks inspired by real-world datasets. Its tight integration with reinforcement learning frameworks and focus on interaction-rich scenarios make it particularly relevant for studying decision making under dense traffic and multi-agent conditions. As such, SMARTS bridges the gap between lightweight traffic simulators and high-fidelity environments, providing a scalable yet interaction-aware testbed for DRL research.

TORCS (The Open Racing Car Simulator) [138] is an open-source driving simulator originally developed for racing scenarios, which has been widely adopted in early reinforcement learning research. Its Python-accessible variant, often referred to as TORCHS, has been used to evaluate DRL-based decision and control policies under high-speed and continuous-action conditions. TORCS provides a simplified yet physically grounded vehicle model and supports continuous control inputs, making it suitable for studying decision making coupled with aggressive maneuvers and control constraints. While its focus on racing environments and limited traffic interaction reduce its relevance for general urban or highway driving, TORCS remains a valuable benchmark for analyzing learning stability, exploration strategies, and policy robustness in dynamic and safety-critical scenarios. Consequently, it is primarily employed as a specialized testbed rather than a comprehensive automated driving simulator.

CARLA is an open-source platform for simulating and testing AD systems. It provides a high degree of flexibility in terms of customization and control of the environment's physics, supporting the design of complex scenarios, which is very useful for DRL experimentation. CARLA provides predefined maps and tools (e.g., Scenario Runner [18] and map editor [17]) for developing simulations in highly realistic and/or customized settings. It also features an impressive catalog of carefully simulated vehicle models, with parameters including the maximum RPM, the moment of inertia of the vehicle's engine, the gear switch time, and the drag coefficient of the vehicle's chassis. This is also possible thanks to the reliance on the Unreal Engine game engine [128], which provides high-quality 3D rendering and

physics simulations. CARLA uses a modified Unreal 4.26 fork, which contains patches specific to CARLA and requires powerful hardware. Vehicles' dynamic models are based on NVIDIA PhysX [90], which is the standard Unreal Engine 4 vehicle model. The tool's learning curve is quite steep. The overall sophistication and complexity of the environment also have implications for training, which is sensitive to parameters such as the vehicle type, the observations, and, particularly, the timing of the simulation world advancement and the agent decision.

Table 2.2 Comparison of simulation frameworks commonly used for DRL decision making in automated driving.

Simulator	Features	Strengths	Application focus
Highway-env [63]	Lightweight Python/Gym environment for highway and multi-lane driving	Very fast simulation, easy reproducibility, simple interface	Early-stage prototyping, benchmarking of decision-making algorithms
Unity ML-Agents [2]	General-purpose 3D engine with RL API	High flexibility, completely customizable scenarios	Proof-of-concept studies, visually rich but controlled driving tasks
SUMO [59]	Traffic simulation with abstract vehicle dynamics	Extremely fast, scalable, ideal for large-scale DRL training	Tactical decision making, highway and lane-change scenarios
SMARTS [144]	Multi-agent driving simulator built on SUMO with RL benchmarks	Interaction-aware, supports social driving and multi-agent learning	Decision making under dense traffic and multi-agent conditions
TORCS [138]	Racing-oriented simulator with continuous vehicle dynamics	Stable physics, suitable for continuous-action DRL	High-speed driving, decision and control in racing scenarios
CARLA [28]	Dedicated AD simulator on Unreal Engine with urban layouts and full sensor suite	High realism, extensibility, community benchmarks	Comprehensive evaluation of AD systems in realistic traffic scenarios

In summary, the choice of the simulation framework depends on the specific needs and goals of the research project. If realism and physical precision are the main priorities, CARLA would be the ideal choice. Highway-env, on the other hand, is a very convenient option for easy deployment and rapid prototyping and seems particularly suitable as a starting benchmark for novel AD DRL algorithms. Finally, the Unity ML-Agents toolkit may provide an excellent trade-off between the fidelity of VR representations and the efficiency of modeling, training, and simulation.

2.1.2 Robustness and adversarial learning

Robustness is a central requirement for deploying DRL-based decision policies in safety-critical driving scenarios. Recent attempts to improve robustness deploy different kinds of

adversarial policies in robotic simulation problems. In [100], Pinto et al. introduce Robust Adversarial Reinforcement Learning (RARL), which consists in training an adversarial agent with the objective of modifying the environment dynamics to destabilize the victim agent. At first, only the victim policy is trained while the adversarial policy is frozen. Afterwards, the opposite is true and this sequence is repeated until convergence. This results in a more stable and robust final victim policy when compared to the application of traditional DRL methods.

This method has also been improved upon by applying physical constraints to the adversarial perturbation that the agent can apply to disrupt the victim, achieving promising results even under real-life thresholds [80]. Pan et al. [94] adopt a risk-aware version of the original RARL work by deploying a risk-averse protagonist and a risk-seeking adversary in a competing AD scenario to include catastrophic or unexpected events in the models trained buffer.

The adversarial policy has direct access to environment dynamics and is able to act upon the state of the system itself, whereas more recent studies introduce a physical attacker entity acting in the simulation environment the same way as the victim would. Gleave et al. [36], introduce this concept by analyzing the effect this adversarial agent has on the victim on an observation level in different zero-sum two-player competitive games. As in RARL, the objective of the attacker is defined through the minimization of the reward intake of the victim model. The victim policies are pre-trained through classical self-play and then frozen for the duration of adversarial learning. The adversarial policy fails to effectively learn how to play the game efficiently, but still manages to win against a well-trained opponent by inducing off-distribution neural activations in the victim's policy network. Here, through a re-training procedure, the victim policy becomes robust to the current adversary. Such adversarial policies have also proven to be capable of beating state-of-the-art AIs at GO, which were capable of defeating human champions of the game [133].

Since the proposed methods only tackle zero-sum games, their applicability to more complex problems remains limited. To assess this problem, Guo et al. [41] introduce a new adversarial learning algorithm by designing a new optimization function, based on the Trust Region Policy Optimization (TRPO) [111] algorithm surrogate objective. This enables the training of an effective adversarial policy guiding an agent in a two-player competitive scenario without the zero-sum constraint, thus expanding the capabilities of the approach to a broader range of problems. This is showcased through testing to more complex scenarios such as the StarCraft II game.

Minimal DRL-based adversaries have also come to light thanks to recent work by Lu et al. [78].The proposed adversarial model can only add to the observation as a fixed deterministic function of the remaining state and is unable to modify or block any information

that the victim could collect from the environment. This proves how even minimal adversarial influence can efficiently disrupt the training procedure of a DRL-based policy and highlight potential weaknesses.

The efficacy of adversarial policies in disrupting the normal behavior of the victim model usually declines with the amount of information regarding the environment that the target model has access to. To prove that efficient adversaries are still capable of major disruption even in partially observable scenarios, Ma et al. [79] introduce a divide-and-conquer approach in a multi-agent system. Here, many adversarial sub-policies are learned for a collection of sub-problems involving the victim and the best for the current instance is chosen optimally. Results show how this kind of attack is effective even with victims able to observe tiny amounts of the complete environment state.

The use of the adversarial policy paradigm comes with the major limitation of scenarios' realism. Adversarial policies studied in robotics simulation problems usually leverage the observable space of their victims to effectively induce policy failure states at the cost of unnatural and unrealistic movements and behaviors. To generate more natural and realistic adversarial policies, He et al. [44] proposed the RIGID method to construct a natural-adversarial frontier by training adversarial human-like policies for an robotics human-assistive problem. Here, a discriminator is trained to score the action taken by the adversarial agent based on their naturalness and thus guiding the its learning procedure.

To train policies resistant to adversarial attacks, Liang et al. [69] introduce the Worst-case-aware Robust RL (WocaR-RL) framework. This method directly estimates and optimizes the worst-case reward of a policy under bounded strong attacks, without requiring a trained adversarial agent, thus requiring much lower computational power. This results in stronger victim policies in the tested scenarios. Despite this, recent work by Zheng et al. [143] achieves promising results in terms of policy disruption against WocaR-RL by deploying Intrinsically Motivated Adversarial Policy (IMAP). IMAP is based on four types of adversarial regularizers at the core of the objective function to undermine the victim capabilities methodically. This achieves efficient black-box adversarial policy learning in both single- and multi-agent environments.

Recent work also leverage the generation of multiple adversarial policies to create diverse and realistic scenarios and trajectories to challenge modern AD systems. The deployment of adversarial agents in the AD context has been mainly tackled in the lane-change scenario with an attacker policy guiding multiple environment vehicles at once. In [20], Chen et al. propose a method to train multiple attackers to disrupt the target AD systems, heuristic and reinforcement learned, being also guided by a penalty for violating traffic laws to generate more rational and realistic adversarial behavior. Here, the adversarial policies begin the

training procedure by skipping the exploration step, overfitting a behavior. This is adjusted by generating multiple randomly initialized adversaries whose output trajectories are then clustered through a non-parametric Bayesian approach to better identify the short comings of the victim AD system and study diverse generated scenarios.

A similar approach has been proposed by He et al. in [46], exploiting an adversarial objective which is not directly based on the victim RF. The presented lane-change scenario is similar, but the training phase of the adversarial model is complete and yield to more generalized attack patterns.

More recently, Chen et al. [21] proposed the Feasibility-guided REasonable Adversarial policy (FREA), a framework for generating safety-critical scenarios that remain feasible for the autonomous vehicle (AV). Unlike prior methods that maximize adversariality at the cost of leading to unavoidable collisions, FREA leverages the concept of the Largest Feasible Region (LFR) of the AV as an upper bound for adversariality. Experiments in the CARLA simulator, combined with SafeBench scenarios, show that FREA successfully generates near-miss events while reducing over-collision cases and achieves robust generalization across different surrogate AV methods and environments [21].

Beyond DRL-trained adversarial agents that explicitly minimize the victim return, recent work in autonomous driving has increasingly shifted towards closed-loop scenario generation in which adversarial behaviors are constrained by realism, while still exposing failure modes of the ego policy. This trend complements feasibility-aware MARL approaches (e.g., FREA) by leveraging data-driven priors and structured control mechanisms to generate critical-yet-plausible interactions.

In this direction, Safe-Sim [19] casts adversarial scenario generation as a controllable diffusion process in closed loop. Rather than learning an attacker purely as a reward-maximizing policy, adversariality is introduced as guidance during denoising, enabling the synthesis of challenging multi-agent rollouts while preserving reactive behaviors and overall realism. Crucially, the framework supports explicit control over scenario attributes (e.g., collision type and aggressiveness), aligning the generated outcomes with evaluation needs rather than producing uncontrolled worst-case behaviors.

SEAL (Skill-Enabled Adversary Learning) [118] builds a hierarchical, skill-conditioned adversary whose actions are sampled from an adversarial skill prior learned from safety-critical segments, thereby discouraging unnatural interactions. A learned scoring objective ranks adversarial futures not only by collision proximity but also by the magnitude of ego behavior deviation (e.g., forced braking or evasive steering), which directly targets policy brittleness under realistic reactive perturbations in closed-loop training.

Table 2.3 Summary of the presented state-of-the-art for adversarial learning.

Ref	Approach	Simulator	Scenario
[100]	RARL	MuJoCo [125], Gym [13]	Inverted Pendulum; Half Cheetah; Swimmer; Hopper; Walker2D
[80]	RARL with physical constraints	MuJoCo, Gym	Inverted Pendulum; Walker2D; Half Cheetah; Hopper
[94]	Risk-aware RARL	TORCS [138]	Ego driving on the Michigan Speedway
[36]	Adversarial policy in zero-sum games	Created by [9]	Kick and Defend; You Shall Not Pass; Sumo Humans; Sumo Ants
[41]	Adversarial policy without zero-sum constraint	Blizzard Ent. + Created by [9]	StarCraft II + Kick and Defend; You Shall Not Pass; Sumo Humans
[78]	Minimal attacker with Evolution Strategies (ES)	MuJoCo [125], Gym [13]	Cartpole; Inverted Pendulum; Reacher; Minatar Breakout
[44]	RIGID: Natural-Adversarial Frontier	Assistive Gym [31]	Assistive task: itch scratching with unknown itch locations
[79]	MARL: SUB-PLAY	Created by [24]	Predator-prey
[69]	WocaR-RL (Defense)	MuJoCo [125], Atari	Pong; Ant; Half Cheetah; Walker2d; Hopper
[143]	MARL: IMAP	MuJoco, Created by [9]	AntUMaze; Fetch Reach; You Shall Not Pass
[20]	MARL: Adversarial policy & Bayesian clustering	CARLA [28]	Lane-change
[46]	MARL: Adversarial-oriented DRL	Not reported	AD Lane-Change
[21]	FREA: Feasibility-guided adversarial scenario generation	CARLA [28] + SafeBench [140]	Intersections, lane-keeping, near-miss events
[19]	Diffusion-guided controllable closed-loop adversarial scenario generation	Bits [142]	Urban; Intersection navigation
[118]	Skill-based adversary learning for closed-loop scenario perturbation	MetaDrive [68]	Highway on-ramp merging
[32]	Sparse adversarial observation attack on DRL driving policies	SUMO [59]	Unprotected left turn; On-ramp merging

Adversarial influence can also be exerted without introducing a physically embodied attacker, by selectively corrupting the victim's observations. The Less Is More attack [32] emphasizes a stealth-oriented threat model in which effectiveness is maximized under a constrained intervention budget: instead of continuous perturbations, the attacker learns when to intervene and applies gradient-based perturbations only at a few critical timesteps. This highlights that even sparse, low-frequency adversarial interventions can induce severe degradation in DRL driving policies, and motivates defenses that explicitly account for intermittent and hard-to-detect attack surfaces.

A summary of the presented state-of-the-art works is displayed in Table 2.3.

Robustness perspectives

Robustness in RL has been studied from multiple perspectives. Classical robust MDPs optimize worst-case performance across uncertainty sets in transition dynamics [51, 88]. Risk-sensitive RL incorporates measures such as Conditional Value-at-Risk (CVaR) to account for tail-risk events [22, 122]. Distributional RL models the entire return distribution, offering a richer view of variability [10].

More recent surveys classify robustness into categories such as transition robustness, disturbance robustness, action robustness, and observation robustness [86]. Transition robustness accounts for uncertain dynamics, disturbance robustness captures perturbations or adversarial forces [100], action robustness models actuator corruption, and observation robustness deals with sensor noise and adversarial input manipulation [96]. In AD, this can be attributed to changing road conditions, external disturbances, actuator delays, and perception errors.

For AD, robustness must also encompass environmental, interactional, and algorithmic dimensions. Environmental robustness denotes stable performance under variations in geometry, traffic density, or weather, addressed by strategies such as domain randomization

[124, 12, 134]. Interaction robustness refers to safety under the behaviors of other agents, which may be adversarial. Adversarial policy learning operationalizes this dimension through explicit opponent modeling [36, 80]. Algorithmic robustness concerns the reproducibility and stability of DRL training, long recognized as sensitive to random seeds and hyperparameters [47, 30]. In simulators such as CARLA, additional fragility may arise from synchronization mismatches and computational overhead.

Recent research has also explored certifiable robustness, with approaches such as smoothing-based certification (CROP) providing guarantees on action stability under bounded perturbations [136]. Population-based adversarial training extends robustness by exposing victims to diverse opponent policies [107]. Critiques of simple minimax formulations argue that adversarial robustness must be redefined to reflect realistic behavioral constraints and structural properties of policies [58].

In the context of the conducted studies, robustness is defined broadly as the sustained ability of DRL-based policies to preserve safety and task performance under environmental variations and adversarial interactions. The methods employed, including curriculum learning, environment diversification, and adversarial fine-tuning, are complementary strategies for embedding robustness into decision-making policies.

2.2 Reinforcement Learning

Reinforcement learning (RL) is a paradigm for sequential decision making in which an agent learns to act in an environment so as to maximize a long-term measure of cumulative reward. Unlike supervised learning, which relies on labeled data, or unsupervised learning, which extracts structure from static datasets, RL is inherently interactive: the agent learns by trial and error, guided by feedback signals from the environment. This makes RL particularly suitable for problems where explicit supervision is scarce or impractical, but where the consequences of decisions can be observed over time [119].

Early RL algorithms operated in tabular settings. Q-learning [135] estimates the action-value function through temporal-difference updates and has convergence guarantees under certain assumptions. SARSA [109] is its on-policy counterpart, updating Q-values according to the actions selected by the current policy. Policy iteration and value iteration provided additional methods rooted in dynamic programming. However, these approaches scale poorly with large or continuous state-action spaces due to the curse of dimensionality.

Function approximation was introduced to overcome these limitations. Linear approximators enabled generalization but lacked expressivity for complex tasks. The breakthrough came with the use of DNN as function approximators, giving rise to DRL. The Deep Q-Network

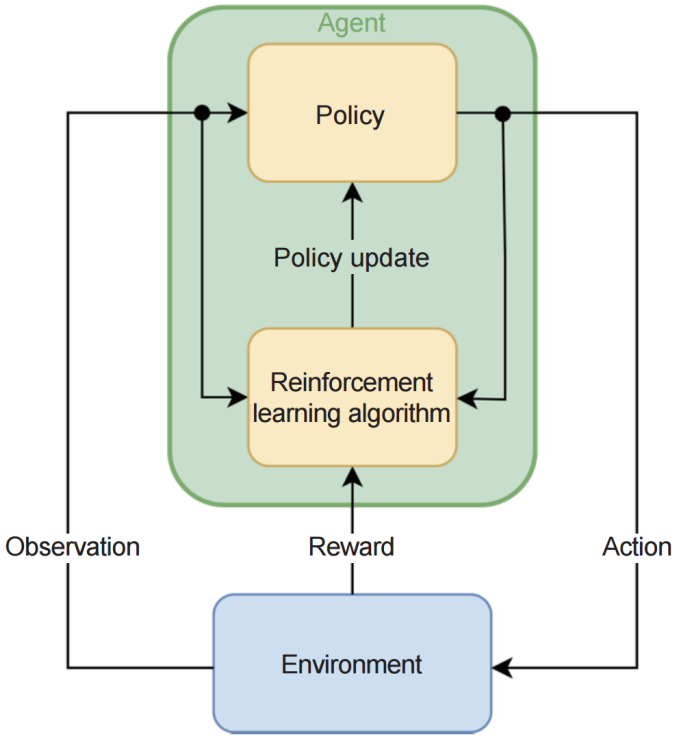


Fig. 2.1 Reinforcement learning loop.

(DQN) by Mnih et al. [84] demonstrated the potential of DRL by combining Q-learning with convolutional neural networks to play Atari games directly from pixels. Stabilization mechanisms such as experience replay and target networks made this feasible, establishing DRL as a powerful paradigm for high-dimensional decision problems.

Beyond value-based methods, policy gradient approaches optimize parameterized policies directly. Although these gradients are unbiased, they exhibit high variance. Actor-critic methods mitigate this by combining a parameterized policy (actor) with a value function (critic) to guide learning. A schematic representation of the classic actor (agent)-environment interaction loop is provided in Fig. 2.1.

2.2.1 Formal foundations

Formally, an RL problem is often modeled as a Markov Decision Process (MDP) defined by a tuple $(\mathscr{S}, \mathscr{A}, \mathscr{T}, R, \gamma)$, where:

- $\mathscr{S}$ is the set of states.
- $\mathscr{A}$ is the set of actions.

- $\mathscr{T} : \mathscr{S} \times \mathscr{A} \times \mathscr{S} \to [0,1]$ is the state transition probability function, where $\mathscr{T}(s'|s,a)$ gives the probability of transitioning to state s' from state s after taking action a.
- $R : \mathscr{S} \times \mathscr{A} \to \mathbb{R}$ is the RF, where $R(s,a)$ gives the immediate reward received after taking action a in state s.
- $\gamma \in [0,1]$ is the discount factor, which determines the importance of future rewards.

The objective is to train a policy $\pi : \mathscr{S} \to \mathscr{A}$ that maximizes the expected discounted return, defined as $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$, where G_t is the return at time step t. Two fundamental concepts in RL are the state-value function $V^\pi(s)$ and the action-value function $Q^\pi(s,a)$. The state-value function is defined as the expected return starting from state s and following policy π:

$$V^\pi(s) = \mathbb{E}[G_t|s_t = s] \tag{2.1}$$

The action-value function is defined as the expected return starting from state s, taking action a, and thereafter following policy π:

$$Q^\pi(s,a) = \mathbb{E}[G_t|s_t = s, a_t = a] \tag{2.2}$$

RL algorithms can be categorized into value-based methods, policy-based methods, and actor-critic methods. Value-based methods focus on estimating the value functions, policy-based methods directly optimize the policy, and actor-critic methods combine both approaches.

2.2.2 Proximal Policy Optimization

Proximal Policy Optimization (PPO) [112] is a state-of-the-art DRL, policy gradient algorithm. PPO is simpler to implement and train than its predecessor, Trust Region Policy Optimization (TRPO) [111], while achieving similar or better performance. This method has been deployed throughout all of the current work.

Policy Gradient Methods

Policy gradient methods optimize policies by increasing the probabilities of actions that yield higher returns and decreasing those that yield lower returns. For a stochastic policy π_θ with

parameters θ, the gradient of the expected return $J(\pi_\theta)$ is:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) A^{\pi_\theta}(s_t, a_t)\right]. \tag{2.3}$$

where A^{π_θ} is the advantage function, which measures the benefit of taking action a in state s compared to the average action.

Clipped Surrogate Objective

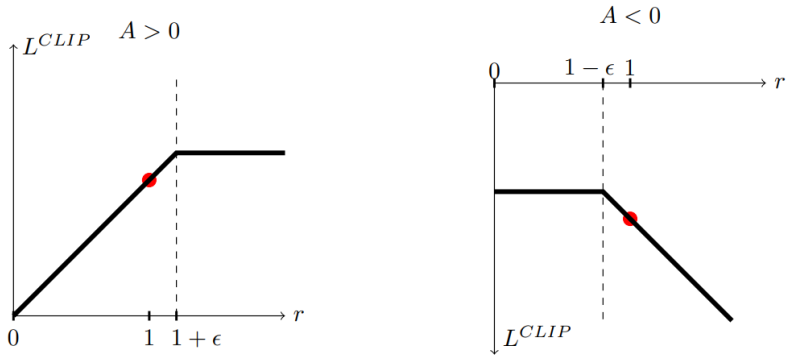


Fig. 2.2 Surrogate function L^{CLIP} for positive (left) and negative (right) advantages. The red circle shows the starting point ($r = 1$).

PPO improves training stability by introducing a clipped surrogate objective. This objective limits the policy change at each timestep, preventing instability caused by large updates. The PPO objective function is:

$$L^{CLIP}(\theta) = \mathbb{E}t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t)\right], \tag{2.4}$$

where $r_t(\theta)$ is the probability ratio of the new and old policies, and ε is the clipping factor. This formulation ensures stable policy updates by considering both the unclipped and clipped objectives and minimizing the overall term. PPO alternates between sampling data from the environment and optimizing the surrogate loss with mini-batch gradient descent. By iteratively adjusting the policy parameters based on a clipped surrogate objective, it strikes a balance between exploration and exploitation while maintaining stability. A graphical representation of the PPO clipped surrogate function is depicted in Fig. 2.2.

2.2.3 Curriculum learning

Curriculum learning (CL) arranges experience from easy to hard to speed up training and improve generalization [12]. In RL, the idea becomes a training schedule that first exposes the agent to simplified instances with dense signals, then increases difficulty and sparsity once core skills emerge [103]. This ordering shapes exploration, stabilizes updates, and reduces wasted interaction in very large state and action spaces [87]. In practice, curricula can be applied by progressively increasing traffic density, initial-state variability, or disturbance magnitude as the policy competence improves.

Environmental curricula vary initial states such as positions, scene layouts, and disturbance levels to control task difficulty [102]. Task curricula stage objectives so the agent solves base skills before multi objective goals. Reward curricula transition from shaped signals to sparse goals to avoid plateaus while keeping the target objective intact [87]. Automatic curricula replace manual schedules with teacher algorithms that adapt difficulty online to the learner's competence, for example by sampling start states around the goal and expanding outward as performance improves [35, 81, 102].

Empirical evidence supports faster and more reliable convergence on tasks that otherwise fail with naive training due to sparse rewards or brittle exploration as well as better generalization when evaluation conditions differ from training. Agents trained with curricula tolerate broader variations in layouts and dynamics, consistent with broader RL findings that diverse training distributions mitigate overfitting [23, 134, 117]. These gains appear across control and robotics domains and extend to driving simulators where difficulty is raised by increasing traffic density, complexity of scenes, or disturbance magnitude [103].

Chapter 3

Decision making in a highway driving environment

The first work that was carried out in the domain of decision making for AD arose from the necessity of studying the capabilities of DRL in such a dynamic and safety-critical context. Thus, the highway-env simulation framework was chosen as a target for this initial study to deploy decision making agents (i.e., able to exploit state of the art ADAS) in a highway scenario. Here, a high-level action space was developed with the goal of abstracting the standard control output of the models to induce safe and human-like behavior. The presented work also highlights the importance of reward shaping in DRL, implementing different driving styles for the agents that can be employed in realistic implementations of traffic. This study in particular has been carried out in the Serious Games (SG) research field, as the developed agents may serve as an instructionally useful representation of diverse and realistic traffic behaviors.

3.1 Method

For the online training procedure of the models, the PPO algorithm [112] was employed. The development relies on the highway-env environment, which exploits Stable-Baselines3 [105] as a DRL support library. The environment represents a multi-lane highway (Fig. 3.1), where the ego vehicle (EV) is controlled by a DRL agent and has the goal of covering as much distance as possible, avoiding collisions with non-player vehicles (NPVs).

NPVs in the training environment follow a heuristic behavior, with some randomness applied in order to avoid overfitting on the agent's part [64]. The number of vehicles is randomized at each episode, and the behavior of the NPVs has been modified to improve

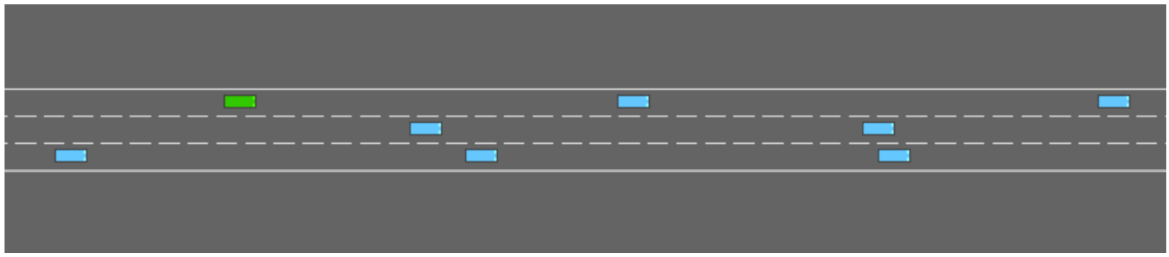


Fig. 3.1 Snapshot from highway-env. The EV is colored green; NPVs are light blue.

realism and EV road navigation. In particular, NPVs drive at a different speed depending on their current lane (slower on the right lane, faster on the left). The discrete lane change decisions are given by the Minimizing Overall Braking Induced by Lane Changes (MOBIL) model [37], [55]. The RF is key to shaping the behavior of an agent, for instance by penalizing high or low speed, or frequent changes of the lane. Thus, a main goal of this research is to explore the shaping and tweaking of different RFs to represent different driving styles, as described in the next section.

3.1.1 Environment

The adopted training environment, depicted in Fig. 3.1, is composed of an EV and several NPVs, whose number is chosen randomly at the beginning of each episode between 15 and 20, that travel along a 3-lane straight highway. The above number of NPVs has been chosen to simulate intense but flowing highway traffic, with an average of about 7 to 8 vehicles ahead of the EV in a 0–100 m range at any time. The maximum allowed speed is 36 m/s (130 km/h). According to the default environment settings, a vehicle cannot drive outside the roadway. The RF, as well as many different environment parameters, can be modified through the default Gym configuration interface by overriding the *'env.config.update'* method and specifying the desired key-value pairs [13].

3.1.2 Observations

The observations provided by the environment are the 5 default features defined in highway-env: presence (whether a vehicle is present or not in the current scene), longitudinal and lateral position and velocities; for the EV and the V closest vehicles, where V is a parameter, set to 6 in our case. The employed observation (i.e., kinematic observation [63]) is a $V \times F$ matrix, where F is the number of observed features. Hence, the considered kinematic observation matrix is a 7×5 matrix in our case, of which an example is provided in Eq. (3.1):

$$K = \begin{bmatrix} 1 & 1.00 & 0.00 & 0.45 & 0.00 \\ 1 & 0.19 & 0.35 & -0.22 & -0.01 \\ 1 & 0.41 & 0.33 & -0.11 & 0.00 \\ 1 & 0.46 & 0.67 & -0.22 & 0.00 \\ 1 & 0.93 & 0.67 & -0.20 & 0.00 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.1}$$

The first row always refers to the EV, while the remaining rows describe, in ascending order of Euclidean distance, the V closest vehicles. The observation is normalized within a fixed range: x and y within [-100, 100] and v_x and v_y within [-20, 20]. The coordinates are relative to the EV, except for the EV itself, which stays absolute. If less than V vehicles are observed in the current scene, the last rows are filled with zeros.

3.1.3 Action space

Our approach involves raising the level of the agent decision making, by keeping into account availability of state-of-the-art ADAS in vehicles (e.g., the ACC). Thus, we built a decision hierarchy where the EV takes high-level decisions while the ADAS (of which we developed a simple model implementation) translates such decisions into the low-level agent action space (i.e., *"faster"*, *"slower"*, *"lane right"*, *"lane left"*, *"idle"*),which is implemented in the highway-env. Particularly, we define the following discrete high-level actions for the decisions of the agent:

- *Keep lane*: the ACC is activated. If a vehicle is present ahead in the lane, the EV keeps a safe time headway. Otherwise, it goes at the maximum speed.
- *Overtake*: change lane to the left so as to make an overtake if a vehicle is present in the ego lane ahead. Otherwise, the Cruise Control is activated.
- *Go to rightmost lane*: change the lane to the right, wait one second, and repeat this until the rightmost lane is reached.

Fig. 3.2 provides a complete overview of the system. Fig. 3.3 shows the vehicular behaviors enacted by the proposed actions. In our design, the EV includes serial subsystems organized in hierarchy [83]: the higher-level Decision Maker (DM) and the lower-level Behavior Executor (BE). The DM gets the observations from the environment and selects the appropriate high-level action accordingly.

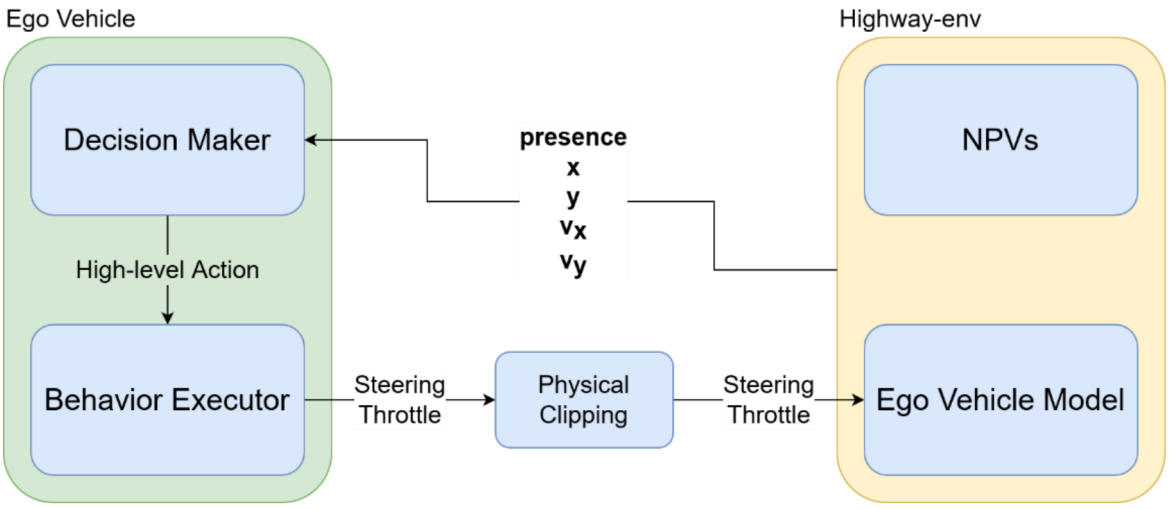


Fig. 3.2 High-level schema of the overall system architecture.

The high-level actions are received by the BE, which implements the appropriate behavior as a series of low-level controls (throttle levels and steering angles). Such values are clipped to enforce physical feasibility before feeding the environment, whose interface is left unchanged. Here we stress that this architecture can implement different behaviors, corresponding to different driving styles. For instance, the DM might indicate the EV to left-overtake vehicles at the maximum possible speed in case of aggressive driving behavior, while a more conservative EV might prefer following the vehicle in front, switching on the ACC.

Each behavior is executed in a loop, which is terminated as soon as the DM chooses a different action than the previous one. The BE implementation simulates the behavior of state-of-the-art onboard ADAS through simple PID controllers in a closed-loop control schema. For deriving the acceleration and steering command, we used a straightforward bicycle kinematic model of the vehicle [101], detailed in Eq. (3.2):

$$\begin{aligned} \dot{x} &= v\cos(\psi+\beta) \\ \dot{y} &= v\sin(\psi+\beta) \\ \dot{v} &= a \\ \dot{\psi} &= \frac{v}{l}\sin\beta \end{aligned} \tag{3.2}$$

where (x,y) is the position of the vehicle, v its forward speed, ψ its heading, a the acceleration command, and β the slip angle (i.e., the angle between the direction of the vehicle and the direction of the front wheels) at the center of gravity, defined in Eq. (3.3):

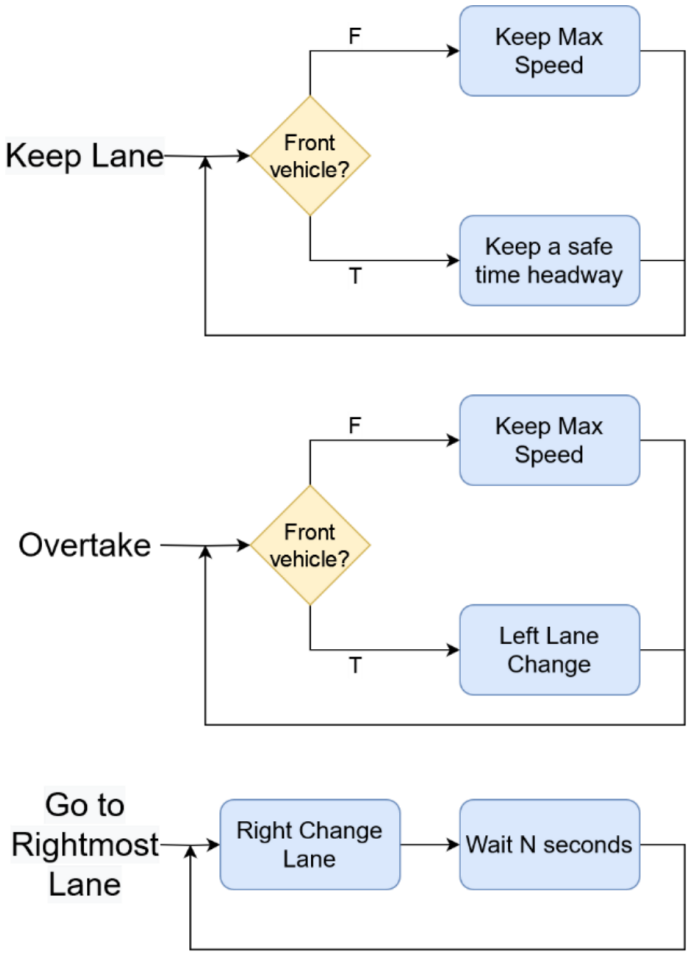


Fig. 3.3 Flowcharts of the high-level behaviors.

$$\beta = \tan^{-1}\left(\tfrac{1}{2}\tan\delta\right) \tag{3.3}$$

where δ is the front wheel angle, which is used as the steering command.

We define a simulation step as a time interval in which the environment state and the derived agent observations are updated according to the changes in the scene. The frequency of the simulation steps is set to 5 Hz. A training step is composed of multiple simulation steps after which the agent's behavior is updated according to the state-action-reward tuples for each simulation step in the algorithms replay buffer. All the variables of the kinematic model are calculated at each model simulation step, and such calculations allow the propagation of the vehicle state.

3.1.4 Reward

We define the RF for this problem as the weighted sum of 3 rewards that are provided by the environment:

$$R = c_{\text{coll}} R_{\text{coll}} + c_{\text{rml}} R_{\text{rml}} + c_{\text{hs}} R_{\text{hs}} \tag{3.4}$$

where:

- R_{coll} (Eq. (3.5)) is the collision reward and is equal to -1 when a collision happens. In that case, the training episode is also terminated, in order to prevent the agent to accumulate positive rewards. On the other hand, if the EV reaches the end of the simulation without crashing, then R_{coll} is equal to 0.

$$R_{\text{coll}} = \begin{cases} -1, & \text{if the vehicle crashes,} \\ 0, & \text{otherwise.} \end{cases} \tag{3.5}$$

- R_{rml} (Eq. (3.6)) is the right-most lane reward, which is employed to encourage the EV to keep the right-most lane whenever it is possible. The maximum reward is given when the EV travels on the right-most lane. The reward linearly decreases until the left-most lane, where it drops to 0. In (Eq. (3.6)), N is the number of lanes in the road. Differently from the previous one, this reward is dense, meaning that it is given to the learning agent at each simulation step.

$$R_{\text{rml}} = \frac{\text{current EV lane index}}{N} \tag{3.6}$$

- R_{hs} (Eq. (3.7)) is the high-speed reward and is designed to encourage the EV to keep high traveling speed values. This reward is dense, as well. The maximum reward is attributed when the cruising speed of the EV is between a certain interval, which in our case goes from roughly 30 to 36m/s (108 to 130 km/h).

$$R_{\text{hs}} = \text{clip}(\text{scaled\_speed}, 0, 1) \tag{3.7}$$

scaled speed is a linear interpolation described as:

$$\text{scaled\_speed} = \frac{v - l_{\text{min}}}{l_{\text{max}} - l_{\text{min}}} \tag{3.8}$$

where l_{min} and l_{max} are respectively the extremes of a speed range in which the agent receives a positive reward and v its forward speed.

The output value of dense rewards is linearly mapped between 0 and 0.1 before being fed to the DRL algorithm, while the collision reward is added as is. This normalization process is necessary to avoid a significant increase in the numerical value of the total reward and, therefore, to stabilize the learning phase.

In the next section, we analyze how, by changing the weights of the proposed reward terms, it is possible to model three different driving styles, such as:

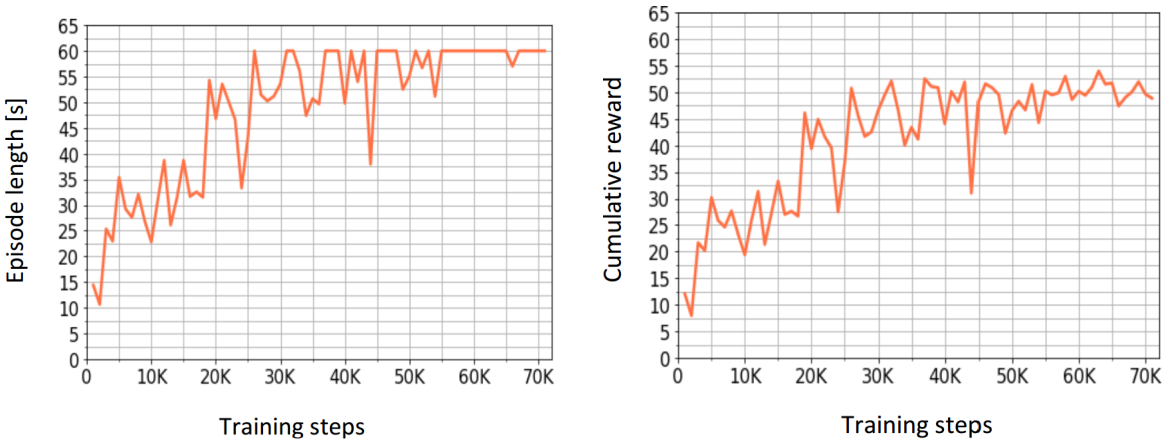
- *Standard*: normal behavior adopted when driving on a highway. It should involve a limited number of collisions.
- *Comfort*: this driving style shows a preference to occupy the rightmost lane, and travel at a relatively slow speed.
- *Aggressive*: behavior that implies high traveling speed, a high number of overtaking maneuvers,and quick decision changes.

3.2 Results

This section presents the results obtained by testing the above described design. Results refer to a training of the agents performed on a laptop with an Intel Core i7-12700H CPU, 16 GB of RAM, and an NVIDIA RTX 3060 GPU.

For the DRL training, we employed a PPO algorithm and tuned the hyperparameters by setting the number of policy update steps to 4096, batch size to 128, gamma to 0.997, and entropy coefficient to 0.1. In PPO, a policy update step occurs after a specified number of simulation steps (4096 in our case), such as for the mentioned training step, and involves selecting a new agent policy, and consequently updating the policy gradient, on the basis of the state-action-reward-new state tuples observed since the last policy update step.The gamma hyperparameter refers to the algorithm's discount factor used to balance the newly obtained rewards in the agent's objective function. In this context, the entropy coefficient quantifies the degree of uncertainty the agent has about its actions given a particular state. In PPO, it is used to encourage the model to explore the environment by penalizing excessively deterministic and fixed behaviors on which the agent may already have converged.

The DNN configuration is the highway-env default 2-layer multi-layer perceptron (MLP), with 64 neurons per layer. We used a 3-lane highway environment (Fig. 3.1), setting policy frequency (i.e., decision rate) to 1 Hz, in order to give the agent sufficient time to observe



(a) (b)

Fig. 3.4 Training metrics (a) episode length and (b) total reward as a function of number of iterations.

the complete effects of its previous decision. The training procedure consists of about 5000 episodes, each one of a maximum length of 60 seconds, and encompassing about 200k training updates. The whole training is completed within 90 minutes.

3.2.1 High-level decision making

Fig. 3.4 shows the evolution of the training of a standard agent in a TensorBoard view [4]. The standard agent has the following weights for the RF terms: -3 for collisions, 0.4 for speed, and 0.2 for the right lane (negative weights indicate penalties). The average episode length reaches its maximum (i.e., 60 seconds, meaning no collisions occurred) at around 60K iterations. The average total reward (Fig. 3.4b) has a similar shape since it is strongly influenced by the collision penalty. We notice that the training is quite rapid, smooth,and stable, without catastrophic forgetting [57], which frequently affects the DRL training.

Table 3.1 Performance over 60-second episodes.

Model	Collisions	Km travelled
High-level DM agent	0	99
Low-level agent [16]	2	105.7

It appears that the high-level agent performs better under both the considered dimensions. It is important to highlight that the training of the higher-level agent took more than one order of magnitude less training steps than the lower-level agent (70K vs. 1.75 M). We argue that such a significant reduction is motivated by the higher simplicity of the action space in

Table 3.2 Environment settings.

Model	Speed levels [m/s]	No. vehicles per episode	Traffic density	No. reward functions
High-level DM agent	[20, 25, 30]	10	0.5	3
Low-level agent [16]	Cont(20, 36)	Uniform(10-20)	0.4	11

Table 3.3 Reward weights for training three different driving styles.

Driving style	Collision	Speed	Right lane
Comfort	30	0	0.8
Standard	3	0.4	0.2
Aggressive	0	0.8	0

the higher-level decision case, in which the agent can exploit state-of-the-art ADASs for the lower-level longitudinal and lateral motion control, and thus focus on the strategic decisions only. An expert assessment also noticed a more realistic behavior by the high-level agent. A quantitative indicator for this is given by the elimination of the collisions.

Concerning the proposed comparison, it must be specified that the two environments are very similar to each other, but not exactly the same. Table 3.2 shows that an episode in Campodonico et al. [16] has fewer total vehicles (EV + NPVs), but they are slightly more concentrated (density factor 0.5 vs 0.4). The model presented in [16] was specifically developed with a large number of rewards compared to the original highway-env (11 vs 3), in order to better enforce traffic law and safer behavior(e.g., penalties for right overtake, unsafe distance to the front vehicle, hazardous lane change, steering angle).

3.2.2 Driving styles

The reduction in training time ensured by the higher-level DM module provides a significant advantage for training more agents, each one featuring a specific driving style. We explored this by specifying different weights for the rewards and penalties with which the agent is trained, as shown in Table 3.3. In all cases,the training episode was interrupted at the occurrence of a collision, thus preventing the agent to get any further reward for that episode.

The training of the comfort driving style is characterized by a high penalty to prevent collisions and a high reward to encourage the EV to drive in the rightmost lane. On the other hand, the aggressive behavior's training provides no collision penalty and a high-speed reward. Finally, the standard driving style is trained as described in the previous sub-section.

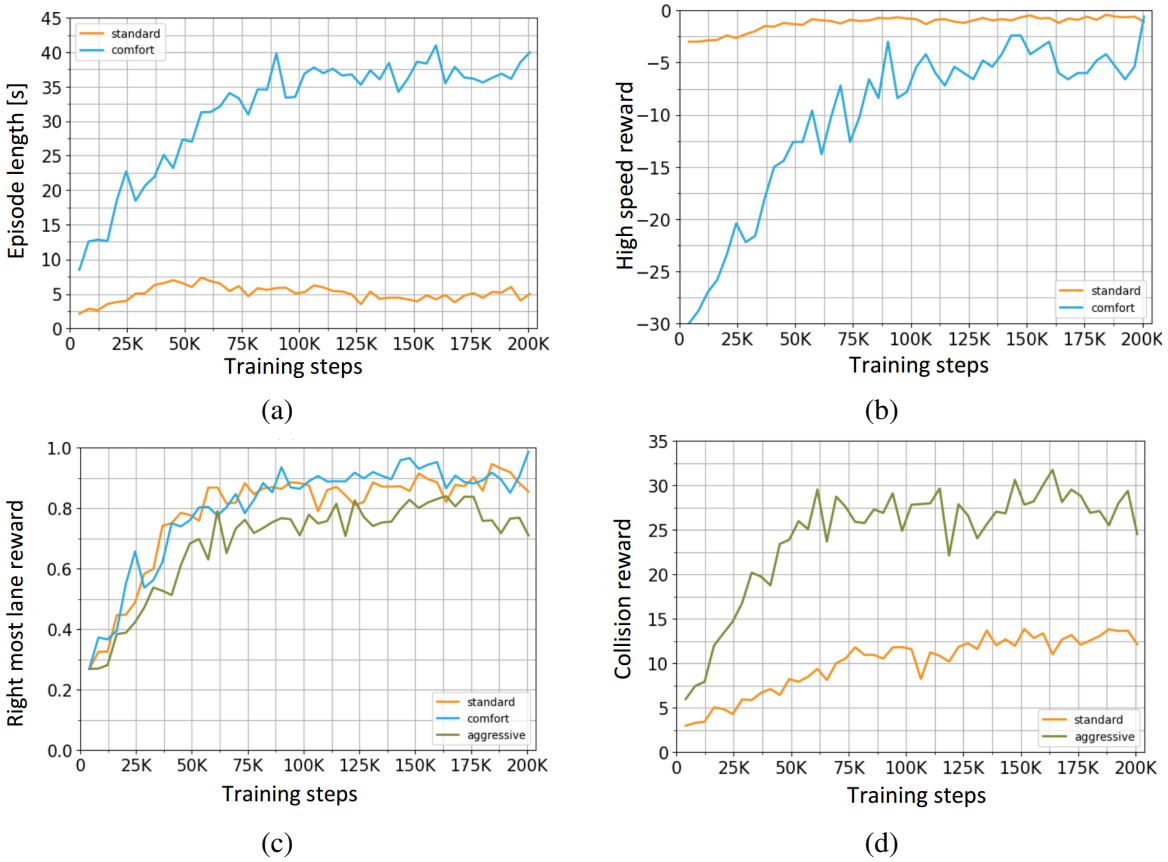


Fig. 3.5 Evolution of the training for the different driving styles. (a) Episode length, normalized from 0 to 1 (i.e., 60 seconds), (b) High speed reward (not assigned to the comfort model), (c) Right-most lane reward (not assigned to the aggressive model) and (d) collision reward (not assigned to the aggressive model)

Fig. 3.5 shows the evolution during the training of four different quantities: episode length, high-speed reward, right-most lane reward and collision reward (actually, penalty). Zero-weighted rewards are ignored.

Fig. 3.5a shows that all the three models tend to achieve a similar value of episode length, which indicates the overall importance of completing each episode (i.e., avoid collisions). However, Fig. 3.5b, 3.5c,and 3.5d indicate that this is achieved through different types of behavior. Fig. 3.5b shows that the high-speed reward gained by the aggressive agent is much higher than for the standard agent, since its weight is twice as much (0.8 and 0.4, respectively; Table 3.3). Even more apparent, the right-most lane reward for the comfort driving agent increases significantly, unlike the standard model, as its weight is four times larger (0.8 vs. 0.2) (Fig. 3.5). Fig. 3.5d, on the other hand, is significant, since it indicates that designing the reward factors may be tricky. Particularly, we see that, while the training of the standard agent is smooth, the training of the comfort agent is more complex, since the huge collision

penalty puts a burden that lasts for a long time. This spurred us to consider an optimization to the training process, that we will present later in this section. It is important to highlight that the episode length results in Table 3.4 differ a bit from those in Fig. 3.5a because they are obtained in exploitation. In contrast, Fig. 3.5a refers to training, which also includes a fraction of environment exploration, leading to lower results.

Table 3.4 Performance over 1000 episodes of the different driving styles obtained from scratch.

Model's style	**Avg. ep. length [s]**	**Collision rate**	**Avg. km travelled**	**Avg. speed [m/s]**	**Avg. decel. [m/s^2]**	**Avg. acc. [m/s^2]**	**Avg. dec. change**	**Avg. left change**	**Avg. right change**
Comfort	60	0.5%	1.43	**23.9**	**-2.76**	**2.46**	**3.61**	0.01	**0.32**
Standard	59	1.2%	**1.82**	30.6	-4.48	2.29	8.22	0.52	0.74
Aggressive	49	**29.2%**	1.62	32.8	-4.89	2.15	12.46	**1.68**	1.69

Table 3.5 Performance over 1000 episodes of the different driving styles obtained from curriculum learning.

Model's style	**Avg. ep. length [s]**	**Collision rate**	**Avg. km travelled**	**Avg. speed [m/s]**	**Avg. decel. [m/s^2]**	**Avg. acc. [m/s^2]**	**Avg. dec. change**	**Avg. left change**	**Avg. right change**
Comfort	60	1%	1.43	**23.9**	-2.88	2.59	6.22	**0**	0.45
Standard	59	1.2%	1.82	30.6	-4.48	2.29	8.22	0.52	0.74
Aggressive	**55**	**11.9%**	**1.92**	33.3	-4.76	1.83	**13.15**	1.14	**0.96**

Table 3.4 presents some typical vehicular metrics that we analyze in order to check whether the different training actually led to their respectively targeted driving styles. Particularly, it is apparent that the comfort model tends to drive in the right-most lane, never overtaking, at a much lower speed than the other driving styles. Moreover, considering the number of decision changes, the comfort model seems to follow a stable policy. On the other hand, the aggressive model features a much higher collision rate,average speed and decision change rate. Overall, these two models look less balanced than the standard one,which drives at high speed, with few collisions.

Until now, we have trained each one of the three different agents from scratch. But this has involved some inconvenience, as shown in Fig. 3.5d. In this last part of the section we explore an alternative technique, such as CL [12]. Thus, we tried to train the comfort and aggressive models (i.e., using the specific weights of their RF),starting their learning phase

from a standard model (i.e., the weights of the DNN of the comfort and aggressive agents are initialized with the values of the weights of a fully trained standard model), thus embodying its previous knowledge.

We explored CL applying the differentiated RFs for half the number of training steps (100k, compared to the 200k employed for training from scratch). Table 3.5 shows that the CL models are more closely related to the standard driving policy than those trained with the classical approach. For instance, a CL aggressive agent features a much lower collision rate and higher mileage. Similarly, the collision rate and decision change rate of the CL-trained comfort model are quite higher. These differences highlight that the CL strategies are learned atop of the baseline model, while the "from scratch" trained agents build up their experience during training without any prior influence.

3.3 Conclusions

This work has explored the training of DRL models deployable as realistic NPVs in driving SGs. We have introduced a hierarchical architecture for behavioral planning of vehicle models, in which the agents decide their motion by taking "human-like", high-level decisions, such as "keep lane", "overtake," and "go to rightmost lane". This is similar to a driver's high-level reasoning and takes into account the availability of ever more sophisticated ADAS in current vehicles. Compared to a low-level decision making system, our model performs better both in terms of safety and speed. As a significant advantage, the proposed approach allows reducing the number of training steps by more than one order of magnitude.

As a second main contribution to advance the state of the art, we demonstrated that it is possible to train agents characterized by different driving behaviors, by tweaking the weights of the factors of a general RF.

To optimize the training, we employed the CL technique, starting the training procedure of a more specialized agent from a base model rather than from scratch. Results indicate that CL allows achieving the same performance, but in much less training iterations (thus time). However, the specialized agents tend to significantly "inherit" behavior from the baseline agent, which may not always be desirable. The method will be transferred to complex simulated driving frameworks for evaluation in more realistic scenarios in the next chapters.

Chapter 4

Decision making in high-fidelity environments

After studying decision making in the highway context and proving the capability of DRL agents to solve a complex task such as AD in a benchmark simulation context, we now move towards more complex and high-fidelity synthetic driving frameworks. Here, we apply similar DRL training methods, such as CL, to tackle the problem of AP. The goal of the studies presented in this chapter is to develop sophisticated simulation environments and to generate effective DRL policies that guide agents to stable solutions.

To achieve this goal, in Section 4.1 we propose a Unity-based environment and train an agent to solve the AP task and challenge the policy in multiple obstacle avoidance problems to stress-test its trajectory stability and safety. Afterwards, in Section 4.2, we deploy a similar CL strategy in the CARLA driving simulator after developing an environment leveraging the Unreal-CARLA open-source framework. This allows for enhancing the level of realism provided by the simulation thanks to CARLA being purposefully designed for a close-to-real-world physical and dynamical agent-environment interaction in the driving context. Finally, in Section 4.3, the previously proposed CARLA environment is expanded upon by introducing new parking variants to enhance the generalization capabilities of learning DRL agents. Here, we validate the new environments by training policies for all the scenarios and evaluating them in terms of parking success.

Collectively, these works serve as a natural step forward towards the generation of stable DRL-based decision making agents in more complex and diverse simulation scenarios.

4.1 DRL-based automated parking in Unity

Building on the decision-making task addressed in Chapter 3, the focus now shifts from highway driving to low-speed maneuvering in structured environments. Whereas highway scenarios emphasize dynamic interactions and continuous control at higher speeds, parking tasks place greater importance on spatial precision and maneuvering in confined areas. AP therefore provides an ideal first step for extending the study of DRL agents to new conditions. The first study in this chapter investigates AP using a Unity-based framework. Unity offers flexibility in scenario design and rendering, making it suitable for testing DRL agents in garage parking and static obstacle avoidance tasks under controlled but visually rich conditions. The main novelty of this work lies in the release of open code and data to support DRL agent development for AP in Unity [29].

Several research questions are explored: What development challenges arise when implementing AP in Unity? Can mapless navigation scale to path lengths of tens of meters? How does training duration evolve under CL? How are computational loads distributed between CPU and GPU? What degree of generalization do Unity-trained agents achieve, and how do they compare to a classical baseline such as Hybrid A*?

This section introduces the Unity environment, the DRL agent design, and the evaluation of training strategies with respect to these questions.

4.1.1 Method

Unity ML-agents

We developed our agents in Unity, a popular cross-platform game engine that is used for the efficient development of games and simulations in various industrial domains [2]. Machine learning agents (ML-Agents) [52] are an open-source toolkit that allows the use of Unity as a simulation environment to create and train autonomous agents. ML-Agents provides a python application programming interface (API) to an implementation, based on the PyTorch library, of the main RL algorithms. These are the key components of ML-Agents:

- Learning environment: This is the Unity scene providing the environment in which the agent observes, acts, and learns. The ML-Agents Toolkit SDK allows wrapping any Unity scene as a learning environment, defining the agents and their behaviors. In a learning environment, it is possible to train more than one agent concurrently, significantly reducing the training time. The training area must of course be built so that the agents do not interact with each other.

- Mlagents-learn: This is the ML-Agents' core utility, which manages the training. The utility is launched with a .yaml configuration file. The file is divided into several sections: behaviors (specifies the training algorithm and its hyperparameters), environment (specifies the environment path, the environment arguments, if any, and the number of environments to be parallelized), engine (specifies the rendering settings, i.e., the screen dimension, the render quality, the time scale, and whether to render the scene or not), checkpoint (contains info about the creation of checkpoints during the training), and torch (specifies whether to use the CPU or the GPU for the training).

- Python low-level API: This API allows communication with the Unity scene during training.

- External Communicator: This component, which is internal to the learning environment, allows communication with a Python API.

- Python trainers: contains the algorithms used to train the agents. It provides a command-line utility (namely, mlagents-learn) and is interfaced with the python low-level API only.

Fig. 4.1 shows the architecture of an example ML-Agent learning environment. In the example, two agents (namely, A1 and A2) are trained by the python trainer through a python API. The communicator module connects the agents to the python API while also retrieving the environmental parameters needed for training (e.g., the target coordinates and the agent location). For the sake of completeness, two other agents are included in the environment. The first one is an agent featuring a policy implemented through a neural network (NN), which was already trained in a previous RL session. The second is a heuristic agent, which implements a behavior determined by a set of heuristic rules.

Fig. 4.2 provides an overview of the typical ML-Agents project workflow. The initial design and implementation step concerns the definition of fundamental aspects such as simulation settings; observations, actions, and reward signals; and NN hyperparameters. The subsequent training consists of executing several simulation episodes, with possible tweaks (e.g., in the RF). When the per-episode reward gained by the agent reaches a suitable level, the training is concluded, and the learned policy can be tested via pure inference. If the test is successful, the model can be deployed; otherwise, training has to restart with a model updated by the designer according to the gained experience.

The workflow for each episode (including the setting of the parameters that are randomized to stimulate the agent's generalization capabilities) is sketched in Fig. 4.3. In the first phase, the environment is initialized according to the user definitions, and the vehicle and

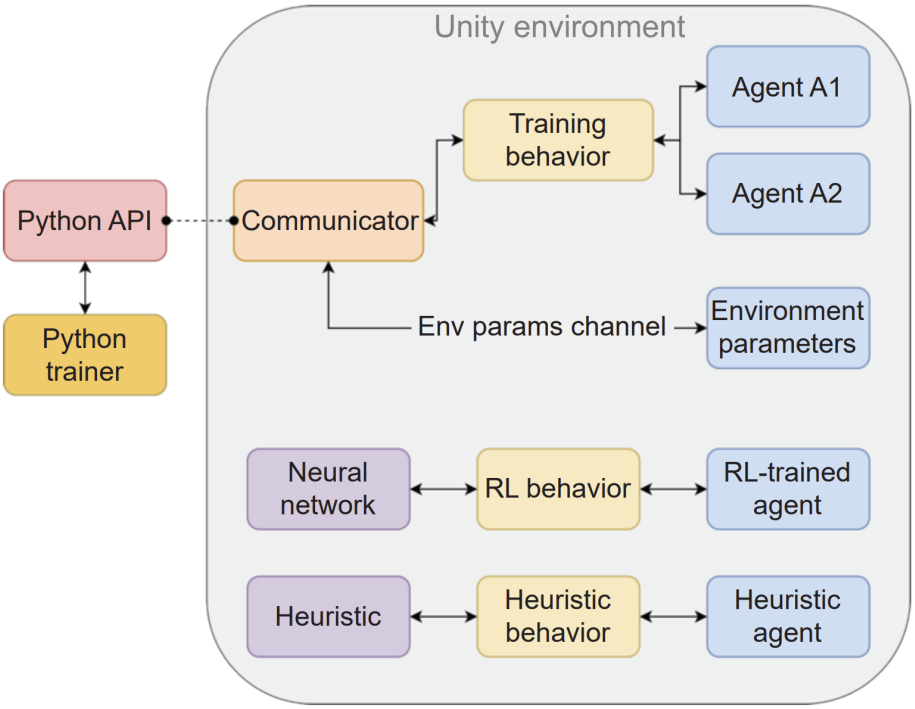


Fig. 4.1 Example of an ML-Agent learning environment.

target positions are randomly generated. Then, the episode loop performs the typical RL loop with the choice of an action by the agent and the consequent modification of the environment. The new PPO iteration is fed by the observation of the environment and the gained reward. Finally, the episode ends either by a collision, a timeout or the reaching of the goal.

Agent details

To implement the vehicle that hosts the DRL agent, we exploited a simple 3D model available at [25]. The vehicle's collider component, which allows the detection of collisions, has the shape of a parallelepiped, with a size of 4.90×1.90 m. Given the scope of the low-speed maneuver, we employ a simple kinematic bicycle model for the vehicle [101], with Eq. (4.1) (see also Fig. 4.4):

$$\begin{cases} \dot{X} = V\cos\big(\psi + \beta(u_2)\big) \\ \dot{Y} = V\sin\big(\psi + \beta(u_2)\big) \\ \dot{V} = u_1 \\ \dot{\psi} = \dfrac{V}{l_r}\sin\big(\beta(u_2)\big) \end{cases} \tag{4.1}$$

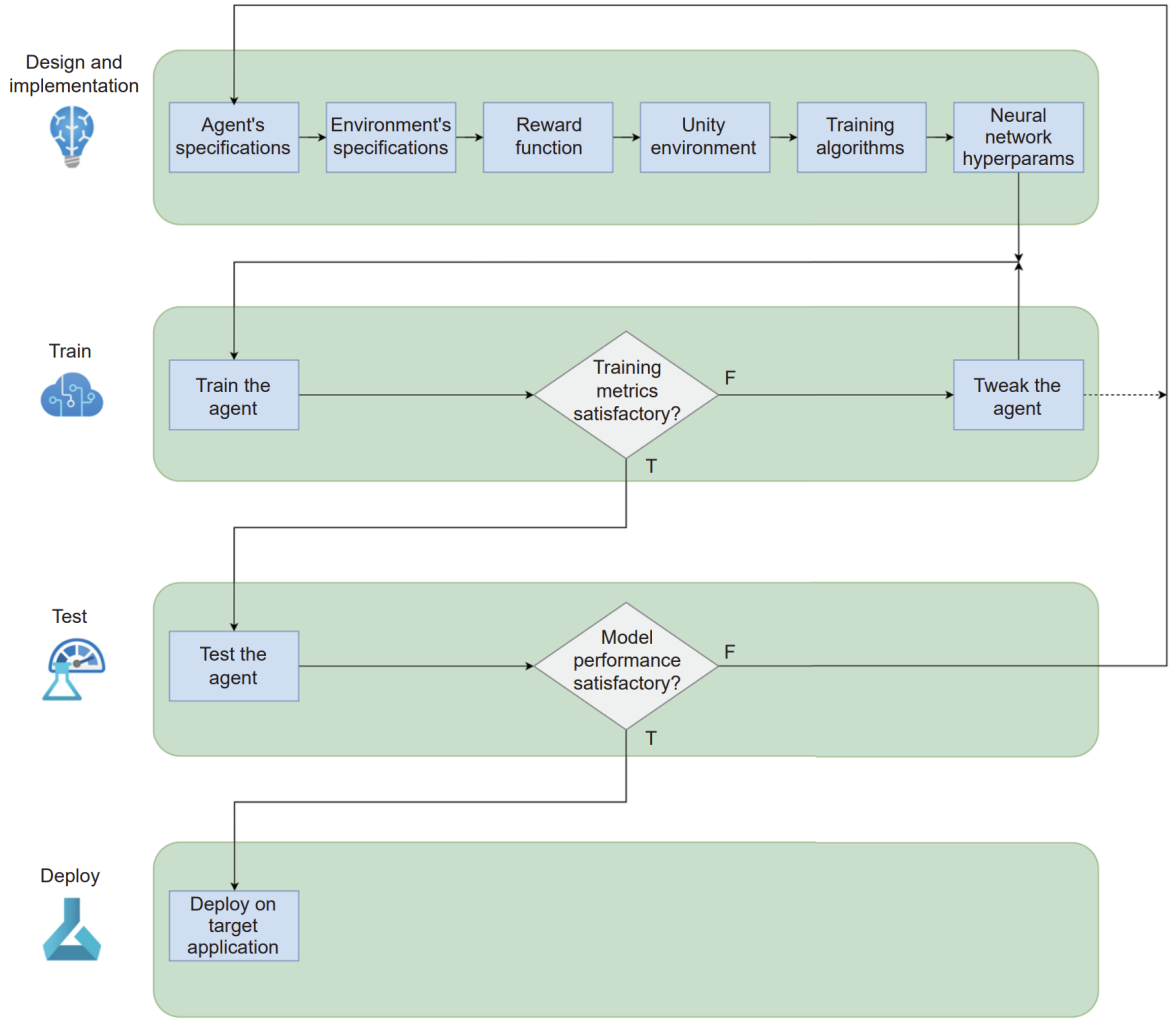


Fig. 4.2 Typical Unity ML-Agents' project workflow.

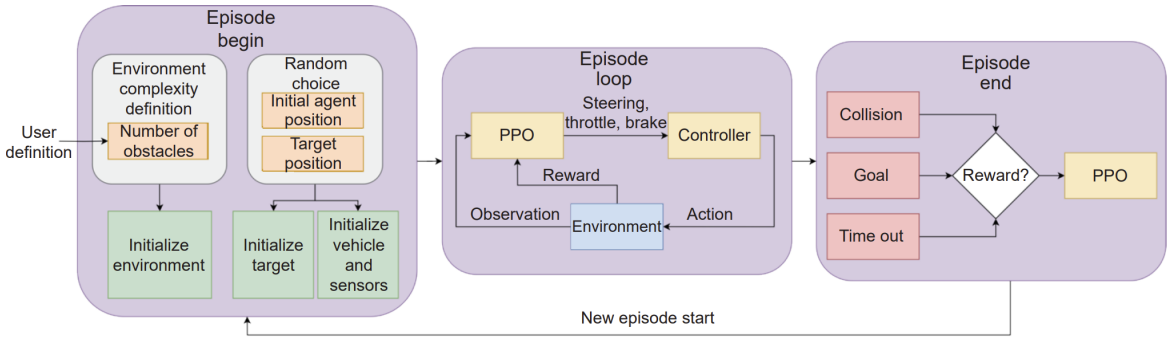


Fig. 4.3 ML-Agents' training episode workflow.

where u_1 represents the acceleration command and u_2 is the front wheel angle used as a steering command. The slip angle at the center of gravity is defined as $\beta(u_2)$. The agent is

supplied by the environment with information about the position of the vehicle and the target, and the speed vector. Moreover, it receives a stream of data from a lidar and or a camera.

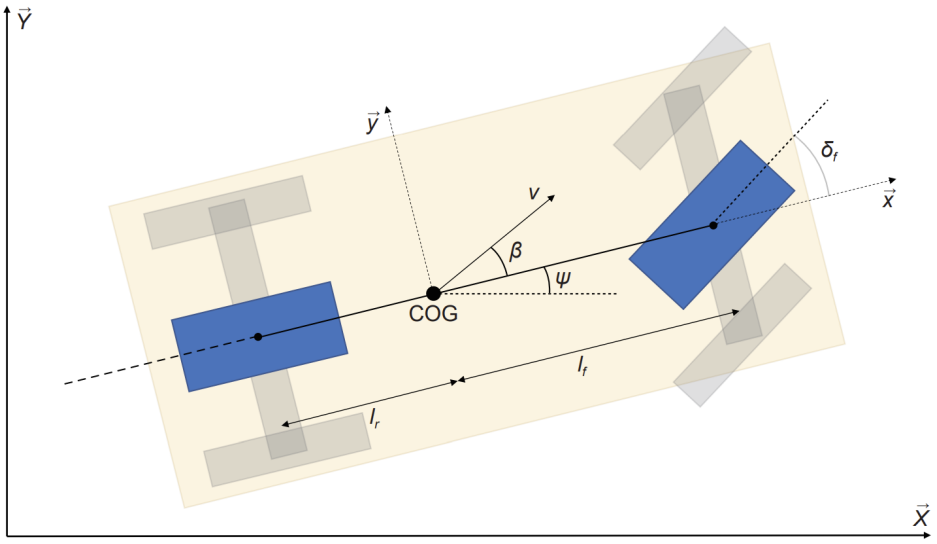


Fig. 4.4 Kinematic bicycle model.

The lidar, with a 360° field of view and an angular resolution of 10°, depicted in Fig. 4.5, is placed on the top and in the center of the roof to produce a 1D array. The radius length is 5 m (we intentionally adopted such a short range to try to enhance the generalization capability of the agent). The camera, placed over the windscreen, has a field of view of 60° vertically and 120° horizontally and a resolution of 84×84 pixels. The lidar is implemented through the ML-Agents raycast sensor component and the camera through a camera component. The raycast sensor provides a vector containing the distances (d) of the last stacked observation (s) for each of the n rays.

The camera sensor provides a 2D vector (size x times y) of the recorded image. The complete observation vector is provided in Eq. (4.2):

$$\begin{cases} S_{\text{lidar}} = (n, d, s) \\ S_{\text{camera}} = (x, y) \\ S_{\text{env}} = \left(\text{agent}_{\text{vel}}, \text{agent}_{\text{dir}}, \text{target}_{\text{dir}}, \text{distance}_{\text{agent}\rightarrow\text{target}}\right) \\ S = \left(S_{\text{env}}, S_{\text{camera}}, S_{\text{lidar}}\right) \end{cases} \tag{4.2}$$

The agent can control the motor torque, the brake torque, and the direction of its front wheels. The steering angle range is $[-\pi/6, \pi/6]$ rad, the brake torque range is $[0, 300]$ Nm, and the motor torque range is $[-400, 400]$ Nm, allowing the vehicle to perform maneuvers forward and backward. The NN model chosen is a multilayer perceptron (MLP) with two

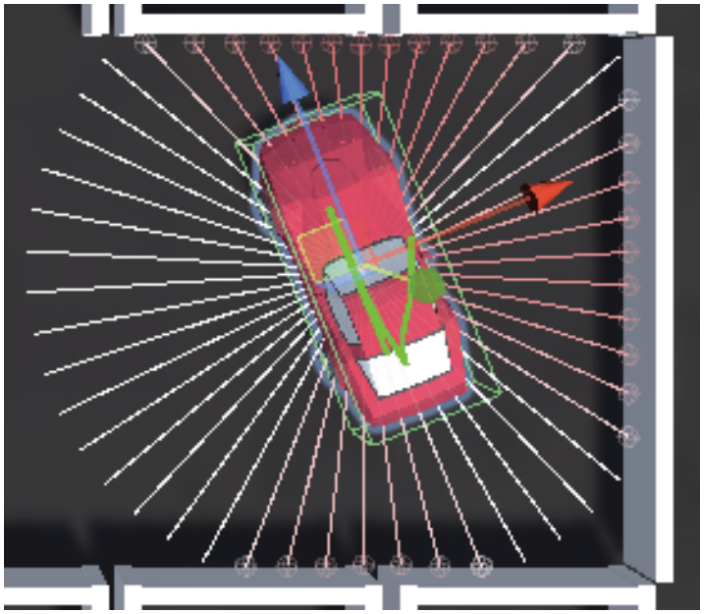


Fig. 4.5 Lidar sensor placed at the center of the car agent.

Table 4.1 Characteristics of the NN architecture.

Layer	**Size**	**Description**
Input	(408 or $84 \times 84 \times 3$) + 8	408 lidar values or an 84×84 RGB image + 2D values of distance between agent and goal, agent's speed, agent's heading, and goal's heading
Convolutional layer	1^{st} Kernel size = [8,8] 1^{st} Stride size = [4,4] 2^{nd} Kernel size = [4,4] 2^{nd} Stride size = [2,2]	2 convolutional layers to pre-process the camera input, when provided
Dense layer	256	2 layers
Output layer	3	Possible agent's actions: throttle, steering, brake

hidden layers of 256 neurons each. When the camera sensor is present, the NN has two additional convolutional layers to preprocess the visual signal. Table 4.1 synthesizes the values characterizing the network architecture.

Fig. 4.6 Bird-eye view of the 3D parking environment.

Parking in a garage

Using 3D settings, we built from scratch a Unity scene representing a garage environment with ten parking lots in a 400 m2 area (Fig. 4.6).

Each lot, 5.3×3.5 m in size, is delimited by walls. The central corridor, which is drivable in both directions, is 20×8 m. At each episode, only one parking lot is open. To speed up the training, we replicated the same environment 10-fold in a single scene, allowing the use of up to 10 vehicles simultaneously. The goal of the DRL agent is to define and execute a path from a random initial position to the target lot, avoiding any possible collision with the walls. A key factor in autonomous agent design is represented by the RF, which is the means through which the environment shapes the behavior of the agent. We started by reproducing the RF (Eq. (4.3)) of a popular open-source project based on ML-Agents [106]. This reward penalizes the agent for collisions and rewards it when it is moving and when it reaches the target. Alignment of the vehicle to the parking lot is also rewarded. Here, θ is the angle between the ego heading vector and the ego-target conjunction vector.

$$\text{reward} = \begin{cases} 0.2\,|\text{alignment}|, & \text{goal and car parked facing the wall} \\ 0.8\,|\text{alignment}|, & \text{goal and car parked facing the road} \\ -0.01, & \text{collision} \\ 0.001\,\text{target}_{\text{dir}} \times \text{target}_{\text{velocity}}, & \end{cases} \tag{4.3}$$

Table 4.2 Best hyperparameter values for the experimental environment, empirically found

Hyperparameter	**Value**
Batch size	512
Buffer size	51,200
Learning rate schedule	Linear
Learning rate (start)	1×10^{-4}
Time horizon	128

Through various experimental iterations, we refined the RF to achieve better results in terms of both accuracy and convergence time:

$$\text{reward} = \begin{cases} -1, & \text{collision} \\ 0, & \text{goal} \\ c_1 \left[\left(c_2 d^2 \right) + c_3 \dfrac{1 - \cos\theta}{d+1} \right], & \end{cases} \tag{4.4}$$

where $c_1 = 0.01$, $c_2 = -0.01$, and $c_3 = -1.5$.

The function sums two rewards: two sparse rewards (i.e., given when a specific event occurs) and one dense reward (i.e., given at every simulation step). The first sparse reward penalizes the agent in case of collision. The second one rewards the achievement of the goal. An additional goal reward is added proportionally to the final alignment of the vehicle to the parking lot. The dense reward aims at favoring the approach to the goal and consists of two components that correspond to the two targets of final position and final orientation. The first term penalizes the distance from the target, and the second term penalizes the misalignment from the parking lot. The second term is divided by the distance, as controlling misalignment is meaningful only when the vehicle is close to the target. We use the Manhattan distance, which is more appropriate for the environment. The dense reward is normalized in the [-2, 0] interval. The sparse and dense rewards are carefully weighted through the c_1, c_2, and c_3 factors to empirically achieve a good balance.

For the DRL method, we used the PPO, which is the reference for continuous control problems. The best values we could empirically find for the training hyperparameters are reported in Table 4.2.

The scene is reinitialized at each training episode. One parking lot is selected as the target, and its door is opened; the agent is set in a random position and orientation in the central corridor. An episode ends when one of these conditions occurs: (1) the vehicle reaches the goal; (2) the vehicle collides with an obstacle (this criterion is not applied in the initial

training phase, when we prefer to let the agent continue the exploration after a collision); or (3) the maximum number of steps for an episode is reached without arriving at the goal.

We considered the following performance metrics for the training phase:

- *Cumulative reward.* It is the sum of all the rewards accumulated by the agent in an episode. The evolution in time of this metric is a major indicator of the effectiveness of the training. As rewards are given only during training, this quantity, unlike the next quantity, is not applicable in testing.
- *Goal rate.* The ratio between the number of episodes that ended up reaching the goal and the total number of episodes.
- *Collision rate*. The ratio between the total number of collisions and the total number of episodes. As we will see, in some cases, collisions are terminal; in others, they are not, depending on the training policy design choice.
- *Timeout rate*. The ratio between episodes ending with a timeout (i.e., doing the maximum number of steps without reaching the goal) and the total number of episodes. For our analysis of the garage scenario, we conducted three subexperiments, which are described below.
- *Subexperiment No. 1*: There is a single target lot for all the training episodes, thus with limited generalizability. Collision is not a terminal event. The vehicle is equipped with a lidar.
- *Subexperiment No. 2*: The lidar is replaced with a camera, and the NN architecture is modified to manage the different inputs. Specifically, two convolutional layers are added to process the camera signal, and the output is concatenated to the other inputs.
- *Subexperiment No. 3*: The vehicle is equipped with a lidar and trained to reach one of the ten possible parking lots, which are randomly selected at the beginning of each episode. A collision is not a terminal event until the success rate reaches a certain threshold; then, the collision is set as a terminal event.

For all the subexperiments, the testing phase consists of 100 episodes in which the vehicle is spawned in a random position and orientation and has to reach one randomly selected lot among the 10 available. The relevant results are reported in Table 4.3.

Regarding the distribution of computations during training, in this experiment (and in the other experiments), we did not notice a significant improvement by using a dedicated GPU. We argue that this is because the NN used is not so heavy in terms of depth and neurons.

Moreover, RL is CPU intensive due to sequential agent-environment interactions (as detailed in Section 2.2, and rendering is suppressed when a camera is not used. The training speeds measured are approximately 1.4 M step/h, using a device equipped with an Intel Xeon W2223 processor, 32 GB of RAM, and an NVIDIA Quadro RTX 4,000 GPU.

For subexperiment No. 1, we noticed a critical phase in the first 8 M steps, with low/unstable reward and episode lengths dominated by collisions and then by timeouts. In particular, we observed that in several cases, the vehicle repeatedly executed sudden gear inversions, substantially remaining in the same position. This corresponds to a local minimum, which prevents the agent from improving its performance. After 8 M steps, the agent performance quickly reaches a stable state with a high reward and short episode lengths. However, the test phase, where the target can be any of the 10 lots, reveals that the trained agent is not able to make a proper generalization (Table 4.3).

In the second subexperiment, we substituted the lidar with a camera. The training proved effective. However, in this case, the agent's generalization capability to multiple target test cases is not satisfactory (Table 4.3). We thus tried to restart the training by randomly simulating all the possible target lot cases, which slowed learning without leading to significant improvements. Additionally, the idea of penalizing the timeouts did not provide significant advantages.

In the third subexperiment, we returned to use a lidar instead of a camera, which reduced the input size and the training time. We also changed the values of the constants and normalized the rewards. For the tuning of such hyperparameters, it was very important to create and analyze additional TensorBoard charts, as shown in Fig. 4.7. In particular, we logged in Tensorboard the value of all the constituents of the RF (i.e., collision, goal, distance, and alignment) to quantitatively check agent behavior. This was key to efficiently assessing various alternatives and tuning the RF and simulation settings, which is a very time consuming task.

This analysis was also very useful for tuning the CL. CL in RL is tricky because there is no predefined dataset, and the difficulty of evaluating samples can be evaluated by a difficulty measurer [134]. Thus, similar to [6], we used a one-pass algorithm [12], with increasing difficulty levels associated with more complex versions of the learning environment. We

Table 4.3 Testing results of each experiment.

Subexp. No.	Sensor	Number of target parking lots in training	Success rate in test
1	Lidar	1	50%
2	Camera	1	80%
3	Lidar	10	94%

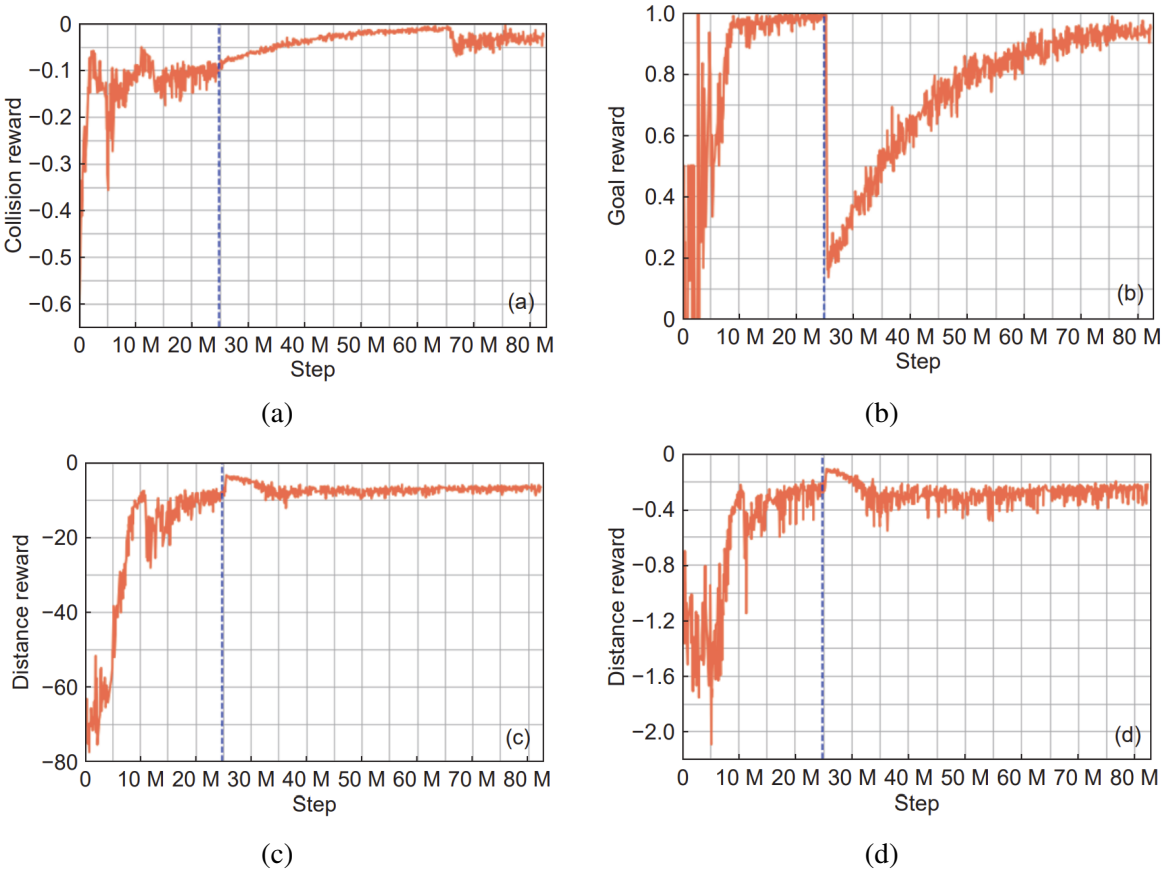


Fig. 4.7 Tensorboard plot of the single components of the reward function in the third subexperiment: (a) the collision reward, (b) the goal, (c) the distance, and (d) the alignment. The dashed blue lines indicate the switch from the first to the second stage.

could have also gradually increased the agent's capabilities in different training stages, but preliminary results showed that the increased training complexity did not provide benefits. In contrast to [6], we did not set a predefined number of steps for each stage but adopted a performance criterion [134]. Moreover, at each stage, we not only increased the context complexity (e.g., type of goal, condition of episode termination) but also tweaked the agent's RF by carefully analyzing the Tensorboard charts of various training attempts, as mentioned above. For this first experiment, we implemented two stages. In the first stage, the agent is trained to reach a single target lot. Collisions are penalized but do not terminate the episode to allow the agent to explore the environment without frequent interruptions [61]. When the goal is achieved with a rate greater than 99.5%, the second stage starts. In this stage, the other 9 targets are randomly activated, collisions become terminal events for an episode, and the collision penalty is increased (the reward decreases from –1 to –10). The introduction of these factors causes the sharp performance drop of the goal reward that we can observe

at the stage transition in Fig. 4.7b. However, the training resumes effectively (even if more slowly because of the much greater complexity of the learning environment) and, after a sufficient number of iterations, allows satisfactory performance in the target environment to be reached.

The first stage is completed at 25 M, while further steps at 57 M are needed for the second stage. Fig. 4.8 reports the evolution of the success rate (i.e., goal reach, without collisions) during training, with the above mentioned sharp performance drop at the change of the stage. The test results show that training on multiple targets appears to be necessary for an agent to reach multiple targets (Table 4.3), which means that training on a single target causes the agent to overfit the actual position of the target.

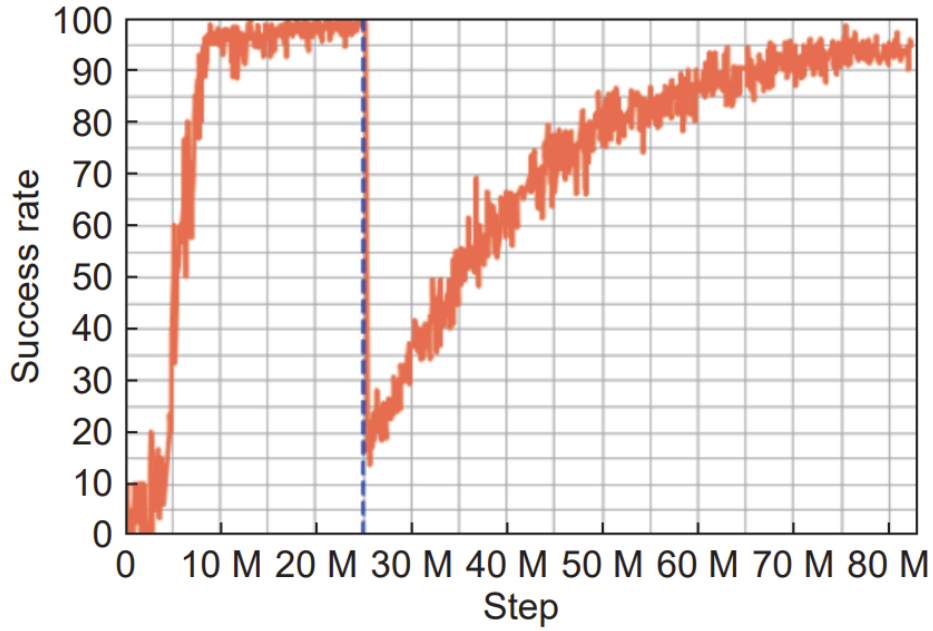


Fig. 4.8 Evolution of the training (success rate) in the third subexperiment. The dashed blue line indicates the switch from the first to the second stage. The sharp performance drop is due to the increased collision penalty, and the episode stops in the case of collision.

We assessed the impact of CL by comparing the achieved results with those of an agent's training starting from the beginning with the same conditions and rewards of our stage 2 agent. Impressively, even after 80 M iterations, the success rate persistently remains at 0. We argue that a pivotal factor for CL is the possibility of having initial episodes with non-terminal collisions. This not only promoted exploration of the environment (characteristics of the movement area and effects of the actions) but also exposed the agent to the rewards associated with reaching the target. On the other hand, the agent heavily penalized since the beginning of collisions learned that staying was still more advantageous than moving in a

risky environment. Thus, CL fosters an explorative learning process, steering the agent away from overly cautious behaviors and toward a more adaptive learning strategy.

We compared the test results from the third subexperiment with those of Hybrid A*. Hybrid A* is a variant of the well-known A* search algorithm, applied to the 3D kinematic state space of the vehicle, which guarantees kinematic feasibility of the path through a state-update rule that takes into account the continuous state of the vehicle in the A* nodes [26], [99].

As a Unity implementation of the Hybrid A* algorithm is available open-source [89], we applied it to our test environment and achieved results that are presented in Table 4.4.

To generate the path, Hybrid A* discretizes the map into 25 cm squared cells. Every cell in the map represents a node with an associated cost in the path generation process. A heuristic function accounts for the vehicle's possible movements, considering a discrete set of actions given by three steering angles: [–30°, 0°, 30°]. A flow field around the obstacles is generated to avoid collisions. The value of its extension can be controlled by modifying the safety margin. The results show that Hybrid A* achieves a 100% success rate, which decreases to 85% in the actual path following phase because of the discretization of the map (Table 4.4). Performance measured on an Intel Xeon W2223 machine with 32 GB of RAM indicates that more than 0.8 s are needed to compute the path (which is not needed by the DRL agent). Achieving more real-time performance with Hybrid A* (almost 0.2 s) requires increasing the size of the cells to 1.2 m, which, however, leads to a decrease in the target reach rate to 21%. Table 4.4 also shows that the DRL agent can reduce the number of gear inversions needed to reach the target by 25% on average. Thus, the DRL agent achieves better performance (goal reach rate, gear inversion rate, latency) and does not need a map.

As important lessons learned, our experience stresses that some factors are key to properly setting up the environment and avoiding systematic (and difficult-to-spot) errors. The simulation tick (i.e., the time between two consecutive updates of the environment) should be fast enough to guarantee a faithful physical simulation (otherwise, some collisions and/or goal reach may be detected too late, if not missed) but should be tuned according to the actual speed of the simulated items and the CPU capacity. The decision period (i.e., the number of ticks between two consecutive agents' decisions) is the other time-related setting that needs careful tuning. A short decision period leads to the vehicle being stationary. Our

Table 4.4 Comparison of 100 episodes of Hybrid A* PPO and DRL PPO in the Garage environment

Path planning algorithm	Success rate	Gear inversions per episode	Time for path planning
Proposed DRL	94%	3.36	-
Hybrid A*	85%	4.48	0.857s

interpretation is that, in this case, the agent does not have the time to see the effect of its actions. With a too long period, on the other hand, the agent becomes insufficiently reactive to properly manage the vehicle's dynamics. To speed up the training phase, we used the possibility offered by ML agents running multiple training environments at the same time. This increases the ratio of steps/second and the load on the CPU. For this reason, the number of parallel environments should not be too high because after a certain (machine-dependent) threshold, the step/second ratio begins to decrease.

The choice of the sensor(s) has a strong impact on how the agent learns and behaves but also on the training time. The decision between a camera sensor and the lidar has a strong impact because the camera requires the rendering of the scene at each timestep, reducing the number of instructions per second and increasing the training time. A camera also requires a larger NN (two convolutional layers were added to process the signal), which also impacts the training time. Since the goal-reach results between the camera and lidar for our experiments were similar, we decided to prioritize the second one. Concerning generalization, we noticed some clear limits: when something changes in the configuration (width or length of the driving area, position of the parking lots, etc.), performance drops significantly, and the agent needs retraining.

Random obstacles

The previous experiment trained an agent to reach any target parking lot, starting from a random position in a garage. As a second investigation, we explored an environment characterized by the presence of random obstacles before the target, with longer origin–destination distances. To analyze this, we set up another Unity environment, which is analyzed in this subsection.

The input type, neural network configuration, and RL training method are the same as those for Garage subexperiment No. 3. The goal of the agent is to reach a target random position in an 80×80 m space, avoiding collisions with obstacles placed within the scene. The number of obstacles ranges between 3 and 9, and each obstacle is placed between the starting position of the agent and the goal, with a random orientation. The presence of obstacles aims at adding variety to the environment, requiring the planning of a proper path to avoid them (e.g., Fig. 4.9).

For the RF, we used a similar approach as for parking but also added a low-speed penalty. This addition was chosen to avoid situations, as experimentally observed, in which the agent stops close to the target, without ever reaching it, and is seemingly satisfied with the short distance reached (likely compared to the risk of a collision).

Fig. 4.9 Sample low-speed maneuvering environments, with different numbers of obstacles spawned with random positions and orientations.

Table 4.5 Difficulty levels.

Difficulty level	**Description**
1	No obstacles.
2	One obstacle midway between the spawning point and the target.
3	Like level 2 but with two additional obstacles on the sides.
4	Like level 3 but an additional obstacle is placed in a second series of obstacles.
5	Like level 3 but two additional obstacles are placed in the second series of obstacles.
6	Like level 3 but three additional obstacles are placed in the second series of obstacles.

For the training, we used a one-step CL procedure, as in the first experiment, with increasing difficulty in terms of the number of obstacles present in the scene. Once the agent masters a difficulty level, the training continues in more complicated settings. The difficulty levels are reported in Table 4.5. The weakening of hyperparameters (including the RF) may also occur at these levels, based on the agent's behavioral observations.

The TensorBoard chart in Fig. 4.10 shows the evolution of the training success rate across the different difficulty levels. Regarding the first three levels, it appears that the entry level is accomplished in 11 M steps, the medium level after an additional 36.5 M steps, and the third level is completed after an additional 363.5 M steps. The decrease in performance at each level shows a lack of generalization ability on the agent's side. In fact, at each stage, the agent seems to learn to reach a target with the current obstacle schema but not to reach a target without collisions (which is our ultimate generalization goal). However, the drop decreases

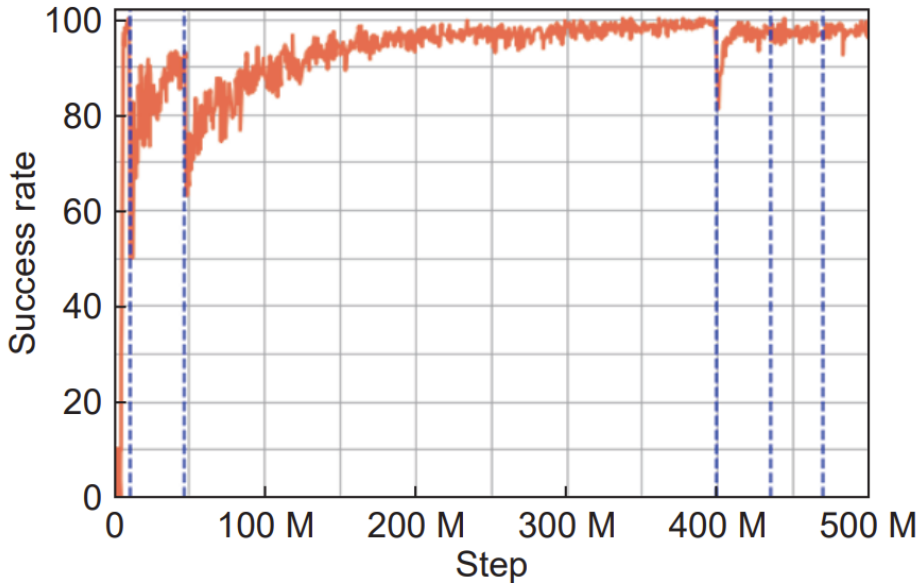


Fig. 4.10 Evolution of the training success (i.e., goal reached) rate at the six different difficulty levels (Table 4.5). Difficulty level switches are indicated by the dashed blue line.

at higher difficulty levels, suggesting that eventually, the agent learns to generalize. As it appears, every level switch requires specific training, which is slowly but clearly achieved by the agent.

Fig. 4.11 shows the performance of a system directly trained at the final difficulty level (Table 4.5). In this case, unlike in garage parking, agents trained with no CL are able to achieve considerable performance, even if they still have a gap with respect to CL (95% vs. 98%). We argue that the difference with the Garage case study could be attributed to the larger spaces between obstacles in this second environment. Although collisions are terminal since the training outset, they are not frequent, so the "novice" agent can explore the environment and learn the task. A comparison with Fig. 4.10, however, shows that the agents start learning much later because of the higher environmental complexity, which would be a problem if fast feedback was needed (e.g., because of investigating different hyperparameters on shared/pay-on-demand hardware).

Additionally, for this experiment, we compare our DRL agent with a Hybrid A* implementation (Table 4.6). Here, Hybrid A* slightly outperforms our DRL agent in terms of the goal reach rate (99% vs. 98%). We argue that this is due to the lower complexity of the maneuvers needed to reach the target. However, reducing the latency from 0.56 to 0.17 s requires increasing the cell side size from 0.5 to 1.2 m, causing a decrease in the success rate to 69%.

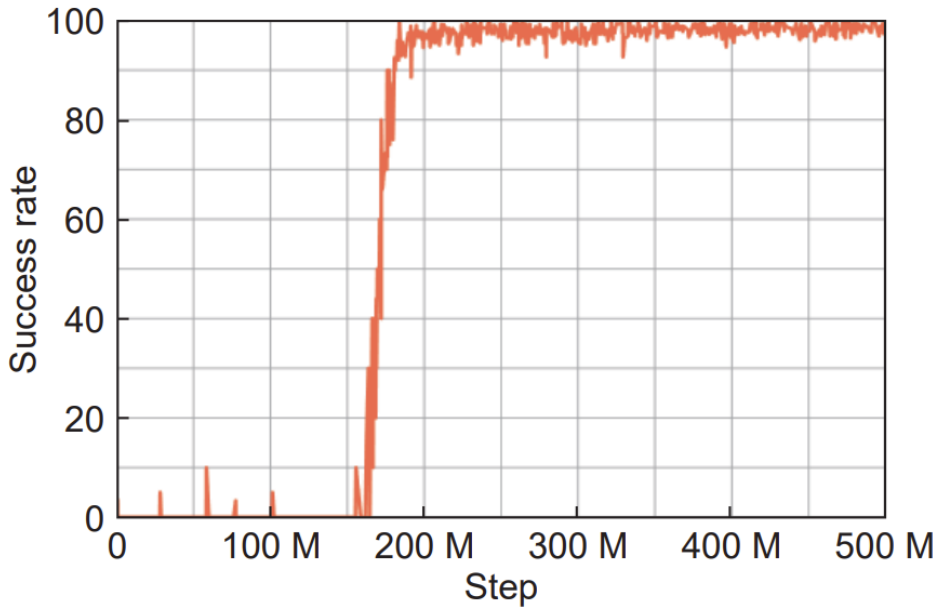


Fig. 4.11 Evolution of the training success (i.e., goal reached) rate for an agent trained directly at the final difficulty level (Table 4.5).

Within this experiment, we also conducted a specific study to better assess the agent's generalizability. After concluding the training at the first three levels, we tested the learned policy in environments with more obstacles. The new obstacles are identified with the same logic as for the first three levels but closer to the target (Fig. 4.9). Table 4.7 reports the results of over 1,000 test episodes of the agent trained using 3 obstacles in different settings.

The outcomes highlight that the additional difficulty is not properly managed by the agent, with a clear increase in the collision rate with the first added obstacle. We thus extended the training to the cases of 4, 5, and 6 obstacles. The results, provided in Tables 4.8, 4.9, and 4.10, respectively, show that training with the fourth obstacle is already almost sufficient for tackling the most complex cases.

Table 4.6 Comparison of 100 episodes between Hybrid A* and DRL PPO in the Random Obstacles environment at the final difficulty level.

Path planning algorithm	**Success rate**	**Time for path planning**
Proposed DRL	98%	-
Hybrid A*	99%	0.561 s

As anticipated, this fact is reflected in Fig. 4.10, which shows the evolution of the training. We see a performance drop at approximately 400 M steps, in which the 4-obstacle training starts, but we do not see significant drops later when switching to training with

Table 4.7 Testing results obtained using different environmental complexities with the agent trained with 3 obstacles.

Env.	Goal ratio	Collision ratio	Timeout ratio
3 obstacles	99.3%	0.7%	0%
4 obstacles	66.6%	33.4%	0%
5 obstacles	65.7%	34.2%	0.1%
6 obstacles	60.8%	39.1%	0.1%

Table 4.8 Testing results obtained using different environmental complexities with the agent trained with 4 obstacles.

Env.	Goal ratio	Collision ratio	Timeout ratio
3 obstacles	96.9%	2.4%	0.7%
4 obstacles	98.2%	1.7%	0.1%
5 obstacles	93.7%	6.1%	0.2%
6 obstacles	93.4%	6.1%	0.1%

5 or 6 obstacles. From Tables 4.8 to 4.10, we also see that a policy trained in a more complex environment also works substantially well in simpler environments. In summary, and particularly referring to Fig. 4.10, given the criticality of the training time, we argue that the agent is (relatively) quickly able to reach the target (11 M steps); then, as an obstacle is added, the reward abruptly drops, substantially because the agent discovers the concept of an obstacle before the target. The agent learns to avoid a single obstacle in 36.5 M steps. Further addition of two lateral obstacles does not degrade the performance as much as in the case of the first obstacle, but it takes the agent many more steps (363.5 M) to slowly learn how to address the increased complexity. We argue that this is necessary for the agent to generalize the concept of obstacle avoidance. Then, for additional obstacles, the drop is relatively smaller (and ever smaller, with increasing complexity), and a shorter tuning stage is necessary to regain the top performance.

This study provides a quantitative measurement of how to abstract the notion of avoidable obstacles, an agent needs training in an environment with a sufficient number of obstacles. This is akin to the concept of class representativeness in supervised learning datasets but in the more complex RL context, where the dataset is built dynamically through the agent's experience.

Table 4.9 Testing results obtained using different environmental complexities with the agent trained with 5 obstacles.

Env.	Goal ratio	Collision ratio	Timeout ratio
3 obstacles	98.3%	1.1%	0.6%
4 obstacles	97.8%	1.5%	0.7%
5 obstacles	98.3%	1.4%	0.3%
6 obstacles	96.3%	3.1%	0.6%

Table 4.10 Testing results obtained using different environmental complexities with the agent trained with 6 obstacles.

Env.	Goal ratio	Collision ratio	Timeout ratio
3 obstacles	98.6%	1.3%	0.1%
4 obstacles	97.6%	2.1%	0.3%
5 obstacles	98.4%	1.5%	0.1%
6 obstacles	97.8%	1.9%	0.3%

4.1.2 Conclusions

The main contribution of this study is a systematic and quantitative analysis of the training of a DRL agent for low-speed AD maneuvering in Unity. In particular, our experience highlights some key factors in successful training. First, CL allows the agent to overcome local minima of a complex loss function (Fig. 1 in [12]). This technique was necessary for the garage case study and highly beneficial for obstacle avoidance. It is also useful for reducing training time and the use of computational resources. It involved carefully modifying, at various stages, factors such as the type of goal, terminal condition, complexity of the learning environment, and RF hyperparameters.

Concerning generalization (i.e., knowledge transfer), the agent can reach different targets by avoiding obstacles (walls or other shapes) with random positions (origin, destinations, and placement of the obstacles) and with difficult maneuvers (parking and narrow spaces among obstacles), provided that it is accurately trained to do that (same type of maneuver, training with random positions, a similar configuration/complexity of the environment). The abstraction of concepts such as obstacle avoidance requires training in a sufficiently complex environment (i.e., an environment with a sufficient number of obstacles).

We have proven the crucial effectiveness of CL for low-speed AD maneuvering. Future research could explore the automation of the process (e.g., difficulty measurement or training scheduling [134]), which would reduce the time spent investigating and tweaking the numerous hyperparameters involved. Other research directions may involve the use of higher-level

AD decision making, multiagent systems [33], [110], and language models for trajectory generation [60], [77], [104]. Finally, we limited our study to static obstacles, for the sake of clarity and because this is typical in low-speed maneuvering, but now the analysis could be extended to dynamic contexts, e.g., with moving vehicles and pedestrians, in more realistic contexts (motorways and urban areas), thus involving other types of maneuvers.

4.2 DRL-based automated parking in CARLA

After examining AP and obstacle avoidance in Unity, the study now progresses to CARLA, a simulator designed specifically for AD research. While Unity proved useful for testing concepts in controlled conditions, its role as a general-purpose engine limits the realism of vehicle dynamics and sensor responses. CARLA, by contrast, integrates high-fidelity physics and sensor models that enable closer alignment with real-world driving.

Within this environment, the objective is to design a DRL agent capable of automated parking by directly controlling steering, throttle, and brake actions, thereby replicating the fine-grained decisions of a human driver. The experiments target low-speed maneuvers in a comb-style parking lot, with non-player vehicles (NPVs) randomly placed to introduce variability and occlusion.

This section describes the CARLA-based framework, details the CL approach adopted for training, and evaluates the extent to which the resulting agents achieve stable and safe parking performance under these more realistic conditions.

4.2.1 Method

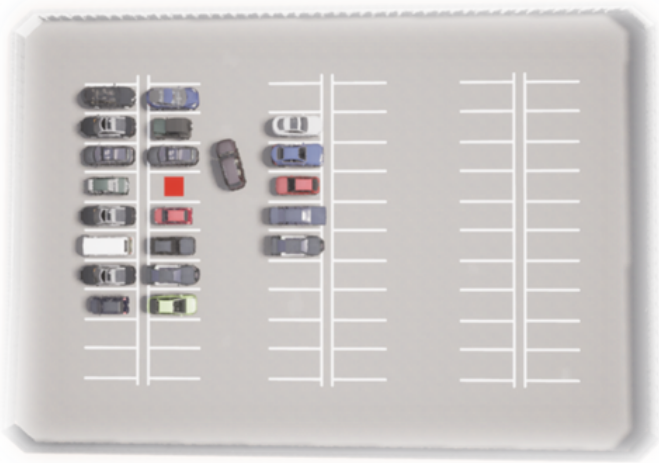


Fig. 4.12 Bird view of the parking area. The red square marks the target parking lot.

To train our model, we took a step toward more realistic simulations than previously done in Chapters 3 and 4.1 for parking and highway driving, respectively. To increase the level of realism in terms of graphics and physics simulation, we opted for CARLA as a driving simulator. It includes several town maps and different vehicle models to which sensors such as radar, lidar, camera, etc. can be attached. In addition, CARLA exposes an API that allows

users to control all aspects related to the simulation, including traffic, pedestrian behaviors, weather, sensors, etc. The official DRL-related section lacks recent updates but some recent works have recently been proposed [95, 67], showing promising results in trajectory tracking applications. Starting from the built-in Town 5, where a parking area is already available, we removed all the unnecessary surrounding entities from the scene, so to obtain a light-weight scenario to reduce the computational cost of rendering and thus training time. The parking area, visible in Fig. 4.12 , measures 50×50 m and is composed of 60 comb parking spaces with a 90° layout, 5×2.5 m each. Brick walls cordon off the area, preventing the agent from falling during training. The weather includes 15 different conditions of sun, rain, fog, and clouds and 41 vehicle models are available, from motorbikes to trucks, with customizable colors.

Experiment

To implement our DRL agent, we opted for Gymnasium (formerly known as OpenAI Gym [13]), an open-source Python toolkit that exposes an API to facilitate the design and implementation of DRL environments which has become a de facto standard for the communication between agents and simulation environments. In addition, Gym offers the ability to define custom environments, so, we wrapped the CARLA parking setting into a Gymnasium environment to get a DRL-compatible interface. Furthermore, Gym is compatible with Stable-Baselines3 (SB3) [105], an open-source Python library that offers a variety of DRL algorithms to set up the training algorithm and the neural network architecture and configuration quickly and intuitively. The schematic of the tools used and their interfacing are shown in Fig. 4.13. At each episode, the vehicle starting point and orientation are chosen randomly within the drivable area. The target to be reached is one of the 60 lots, randomly chosen at each episode. To train our agent, we used the Audi e-tron vehicle model, available in CARLA. The car measures $4.90 \times 1.93 \times 1.62$ m, and has a total mass of 2490 kg. The ego vehicle (EV) is equipped with a lidar sensor placed in the center of the vehicle with a 360° horizontal field of view. The key part in the development of a DRL agent is RF. We split it into several contributions, some dense (given at each simulation step) and some sparse (awarded only when some events occur). Table 4.11 offers an overview the components of the RF. Distance and heading are the two dense contributions to induce EV to approach the target. Collision is sparse, needed to teach the agent to avoid other NPVs and the walls delimiting the area. Two contributions are related to reaching the target parking lot, goal and alignment. The goal reward is awarded when EV reaches the lot, while alignment rewards the agent if it parks aligned well.

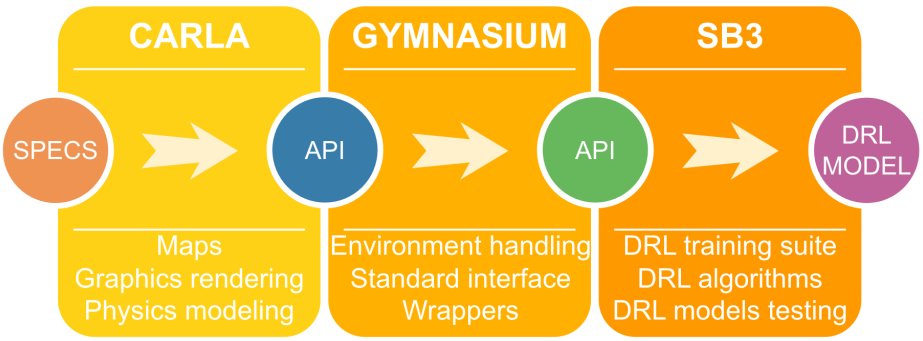


Fig. 4.13 Outline of the tools used for the implementation of the DRL agent.

Table 4.11 RF components

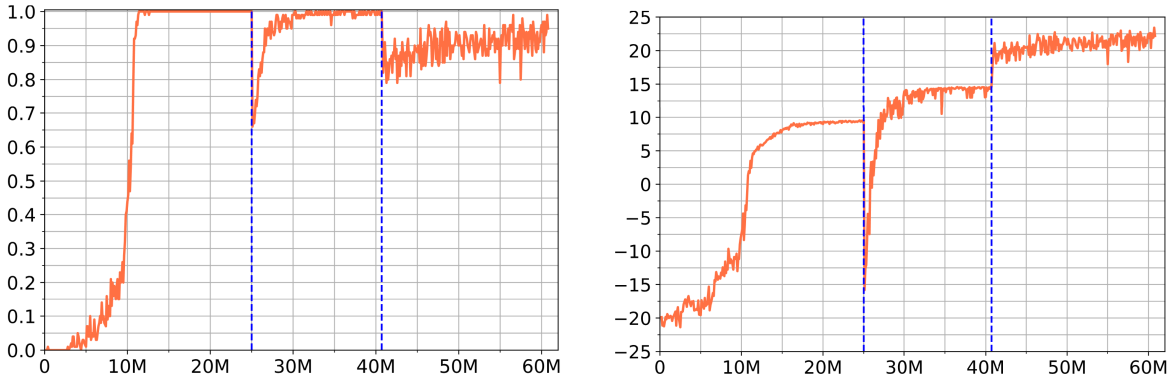
Name	Description	Range	Type
Distance	Euclidean distance from EV to target	[-0.1,0]	dense
Heading	Angle between EV forward vector and EV-target vector	[-0.1,0]	dense
Collision	Collision with walls or NPVs event	[-1,0]	sparse
Goal	Target achieved event	[0,15]	sparse
Alignment	Angle between EV forward vector and parking lot orientation	[0,10]	sparse

For the training of the model, we used the PPO algorithm [112]. As the backbone neural network, we opt-ed for a MLP configuration, composed of 2 hidden layers of 512 neurons, whose input and output values are summed up in Table 4.12. To give the agent the ability to better perceive its motion, at each network update we stacked the last four input values on top of the current one. This allows the agent to capture the dynamics and changes in the environment over time.

Given the results previously obtained in an environment similar to ours, although significantly less complex, such as [61], we adopted a CL strategy for training our agent. In this case, we split the training into three distinct phases. The first phase involves only EV, no NPV is present and an episode terminates if the target is reached or if it goes over 240 steps. Collision is not penalized to incentivize the agent to explore the environment. In the second phase, the number of NPVs is chosen randomly between 0 and 20 at each episode and, again, only the goal reaching or the timeout can end the episode. Collisions are now slightly penalized with a weight equal to -0.1. Finally, in the third phase, conditions are similar to phase 2 but now a collision (penalty weight -1) terminates the episode.

Table 4.12 PPO network input and output.

Type	Quantity	Dimension	Resulting size
Input	LiDAR data	61	340 (5 stacked inputs)
	Distance from target (x, y)	2	
	Speed (x, y)	2	
	Acceleration (x, y)	2	
	Angular velocity (yaw)	1	
Output	Throttle	1	4
	Steering	1	
	Brake	1	
	Reverse	1	



(a) (b)

Fig. 4.14 The PPO agent training over ∼60M steps. (a) success rate, (b) cumulative reward. Dashed blue lines indicate the three CL phases.

4.2.2 Results

We trained our PPO agent for about 60M steps (Fig. 4.14). The first CL phase took 25M steps, after which the agent was able to reach the goal 100% of the time (Fig. 4.14a). Once this is achieved with no NPVs present in the scene, the second phase started, lasting about 15M steps in which the agent learned to park with NPVs parked near the target parking lot. The last phase, in which collisions represented a terminal state, lasted 20M steps, achieving a final success rate of 97%.

Although the success rate is satisfactory, some problems related to the use of CARLA arose during code development. In particular, the lack of direct support for DRL made the implementation of the environment and synchronization with Gym and SB3 particularly

complex. Too high values of the CARLA environment update frequency (greater than 20 Hz) greatly lengthened the training time due to the higher computational load but, conversely, frequencies that were too low (lower than 10 Hz) did not guarantee proper observation of the environment by the agent (e.g., it happened that the agent stepped on the goal without being detected). In addition, it was necessary to adopt the decision period (i.e., the frequency at which the agent takes an action) since too high network update frequencies (greater than 10 Hz) did not allow the agent to learn to move, since too short a press on the accelerator pedal is not sufficient to allow movement. A good trade-off we found is to use a CARLA environment update rate of 20 Hz and a network update rate of 5 Hz (equivalent to a decision period of 4) but this required a lot of trial and error.

4.2.3 Conclusions

The proposed work presented a DRL agent for real-time trajectory planning and tracking in a CARLA-simulated environment where low-speed maneuvers are performed in a parking area with comb-shaped spaces. The agent is able to achieve a 97% success rate in reaching the target parking lot without collisions. The training process involved a CL strategy with three distinct phases that gradually increased the complexity of the environment by introducing NPVs and penalizing collisions. However, the use of CARLA presented challenges in integrating DRL into the environment. Issues related to the environment update frequency and decision period had to be carefully handled to ensure proper learning and observation by the agent but the overall success rate and performance of the trained agent confirm the capability of DRL approaches for the development of ADFs. Future work will focus on different sensor configurations and extend the range of action to more complex parking scenarios.

4.3 Extending the CARLA parking environment

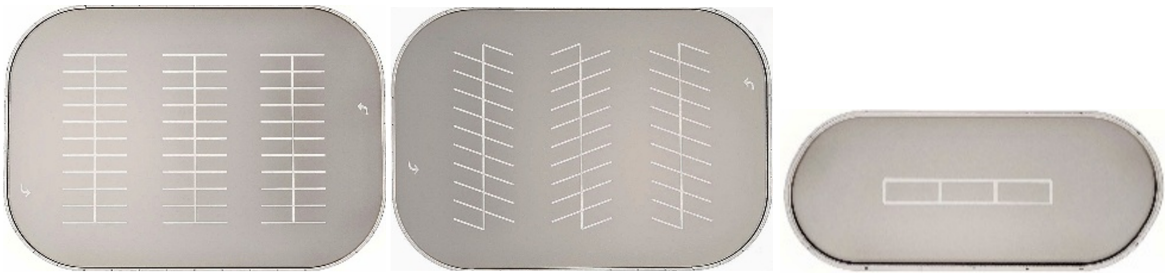
The CARLA experiment introduced in the previous section demonstrated that DRL agents can learn to perform AP reliably in comb-style slots. However, limiting evaluation to a single configuration risks overfitting policies to narrow conditions. A natural extension is therefore to broaden the operational design domain (ODD) by introducing additional parking layouts that reflect the diversity of real-world scenarios.

To this end, we developed a unified CARLA-based simulation framework supporting three distinct geometries: 90° perpendicular, 60° skewed, and parallel slots. Each environment was created from scratch to capture the spatial and maneuvering constraints specific to its layout. DRL agents were then trained and evaluated individually in each scenario, enabling analysis of how geometry influences learning and policy performance.

Although this study focuses on static conditions, the framework was designed with flexibility to support future extensions involving dynamic obstacles and interactive traffic participants. This establishes a platform for studying DRL-based AP under increasingly realistic and heterogeneous conditions.

4.3.1 Method

Building upon previous work, we extend the parking environment described in Chapter 4.2. We adopted CARLA as simulation framework to keep the training logic as intuitive as possible thanks to the CARLA Python API for client-server communication. Although the original work represented a first step towards more realistic simulation mainly in terms of physical similarity to real world scenarios, especially when compared to previous research on Unity 4.1 and highway-env 3, which were limited by the lack of variety in parking scenarios and simulation complexity. Thus, the main goal of this work concerns the modeling and integration of two additional parking lot types, enabling research on diverse scenarios. Fig. 4.15 depicts the three proposed parking environments, which, in contrast to previous work, were not derived from the CARLA built-in Town 5 map, but by creating a custom road layout in Unreal Engine from scratch, leveraging the open-source CARLA fork. All the proposed environments incorporate waypoints which enable dynamic traffic simulation, as well as spawn points for non-player vehicles (NPVs) in the parking lots. All the maps are surrounded by a 1.5 m tall guard rail. Fig. 4.15a shows the new version of the parking area originated from Chapter 4.2. Fig. 4.15b depicts the second setting consisting of 60 parking spaces, inclined by 60°, which is common in urban roads. The dimensions of the whole parking area and of the single lots is the same as in the perpendicular environment. The third proposed scenario concerns parallel parking (Fig. 4.15). Here, the slots are configured for



(a) (b) (c)

Fig. 4.15 Bird-eye view of the proposed simulation environments. (a) 90° perpendicular; (b) 60° skewed; (c) Parallel parking.

both-sided access, ensuring that parking may be performed from each direction by the training agent. This configuration is inherently more spatially constrained, measuring 38×16 m. All environments include all different pre-built CARLA weather conditions, and 41 vehicle blueprints are available for testing.

Experiments

To validate the proposed environments, we conducted three separate experiments involving DRL agents tasked to solve each parking task. To enable DRL in CARLA, as usual, we defined a custom Gymnasium [126] environment and employed the SB3 [105] Python library to quickly setup and train pre-defined models. All agents were trained through PPO [112], with a MLP network backbone composed of 2 hidden 512-units layers. The action and observation spaces for the agents remains unaltered from Table 4.12. The main structure of the RF and its sparse (awarded only when specific events occur) and dense (given at each simulation step) components were kept equal or similar, depending on the task, to the one proposed in Table 4.11. An episode ends when either the EV successfully reached the parking target or a collision was registered.

The first experiment served as a preliminary validation for the new perpendicular parking environment and as an evaluation of the generalization capabilities of the baseline AP agent from Chapter 4.2. We deployed the pre-trained agent in all proposed scenarios under the same original experimental conditions. We froze the weights of the model and, without further fine-tuning, the agent was evaluated. Thus, no modification of the RF was necessary.

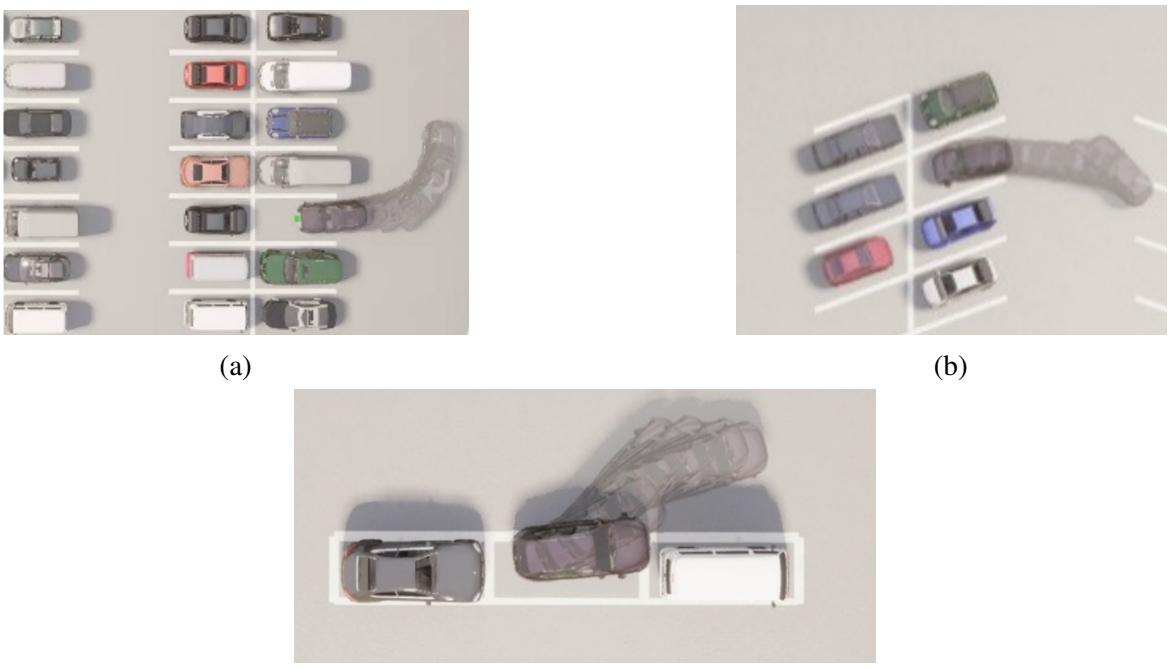
For the second and third experiments we adopted a CL strategy for training two additional agents, one for each new parking scene. In both cases we resumed the training procedure of the 90° expert agent to start the training phase from already established knowledge of the parking task and ensure quicker convergence times.

The second experiment consisted in training an agent with the goal of parking in the 60° skewed lots scenario, adapting its alignment to the modified parking angle. At the beginning of each episode, one of the 60 parking slots is randomly selected as the target, the EV is spawned on the nearest drivable area aligned with the target slot's longitudinal axis, and 0 to 10 randomly selected NPVs are placed in adjacent parking slots. The initial lateral position of EV is perturbed using uniform noise within ± 5 meters and its initial rotation was randomized between 0° and 360° to introduce environmental variability. To encourage correct alignment, the positive *Goal* contribution of the RF, awarded only once per episode if the EV reaches the target, was nullified if the angle difference between the forward vector of EV and the parking lot orientation was $\geq 10°$.

For the third experiment we trained an additional agent to solve the automated parallel parking task. Here, only 3 parking slots are present in the scene, with the central slot always designated as the target. At the start of each episode, 0 to 2 NPVs are placed in the adjacent slots. The initial position of the EV is randomized between the left and right side of the parking area, again with ± 5 meters of lateral shift and randomized initial rotation. As per the second experiment, the condition for EV's success was further refined by introducing a stricter alignment error threshold of 6° upon arrival, to avoid unrealistic parking angles in the parallel scenario. The original RF was also extended to include a negative sparse contribution attributed to the model when the EV moves to the opposite side of the parking area. This served as a new terminal condition for the current episode. It was designed to prevent the agent from learning to park from only one side of the parking space thereby to promote generalization. In addition, depending on the initial position of EV, a positive, dense contribution was given when the model moved the vehicle forwards past the target parking slot. This was developed to encourage the agent to perform reverse parking maneuvers.

4.3.2 Results

Table 4.13 shows the results obtained after evaluating the baseline and the fine-tuned agents for 1,000 episodes in the proposed environments in terms of success rate. The baseline AP model achieves 97% success rate in the redesigned 90° parking lot scenario, as in the environment it was originally trained on. Fig. 4.16 depicts a sample trajectory driven by the pre-trained baseline. This suggests that the environment is well aligned with the original one in terms of task-relevant features and dynamics, enabling near-perfect zero-shot adaptation and validating the fidelity of the new scenario. Because of this, no further fine-tuned was required for this environment. When deployed to the other proposed scenarios, however, the generalization capability of the baseline model begins to falter, as can be seen from the success rate, dropping to 63% for the 60° slots scenario and to 2% for parallel parking. This



(a) (b)

(c)

Fig. 4.16 Visualizations of the trained agents performing parking maneuvers in the three proposed environments.

Table 4.13 Success rate of the proposed models over 1,000 evaluation episodes.

Environment	Baseline model [4]	Fine-tuned models
90° Perpendicular Parking	**97%**	-
60° Skewed Parking	63%	**99%**
Parallel Parking	2%	**95%**

spurred further investigation aimed at establishing stronger baseline performance in the new scenarios.

The final agents for each new environment were obtained after an additional CL phase lasting approximately 10M training timesteps for the 60° skewed lots scenario and 40M for parallel parking. The evaluation results for the second and third experiment are summarized in Table 4.13, where the fine-tuned models achieve success rates of 99% and 95% in their corresponding training scenarios, demonstrating the clear difference of the settings (particularly for the parallel parking case, which required a 4x longer fine-tuning and achieving lower performance than the 60° case) and the need for fine tuning. Representative trajectory examples illustrating the behavior of the fine-tuned agents in each environment are depicted in Fig. 4.16a and Fig. 4.16b, respectively.

4.3.3 Conclusions

This work presented a unified CARLA parking framework that incorporates 90° perpendicular, 60° skewed and parallel slot geometries within a single, custom-built map and exposes them through a Gymnasium-compatible interface. A baseline agent trained exclusively on perpendicular parking retained a 97% success rate when evaluated in the redesigned perpendicular scene, confirming that the new scenario reproduces the task dynamics of the baseline environment and that the original DRL agent in Chapter 4.2 had correctly learned the main task-relevant patterns, without overfitting its training environment. When the same policy was deployed without 'as is' in the skewed and parallel layouts, success rates dropped to 63% and 2% respectively, demonstrating that the new geometries introduce realistic spatial constraints preventing zero-shot generalization. Continued training with a curriculum strategy lifted performance to 99% in the skewed lot scene and 95% in parallel parking, showing that the environment supports knowledge transfer. These findings validate the environment as a practical development environment for research on policy robustness and curriculum design in AP.

Future work will extend the static scenes with dynamic traffic participants and pedestrians to enable more complex and realistic parking scenario generation. A public release of the map and evaluation utilities is planned to allow for comparative studies in learning-based low-speed maneuvering.

Chapter 5

Adversarial training to enhance decision making robustness

The next part of the study shifts the focus from policy learning and curriculum-based generalization to policy and decision robustness as a primary target. Here, we adopt an adversarial view of decision making. DRL adversarial training involves inserting in the environment an adversary agent aimed at preventing the victim (i.e., the target model to be trained) from achieving its goals. The adversary is typically trained with a RF which is the negation of the victim's reward. However, this choice is too simplistic in AD scenarios (e.g., lane change in [20] or intersection traversal in [114]) as it leads to frequent collisions, preventing any meaningful learning on the victim's part. For this reason, the opponent, seeking to expose weaknesses in the victims' decision making process does so by creating difficult but plausible situations. The adversary is not allowed to break physics or the task rules. It must act within the environment dynamics, accept collision penalties, and respect scene semantics. The victim agent tries to complete the task with high safety and stability. Training alternates between exposing the agent to these challenging interactions and updating the agent to remove the newly revealed failure modes. In effect, the process builds a curriculum that is driven by what the agent cannot yet handle and evaluates progress with safety and stability metrics.

In this context, the first study uses the highway setting introduced in Chapter 3 to validate the approach, measure the gap between baseline performance without an adversary and performance under adversarial stress, and assess performance regained after targeted exposure. The second study extends the paradigm to the AP domain in CARLA, following the same staged increase in complexity presented in Chapter 4. Here, we use regularization that favors realistic adversarial trajectories and avoids loss of prior knowledge during fine-tuning.

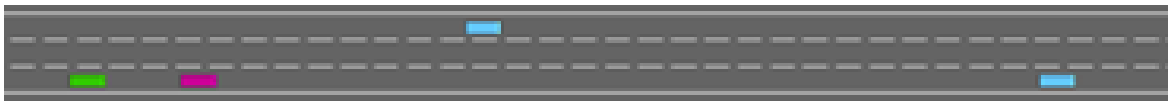


Fig. 5.1 Snapshot of the highway environment. The victim vehicle is colored in light-green, the adversary in purple, while the NPVs in light-blue.

As a result, these studies advance stable and robust DRL based decision making across more complex and diverse simulation settings by applying adversarial policies in the AD context. They reduce unsafe events, improve trajectory quality, and preserve baseline competence while widening the range of conditions in which agents operate reliably.

5.1 Adversarial policy learning in *highway-env*

The first study in this chapter applies adversarial training to the highway driving domain. Returning to this environment, already examined in Chapter 3 for DRL-based decision making, allows us to test whether adversarial interactions can systematically reveal weaknesses in a pre-trained policy and drive measurable improvements. Highway driving highlights the challenges of sequential decision making in AD, where agents operate over large state and action spaces and must ensure safety under uncertain conditions. Standard DRL policies, even when supported by CL or different training setups, remain prone to corner cases. To probe these vulnerabilities, an adversarial agent is introduced that generates difficult but plausible interactions within the simulator's physics and rules, forcing the victim agent into failure modes that standard training rarely uncovers.

The process alternates between training the adversary to disrupt the victim and fine-tuning the victim to overcome the exposed deficiencies. Through this loop, the complexity of the scenarios escalates naturally, yielding a victim policy that is systematically hardened against challenging conditions. Results indicate improved robustness, stability, and safety when compared with the baseline policy.

5.1.1 Method

We present a modified version of the highway-env [63] open-source simulator to develop a DRL training framework, given the simplicity and ease of customization which allows for quick prototyping and testing of different mechanics and details as well as a direct comparison to previous work [16].

Fig. 5.1 depicts the environment we used for developing our method. The scene contains a customizable number of heuristic Non-Player Vehicles (NPVs) navigating the scene alongside

two vehicles playing as the part of the victim (V) and the adversarial vehicles (A), that are guided by two different DRL policies. Two sets of actions are provided for the proposed experiment: the lower-level action space already present in the original framework (faster, slower, lane change right, lane change left) and a high-level action space (keep lane, go to right-most lane, overtake) illustrated in Chapter 3. The observations for both agents are named "kinematic", as per the original highway-env's definition and consist of presence (whether a vehicle is present or not in the scene), longitudinal and lateral position and velocity. These observations are taken for the ego vehicle and the $N = 7$ (for our experiments) nearest vehicles.

Training Cycle Definition

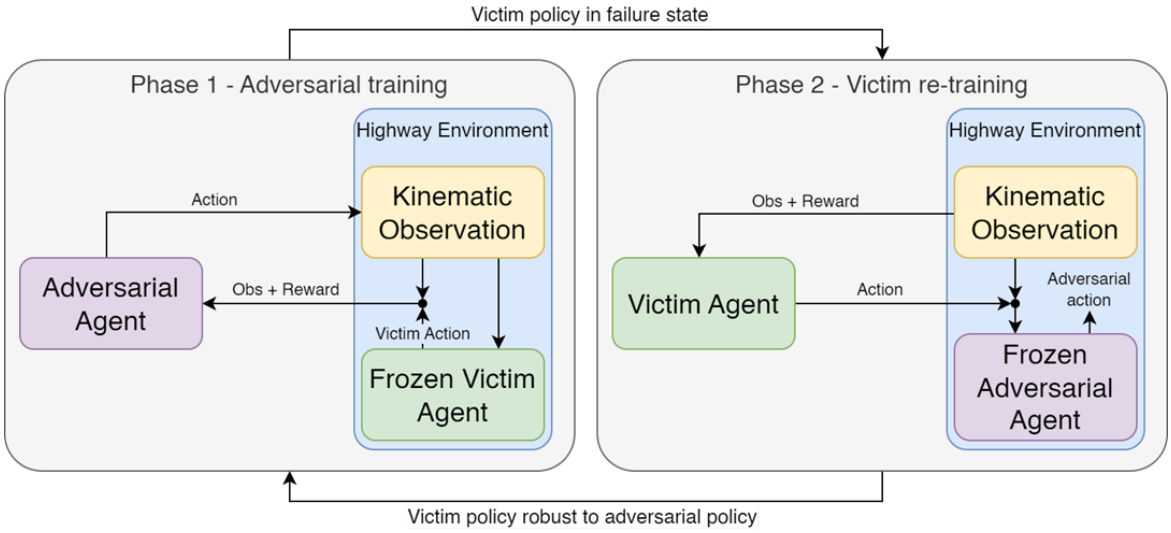


Fig. 5.2 Outline of the proposed two-phase adversarial training cycle.

For training the victim and adversarial agents, respectively driving V and A, we defined a two-phase cycle, which is depicted in Fig. 2. We begin the process by freezing a pre-trained victim model (particularly, the one illustrated in Chapter 3), and let it act in pure exploitation as a part of the highway environment. The goal of V is to drive safely (i.e., no collisions with other vehicles) for as many kilometers as possible. In this first phase, the learner is A acting upon the scene alongside with V and the NPVs. The goal of A is to hinder V. To this end, the A observation includes also the V last observation and action. Unlike V, which uses the aforementioned high-level action space designed to reach better "normal" driving performance, A uses the original lower-level action space, which allows a greater degree of freedom in driving, which is needed to perform abrupt maneuvers to challenge V. The idea

behind the first phase of training is to build a strong adversarial policy and put the pre-trained vehicle in a danger/failure state.

As the main effect of the first training phase, we expect V to collide significantly more frequently than in the normal situation (i.e., without trained A) and maintain a lower mean cruising speed. We conclude the first training phase when the metrics indicate a clear degradation of V performance and proceed to the second phase. Here, A is frozen and V becomes the subject of the training. Thus, we resume the victim's model training through a continued learning procedure without any changes to the original RF or state and fine-tune it in the adversarial scenario. The goal of this second phase is to make V policy robust to the newly introduced adversarial attack, thus enhancing V capability to deal with unexpected dangerous behaviors by other vehicles, that would previously induce a failure state of the agent. This two-phase process is then repeated a certain number of times, until the predefined stopping criteria is met (e.g., in terms of V policy robustness) or results suggest using different hyper-parameter values.

Experiment

The DRL model deployed for both victim and adversarial agents is the Proximal Policy Optimization (PPO) algorithm [112]. The neural network at the core of the algorithm is a 2-layer MLP, with 64 neurons per layer for both analyzed models. The environment policy frequency (i.e., decision rate) is set to 1 Hz., to give the agents sufficient time to observe the effect of their previous action. We set the maximum training episode duration to 60 seconds. In both training phases an episode is terminated whenever a collision occurs. Also, when training A, the episode is terminated if A is overtaken by V, since A did not succeed in making V fail. The number of NPVs is set to a randomly chosen value from 10 to 15 for each episode in both training phases.

The RF is the key to the success of any DRL agent training, since RF numerically shapes the model behavior. For the first training phase (in which the V policy is frozen), RF aims to reward the A's ability to make V fail. Thus, RF of A is simply the opposite of RF with which V was trained. That RF included rewards for high speed and right-most lane, and huge penalty for collision, as detailed in Eq. (3.4) in Chapter 3. It is important to stress that the considered quantities for the A training (i.e., speed, lane collision) are referred to the V. However, the RF of A additionally includes a penalty for its own collision with NPVs (a collision would prevent A from continuing its disturbance task), while a rear-front collision with V is not penalized (we do not reward it to prevent A from learning to really chase for V, which would be an unrealistic behavior).

Finally, if V manages to overtake A, the episode is terminated, with a training penalty. In the second phase, the A policy is frozen, while V is fine-tuned with its "normal" RF, which is described in Eq. (3.4) in Chapter 3. It is important to highlight that, in each phase, only one agent is trained, so to prevent non-stationary conditions that would hinder the training.

5.1.2 Results

We evaluate the victim's policy before and after adversarial re-training. Particularly, the increase in robustness of the V policy is evaluated by comparing the episode metrics reported in Table 5.1 averaging 1000 test episodes. Tests were conducted in the highway 3-lane environment in two conditions: with and without A in the scene, while NPVs are always present. In the first phase, the emergent behavior adopted by our best A model can be described as a continuous, yet not random, swerving procedure. As soon as A detects the presence of V behind, it visibly tries to move up or down the lanes to block V. If possible, A tends to steer at the very last possible moment to cut right in front of V and thus have a chance of causing a "good" collision (rear-front). As shown in Table 5.1, this adversarial behavior leads V to increase the collision rate by one order of magnitude (14.7% vs. 1.4% of the episodes) and lower its speed significantly when compared to the case without the A. The V model also shows a reduction in average deceleration and an increase in average acceleration. We argue that this is due to the fact that V drives more slowly, due to the behavior of A (less average deceleration), then it tries to abruptly accelerate to quickly perform the overtaking.

The second phase aims to improve the pre-trained policy of V, which is the actual goal of the overall process, through a continued learning procedure. V has now to learn to deal with A, which hinders the normal behavior of V through selective lane and speed changes. Comparing the third and first column of Table 5.1, we see that, the V policy at the end of this phase is still negatively influenced by the A presence, mainly in terms of collision number

Considered metrics	**Baseline**	**Baseline vs.** A	**Re-trained vs.** A	**Re-trained**
Mean episode duration (s)	59	53	57	**60**
Number of collisions	14	147	65	**2**
Average travelled kilometers	1.84	1.5	1.82	**1.95**
Average speed (km/h)	111	89	**118**	**118**
Average deceleration (m/s^2)	-5.38	-4.52	-5.47	**-5.57**
Average acceleration (m/s^2)	1.62	1.84	**1.28**	1.34
Average left lane changes	0.32	0.38	0.9	**0.96**
Average right lane changes	**1.33**	1	1.09	1.27

Table 5.1 Performance over 1000 episodes of the victim models

and average kilometers travelled, compared to the original V without A. The average left-lane change number is also in-creased, suggesting that the V has learned to overtake vehicles more often in the adversarial scenario. On the other hand, comparing the second and third column, we see with no surprise that, in the adversarial scenario, the re-trained agent outperforms the original one in all the metrics.

Finally, testing the agent in a normal, non-adversarial scenario (fourth and first column), we see that the re-trained model collides less frequently, while traveling more kilometers at a slightly higher speed and accelerates less abruptly. This indicates a significant agent improvement in terms of safety and overall robustness.

5.1.3 Conclusions

We explored the implementation of an adversarial learning procedure to increase robustness of a DRL agent for AD in a 3-lane highway-env simulated environment. Experimental results show that the re-trained model's policy has proven to be more robust and safer, outperforming the original one in all the metrics.

These initial results suggest that research should be conducted on the RF, improving the realism of the A disturbances, and loss function customization for the employed learning algorithm to stabilize the training phase. Moreover, the method may be transferred to more complex driving scenarios and simulators.

5.2 RAFT (Regularized Adversarial Fine-Tuning) for self-parking

Having validated adversarial training in the highway domain, the study now extends the approach to the more complex CARLA simulation environment as we did in Chapter 4, here again in the context of AP. To this end, we introduce the Regularized Adversarial Fine-Tuning (RAFT) framework, designed to improve a pre-trained agent while maintaining a balance between the adversary's challenge and the victim's capacity to learn (Fig. 5.3).

As a test case, the self-parking agent developed in Section 4.1 and ported into the CARLA environment in Section 4.2. The evaluation investigates the effectiveness of RAFT by analyzing performance improvements, conducting an ablation study on the regularization hyper-parameter, and assessing multiple task-relevant metrics. Together, these experiments demonstrate the benefits of adversarial fine-tuning when applied to realistic AP scenarios.

5.2.1 Method

The idea behind RAFT is to optimize an adversarial policy to prevent an overly aggressive behavior by using a general regularization term in the RF with the goal to achieve a balanced trade-off between environmental complexity and agent learning efficiency. Fig. 5.4 depicts the proposed framework for the adversarial training of DRL agents. The adversary A is an agent driving a vehicle aimed at obstructing the parking of the victim V (i.e., our target parking agent). The environment consists of 90° parking slots, static parked vehicles around the target slot, V, and the adversary A. At the beginning of the training, V is the pre-trained parking agent, while A is initialized randomly. Then, the adversarial training starts, which involves a cycle two phases per iteration. In the first phase, the adversary A is trained, with its

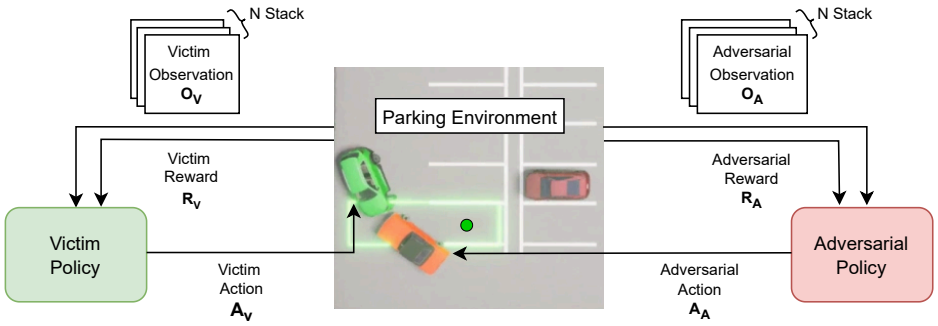


Fig. 5.3 RAFT overview. The scene includes the victim agent (green), adversarial agent (orange), and a static environment vehicle (red). The green dot is the target parking slot, the green rectangle is Ar_{tgt} in Eq. 5.6.

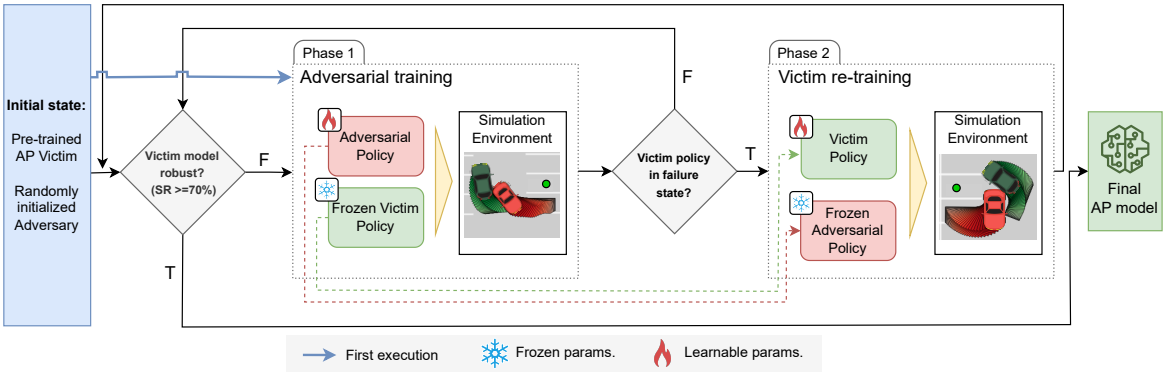


Fig. 5.4 Proposed training pipeline, where adversarial agent A learns to exploit victim agent V in Phase 1, followed by V's re-training against A in Phase 2, iteratively refining both policies until V achieves robustness. The blue arrow represents the first execution of loop, skipping V's robustness check.

own RF, while the victim V (i.e., the target agent) acts in the environment without learning. In the second one, the victim is trained while the adversary policy is kept frozen (i.e., the opponent is part of the environment, without learning). The initial condition is the pre-trained agent and a random initialized adversary. We define the complete state of the environment as $S = \{D, T\}$, where D refers to the position, longitudinal and lateral acceleration, speed and angular velocity of all vehicles in the scene and T to the position and alignment vector of the current parking target.

Adversarial policy learning

The observable state space of the adversarial agent is the same as for the victim, and is defined by the tuple:

$$O_A = \{\Delta(V,T), \Delta(V,A), D_A\}, \tag{5.1}$$

where $\Delta(V,T)$ and $\Delta(V,A)$ represent the distance between the victim agent and the target's position in Cartesian coordinates (x,y) and between the victim agent and the adversary's position, respectively. D_A refers to the subset of D, only containing the features describing the adversarial vehicle. For both the agents, the input stacks past frames to enhance situational awareness - enabling the agent to track its own movement and detect environmental changes. The stack size was empirically set to five frames, aligning with the DRL agent used for comparison from Chapter 4.1. This selection balances the advantages of historical context against the computational overhead introduced by the larger input dimensionality. The adversarial action space is $\mathscr{A}_A = \{throttle, brake, steering, reverse\}$, as that of V.

As DRL training algorithm, we employ PPO. By iteratively tuning the policy parameters based on a clipped surrogate objective, it strikes a balance between exploration and exploitation while keeping stability and reliable performance.

Adversarial reward

To obtain a successful adversarial policy, the reward signal R_A must be designed to disrupt the victim policy, whose objective is defined by R_V. The typical approach (e.g., [36]) formulates the problem as a zero-sum game, which can be modeled as:

$$R_A = -R_V. \tag{5.2}$$

As collisions are heavily penalized by the victim's RF, they are highly rewarded for the adversary. Although this approach (we call it adversarial baseline) can produce highly disruptive adversarial policies, the resulting behavior risks to be excessively aggressive and may not be able to help the victim's to improve its generalization capability. Inspired by [20], we relax the zero-sum assumption and add a regularization reward R_{rs}, expanding Eq. 5.2 as:

$$R_A = -R_V + \omega R_{rs}, \tag{5.3}$$

where R_{rs} encourages the adversarial policy to generate more realistic trajectories by penalizing the collision (R_r) and the full blocking of the parking area (R_s) with a weight (ω) that controls the influence of the regularization term:

$$R_{rs} = R_r + R_s. \tag{5.4}$$

To limit the disruptive adversarial collision strategy, the learning agent is penalized through the following sparse contribution:

$$R_r = \begin{cases} -1, & \text{if } A \text{ collides,} \\ 0, & \text{otherwise.} \end{cases} \tag{5.5}$$

Because of the large time horizon of a training episode, we add the dense contribution R_s which is given at every simulation step:

$$R_s = \begin{cases} -0.05, & \text{if } A \text{ is in } Ar_T, \\ 0, & \text{otherwise,} \end{cases} \tag{5.6}$$

where Ar_T refers to a predefined area that dynamically moves, based on the location of the current target parking slot (Fig. 5.3). In the AP context, R_s is introduced to prevent A from physically obstructing the victim's target slot. More generally, it can be adopted to limit the influence of A outside a user-defined Ar_T to achieve more realistic adversarial trajectories, which may be useful in various AD contexts (e.g., in lane change, to prevent A from learning to drive off-road or in the opposite lane).

Regularized adversarial re-training

The regularized reward is then used in the adversarial training cycle (Phase 1, in Fig. 5.4). The goal of such CL cycles is to make V robust to A, enhancing its capability to better deal with obstacles and improving its trajectory stability. We conclude V's re-training session (Phase 2 in Fig. 5.4) when its policy reaches at least a 70% success rate in the adversarial environment. The threshold was set empirically to optimize the adversarial training loop's efficiency, based on our observation that the victim model typically converges at approximately a 70% success rate.

Notably, while [20] set up an adversarial framework to spot weaknesses of the target agent in the adversarial environment, our aim is to improve the agent through its experience in the adversarial environment. The main originality of our work consists in designing a general regularization technique (Eq. 5.3) and verifying that it enhances a state-of-the-art agent, while [114] proposes a custom adversarial RF (for intersection traversal) and does not apply it to a state-of-the-art model.

5.2.2 Results

Simulation setup

We evaluated the proposed method in the CARLA driving simulator. Fig. 5.3 depicts the scene, comprising the victim V (in green), the adversary A (in orange) and a parked vehicle E, close to the target. Every episode, there are up to two nearby parked vehicles. The green dot marks the target parking slot, and the green rectangle refers to Ar_T, in Eq. 5.6. The parking slot structure was derived by "Town 5", a pre-constructed CARLA map, adapted to the AP problem and optimized to enable the learning of one agent policy at a time for DRL-based adversarial and victim retraining.

We deploy as victim a state-of-the-art DRL-based AP agent that we ported from Unity to CARLA and trained and tested according to the procedure reported in Chapter 4.1. Our goal is to check whether adversarial training allows improving the model's local motion planning performance in its original ODD (i.e., static parking, with 0-2 stationary vehicles parked in

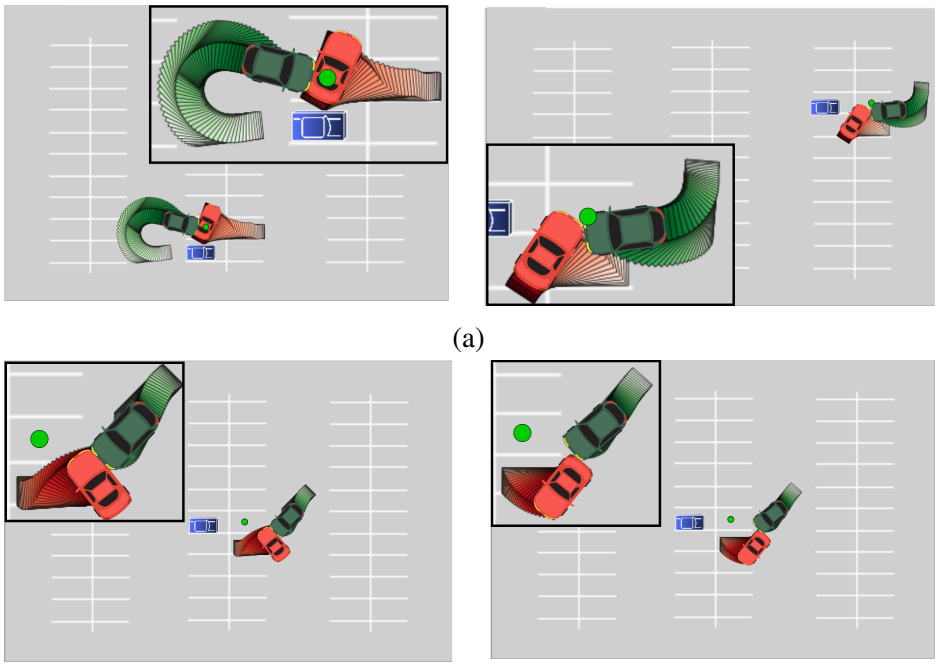
the neighborhood and no adversary). The performance obtained by this original model is reported in Table 5.2 in terms of success rate, alignment deviation A_{err}, number of reverse actions N_{rev}. This condition is challenging, since the main indicator (success rate) is already quite high. Improving the other two (e.g., with another curriculum step, tuning the RF) may be unsuccessful (e.g., for a too low learning rate) or at risk of overall catastrophic forgetting.

A parking attempt is considered successful if V reaches the target T. Alternatively, the episode terminates if V collides with any object or at a 120 second timeout, which we empirically set to balance the competing objectives of allowing sufficient exploration time for the agent while preventing unnecessary time expenditure in deadlock scenarios. At each episode, the alignment of the target is randomized as well, between front and rear parking. The deviation concerns the discrepancy with the vertical alignment to the parking slot. The number of reverse actions is to be minimized, to have a better experience and reduce fuel consumption. At each episode, a target slot is randomly chosen. V is generated outside the slot, within a 10 m distance and with a random orientation. The adversarial agent is randomly spawn as well, in the proximity of the victim. The policy guiding the victim agent perceives its surroundings through a short-distance (i.e. 5-meters) 360° LiDAR sensor (Chapter 4.2).

We perform two experiments to verify the two types of adversarial rewards (baseline adversary, Eq. 5.2; and regularized adversary, Eq. 5.3). Each experiment executes an adversarial training cycle and assesses the achieved performance over 10K simulation episodes.

Baseline adversary

Results for the baseline adversary are reported in the second row of Table 5.2. We do not explicitly report the collision rate of V, as it is simply the complement of the success rate, as no episode timeout was observed in the test phase. After re-training with the baseline adversary, V's success rate drops significantly (-3.62%) and its alignment error to the parking target increases. This result already emphasizes the inadequacies of such adversarial strategy, which leads to catastrophic forgetting in the policy of V. Table 5.3 shows how the adversary is able to disrupt the normal victim capabilities when compared to the non-adversarial environment. However, Fig. 5.5a shows that the sample adversarial trajectory generated by this preliminary phase is not indicative of a true failure of the V policy, since the adversary's behavior completely neglects safety. The goal of A is solely achieved through physical target blocking and by purposedly seeking a direct collision with V, which is reflected by the comparison between adversary models presented in Table 5.4. Thus, we argue that the adversary RF should be tuned to allow our target model to learn also from the interaction with the adversary.



(a)

(b)

Fig. 5.5 Trajectories of V (green) and A (red) using the baseline (a) and the proposed (b) adversarial policy. The green dot is the parking target. (a) shows A simply *blocking the parking slot*, making V's parking impossible, while (b) shows A not intentionally colliding with V but exiting the parking slot.

Regularized adversary

The second experiment concerned the regularized reward (Eq. 5.3). Results, reported in the third row of Table 5.2, show the influence of the regularized adversarial re-training session on V in the targeted static environment. While the success rate increases only slightly, the alignment error and the mean number of reverse gear changes significantly decrease, hinting at a more stable and targeted trajectory.

Table 5.3 shows that the regularized adversarial policy leads V to a higher success rate and lower collision rate than the baseline. This is negative, from the adversary-only point of view, but the generated adversarial behavior and trajectories are more realistic, as it appears from the metrics in Table 5.4, fourth row. Studying the influence of each component of R_{rs}, the second and third row of Table 5.4, show that the adversarial agent becomes either too aggressive or conservative when only one of the two regularization contributions is activated (R_r or R_s respectively). This demonstrates that both components are necessary to attain stable and realistic behavioral outcomes.

Table 5.2 Evaluation of the AP agent in the ODD environment, pre/post adversarial re-training, per episode. A_{err} is the alignment deviation and N_{rev} the number of reverse changes.

Agent	**Success rate**	A_{err}	N_{rev}
Unity Baseline Ch. 4.1	96.98%	4.12°	3.21
Baseline	93.36%	5.62°	3.19
Regularized	**97.15%**	**3.24°**	**1.83**

Table 5.3 Evaluation of the victim (V) in the adversarial environment.

Environment	**Success rate**	A_{err}	N_{rev}
Base ODD	96.98%	4.12°	3.21
Baseline Adversarial	**9.21%**	41.21°	6.34
Proposed Adversarial	**31.43%**	39.67°	4.47

Additionally, we conducted a sensitivity analysis on the ω weight (Eq. 5.3), with values in the 0.5-3 range (Table 5.5). A remains stable in the 0.5-1.5 interval, while becomes passive with $\omega \geq 2$. We attribute this to the collision penalty (R_r) becoming dominant in the reward signal.Thus, we set $\omega = 1$ in all the experiments, as it offers the best trade-off between policy stability and effectiveness.

When compared to the baseline, A exhibits a drastic reduction in the mean number of collisions per episode, which is indicative of a more stable driving behavior. Moreover, the new adversarial policy guided A outside of Ar_T and learned not to completely block the target. The different, more useful, adversarial behavior also appears from the generated trajectories. For instance, Fig. 5.5b shows that A limits its disruptive behavior by exiting the parking slot at a certain angle, depending on the initial simulation scene, and then braking. This behavior allows to assess the response of V which (before the fine-tuning cycle) often led to a policy failure and not seldom to a collision (now caused by itself rather than purposedly by A), highlighting gaps on V generalization capability in such challenging scenarios. This is also confirmed by the collision rate of V, which is significantly higher than in the non-adversarial testing environment.

5.2.3 Limitations

The experiments showed the effectiveness of RAFT in enhancing the target agent's robustness in the targeted static parking scenario.

However, the proposed adversarial training introduces the need for the target agent to learn also dynamic obstacle avoidance, which would represent an ODD extension. We verified this

Table 5.4 Evaluation of the adversary models against V pre/post regularization, per episode. C_{all} is total collisions, C_E collisions with E or walls, CR_V collisions with V, and T_{block} time steps of A in Ar_T.

Adversarial Model	C_{all}	C_E	CR_V	T_{block}
Baseline ($-R_V$)	2.44	1.82	**82.23%**	11.87
$-R_V + R_s$	4.52	4.34	44.02%	1.05
$-R_V + R_r$	0.02	0.02	1.76%	8.08
Proposed ($-R_V + R_{rs}$)	**0.98**	**0.34**	59.44%	**1.43**

Table 5.5 Sensitivity analysis on weight ω (Eq. 5.3).

ω	C_{all}	C_E	CR_V	T_{block}
0.5	1.13	1.02	61.21%	2.13
1	**0.98**	**0.34**	**59.44%**	**1.43**
1.5	0.99	0.35	59.64%	1.52
2	0.08	0.01	1.03%	0.12
3	0.02	0.01	0.52%	0.01

potential in an experiment comparing V's pre- and post- adversarial training behavior in a new test environment with a heuristic vehicle leaving a nearby parking slot. The achieved 93% success rate overcomes the score of the original agent (89%). However, extending the ODD to dynamic environments involves moving beyond one-to-one interactions to address the complexities of multi-agent systems (e.g. multiple vehicles and pedestrians), which is a challenge for future work.

5.2.4 Conclusions

We have proposed an adversarial DRL training framework (RAFT) and tested it in a CARLA self-parking application. Results show that RAFT is able to enhance a state of the art agent in its original static parking ODD, by increasing its robustness, with improvements in all task-relevant metrics. Notably, we achieved this not through an ad-hoc RF, but simply by adding a general regularization term to the baseline adversary reward.

We focused on in-ODD improvement as a fundamental step to verify the validity of the approach, also from a CL perspective. The obtained results open significant research perspectives, for instance exploring multi-agent adversarial settings with more complex and realistic victim-adversary interactions. Modeling adversary behavior using real-world data is

another possible research direction. Finally, the framework is general and could be tested in other use cases as well, such as urban or high speed driving scenarios.

Chapter 6

Conclusions

This thesis advances the state of the art in DRL-based decision making for automated driving by addressing four practical gaps observed in prior work: (i) how the decision problem is posed to the agent, (ii) how learning is stabilized in low-speed, contact-prone tasks, (iii) how operational design domains (ODDs) are expanded beyond a single scenario without retraining from scratch, and (iv) how adversarial robustness is improved without inducing unrealistic behaviors or catastrophic forgetting. These gaps are addressed through maneuver-level action abstraction, curriculum learning, ODD benchmarking and targeted fine-tuning, and Regularized Adversarial Fine-Tuning (RAFT). The resulting contribution is a unified training stack that treats stability and safety under perturbations as design objectives.

At the tactical layer in highway driving, the thesis shows that robustness is not only a matter of bigger networks, but also of better problem structure. By replacing flat, low-level control with maneuver-level action abstraction and delegating execution to a dedicated module, the agent explores more safely, learns faster, and exhibits more stable behavior in dense interactions. This choice also increases interpretability. Each action maps to an operationally meaningful maneuver, making failure analysis and reward design more transparent. In this setting, the study formalizes reward shaping as an explicit driving contract and demonstrates that distinct driving styles can be obtained by tuning few terms while maintaining feasibility and safety constraints.

For low-speed AP, where sparse rewards and credit assignment are exacerbated by frequent contacts and reversals, the thesis establishes CL as an effective mechanism to avoid common local minima. Across Unity-based studies, staged curricula that initially relax terminal conditions and progressively increase difficulty yield stable convergence and policies that match classical planning baselines in success rate while improving practically relevant behaviors (e.g., fewer gear inversions and reduced latency). The analysis also highlights

implementation factors that are often underreported in DRL AP literature but materially affect learnability, including simulation tick rate, decision period, and sensor modality.

To move beyond single-scenario validation, the thesis transfers the methodology to higher-fidelity simulation and makes ODD extension an explicit object of study. In CARLA, a LiDAR-based PPO agent trained with a staged curriculum achieves high success in comb-style parking, confirming that the training logic generalizes when timing and perception are matched to the simulator. More importantly, a unified CARLA parking benchmark covering perpendicular, skewed, and parallel layouts is introduced through a Gymnasium-compatible interface, enabling systematic evaluation and comparison. The measured performance drop under geometry shift, and its recovery via targeted CL-based fine-tuning, provides a concrete recipe for expanding ODDs while reusing existing policies and preserving prior competence.

Finally, the thesis targets robustness under adversarial interactions, where adversarial agents that succeed through unrealistic collisions or trivial blocking can destabilize the victim via catastrophic forgetting. In highway-env, alternating training between attacker and victim exposes failure modes rarely observed in standard training, and subsequent victim fine-tuning improves resilience under attack while also improving safety in non-adversarial evaluation. In CARLA, the proposed RAFT framework introduces a regularized adversarial reward that discourages unrealistic behaviors and region blocking, resulting in adversarial trajectories that remain challenging yet feasible. After RAFT, the AP policy maintains its baseline success in the original static ODD and improves trajectory-quality indicators such as alignment error and reverse actions, without catastrophic forgetting.

Limitations remain. Results are obtained primarily in simulation. Despite CARLA's increased fidelity, sim-to-real transfer will require systematic domain randomization, sensor calibration, and validation on real vehicles or high-quality logged datasets. Generalization is still bounded by the diversity of the training distribution. Changes in geometry or interaction patterns can degrade performance unless variations are explicitly covered during training. Moreover, adversarial training introduces design degrees of freedom (e.g., realism constraints and definitions of feasible safety-critical interactions) that influence which failure modes are discovered and how broadly robustness improvements transfer.

The contributions of this thesis enable several research directions. Automated curriculum design can replace manually defined stages with teacher algorithms that adapt difficulty based on competence and explicitly optimize safety-related metrics. Robustness can be extended from one-on-one adversarial interactions to multi-agent settings involving multiple vehicles and pedestrians, with adversaries constrained by behavioral priors learned from real data. Stronger deployment guarantees can be pursued by integrating uncertainty estimation, safety filters, and post-training verification, with the goal of moving from empirical robustness to

certifiable robustness. Finally, the released environments, interfaces, and benchmarks support reproducible comparison and can be used by other researchers to test new training algorithms, robustness objectives, and ODD-expansion strategies across both tactical highway driving and low-speed maneuvering.

References

[1] (2010). Pelops white paper.

[2] (2023). Unity. Website.

[3] (2026). A deep reinforcement learning approach integrating lstm-gat spatiotemporal fusion for autonomous driving decision-making. *Physica A: Statistical Mechanics and its Applications*, 681:131121.

[4] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., et al. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*.

[5] Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., and Mané, D. (2016). Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*.

[6] Anzalone, L., Barra, P., Barra, S., Castiglione, A., and Nappi, M. (2022). An end-to-end curriculum learning approach for autonomous driving scenarios. *IEEE Transactions on Intelligent Transportation Systems*, 23(10):19817–19826.

[7] Ashraf, N. M., Mostafa, R. R., Sakr, R. H., and Rashad, M. (2021). Optimizing hyperparameters of deep reinforcement learning for autonomous driving based on whale optimization algorithm. *PLoS ONE*, 16.

[8] Badue, C., Guidolini, R., Carneiro, R. V., Azevedo, P., Cardoso, V. B., Forechi, A., Jesus, L., Berriel, R., ao, T. P., Mutz, F., Veronese, L., Oliveira-Santos, T., and Souza, A. F. D. (2021). Self-driving cars: A survey. *Expert Systems with Applications*, 165:113816.

[9] Bansal, T., Pachocki, J., Sidor, S., Sutskever, I., and Mordatch, I. (2017). Emergent complexity via multi-agent competition. *arXiv preprint arXiv:1710.03748*.

[10] Bellemare, M. G., Dabney, W., and Munos, R. (2017). A distributional perspective on reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 449–458.

[11] Bellotti, F., Lazzaroni, L., Capello, A., Cossu, M., De Gloria, A., and Berta, R. (2023). Explaining a deep reinforcement learning (drl)-based automated driving agent in highway simulations. *IEEE Access*, 11:28522–28550.

[12] Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *Proc. ICML*.

[13] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym.

[14] Burda, Y., Edwards, H., Storkey, A., and Klimov, O. (2018). Exploration by random network distillation.

[15] Caesar, H., Bankiti, V., Lang, A. H., Vora, S., Liong, V. E., Xu, Q., Krishnan, A., Pan, Y., Baldan, G., and Beijbom, O. (2020). nuscenes: A multimodal dataset for autonomous driving. *arXiv preprint arXiv:1903.11027*. CVPR 2020 camera ready.

[16] Campodonico, G., Bellotti, F., Berta, R., Capello, A., Cossu, M., Gloria, A., Lazzaroni, L., Taccioli, T., and Davio, F. (2021). *Adapting Autonomous Agents for Automotive Driving Games*, pages 101–110.

[17] CARLA Simulator (2023a). Carla map editor. https://github.com/carla-simulator/carla-mapeditor. GitHub repository.

[18] CARLA Simulator (2023b). Carla scenario runner. https://github.com/carla-simulator/scenario_runner. GitHub repository.

[19] Chang, W.-J., Pittaluga, F., Tomizuka, M., Zhan, W., and Chandraker, M. (2024). Safe-sim: Safety-critical closed-loop traffic simulation with diffusion-controllable adversaries. In *European Conference on Computer Vision*, pages 242–258. Springer.

[20] Chen, B., Chen, X., Wu, Q., and Li, L. (2022). Adversarial evaluation of autonomous vehicles in lane-change scenarios. *IEEE Transactions on Intelligent Transportation Systems*, 23(8):10333–10342.

[21] Chen, K., Lei, Y., Cheng, H., Wu, H., Sun, W., and Zheng, S. (2024). FREA: Feasibility-guided generation of safety-critical scenarios with reasonable adversariality. In *8th Annual Conference on Robot Learning*.

[22] Chow, Y., Tamar, A., Mannor, S., and Pavone, M. (2015). Risk-sensitive and robust decision-making: a cvar optimization approach. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[23] Cobbe, K., Klimov, O., Hesse, C., Kim, T., and Schulman, J. (2019). Quantifying generalization in reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, volume 97 of *Proceedings of Machine Learning Research*, pages 1282–1289.

[24] Dalmia, P. (2022). priyamdalmia/predator-prey: Predprey variations.

[25] Dev, Y. (2023). Chevrolet corvette 1980 different colours. https://sketchfab.com/3d-models/chevrolet-corvette-1980-different-colours7e428bdb3ab54b4e9ac610e545fd9d03. Sketchfab model.

[26] Dolgov, D., Thrun, S., Montemerlo, M., and Diebel, J. (2008). Practical search techniques in path planning for autonomous driving. *ann arbor*, 1001(48105):18–80.

[27] Dong, C., Wang, H., Ni, D., Liu, Y., and Chen, Q. (2020). Impact evaluation of cyber-attacks on traffic flow of connected and automated vehicles. *IEEE Access*, 8:86824–86835.

[28] Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., and Koltun, V. (2017). CARLA: An open urban driving simulator. In *Proc. CoRL*, pages 1–16.

[29] Elios Lab (2023). Pathfollowing. https://github.com/Elios-Lab/pathfollowing. GitHub repository.

[30] Engstrom, L., Ilyas, A., Santurkar, S., Tsipras, D., Janoos, F., Rudolph, L., and Madry, A. (2020). Implementation matters in deep policy gradients: A case study on ppo and trpo. In *International Conference on Learning Representations*.

[31] Erickson, Z., Gangaram, V., Kapusta, A., Liu, C. K., and Kemp, C. C. (2020). Assistive gym: A physics simulation framework for assistive robotics. In *Proc. ICRA*, pages 10169–10176. IEEE.

[32] Fan, J., Lei, X., Chang, X., Miši, J., Miši, V. B., and Yao, Y. (2025). Less is more: A stealthy and efficient adversarial attack method for drl-based autonomous driving policies. *IEEE Internet of Things Journal*.

[33] Fei, Y., Shi, P., Li, Y., Liu, Y., and Qu, X. (2024). Formation control of multi-agent systems with actuator saturation via neural-based sliding mode estimators. *Knowledge-Based Systems*, 284:111292.

[34] Fliess, M. and Join, C. (2021). An alternative to proportional-integral and proportional-integral-derivative regulators: Intelligent proportional-derivative regulators. *International Journal of Robust and Nonlinear Control*, 32:9512 – 9524.

[35] Florensa, C., Held, D., Wulfmeier, M., Zhang, M., and Abbeel, P. (2017). Reverse curriculum generation for reinforcement learning. In *Conference on Robot Learning (CoRL)*, volume 78 of *Proceedings of Machine Learning Research*, pages 482–495.

[36] Gleave, A., Dennis, M., Wild, C., Kant, N., Levine, S., and Russell, S. (2019). Adversarial policies: Attacking deep reinforcement learning. In *Proc. ICLR*.

[37] González, D., Pérez, J., Milanés, V., and Nashashibi, F. (2015). A review of motion planning techniques for automated vehicles. *IEEE Transactions on intelligent transportation systems*, 17(4):1135–1145.

[38] González, D., Pérez, J., Milanes, V., and Nashashibi, F. (2016). A review of motion planning techniques for automated vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 17(4):1135–1145.

[39] Grigorescu, S., Trasnea, B., Cocias, T., and Macesanu, G. (2020). A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, 37(3):362–386.

[40] Grigorescu, S., Trasnea, B., Cocias, T. T., and Macesanu, G. (2019). A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, 37:362 – 386.

[41] Guo, W., Wu, X., Huang, S., and Xing, X. (2021). Adversarial policy learning in two-player competitive games. In *Proc. ICML*, pages 3910–3919. PMLR.

[42] Gutiérrez-Moreno, R., Barea, R., López-Guillén, E., Araluce, J., and Bergasa, L. M. (2022). Reinforcement learning-based autonomous driving at intersections in carla simulator. *Sensors*, 22(21):8373.

[43] Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor.

[44] He, J. Z.-Y., Erickson, Z., Brown, D. S., and Dragan, A. D. (2023a). Quantifying assistive robustness via the natural-adversarial frontier.

[45] He, X. and Lv, C. (2023). Towards safe autonomous driving: Decision making with observation-robust reinforcement learning. *Automotive Innovation*, 6(4):509–520.

[46] He, Z., Zhang, J., Yao, D., Zhang, Y., and Pei, H. (2023b). Adversarial generation of safety-critical lane-change scenarios for autonomous vehicles. In *Proc. IEEE ITSC*, pages 6096–6101.

[47] Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. (2018). Deep reinforcement learning that matters. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 3207–3214.

[48] Hoel, C., Driggs-Campbell, K., Wolff, K., Laine, L., and Kochenderfer, M. J. (2019). Combining planning and deep reinforcement learning in tactical decision making for autonomous driving. *IEEE Transactions on Intelligent Vehicles*, 5:294–305.

[49] Hoel, C.-J., Wolff, K., and Laine, L. (2020). Tactical decision-making in autonomous driving by reinforcement learning with uncertainty estimation. In *2020 IEEE intelligent vehicles symposium (IV)*, pages 1563–1569. IEEE.

[50] Hubmann, C., Schulz, J., Becker, M., Althoff, D., and Stiller, C. (2018). Automated driving in uncertain environments: Planning with interaction and uncertain maneuver prediction. *IEEE Transactions on Intelligent Vehicles*, 3:5–17.

[51] Iyengar, G. (2005). Robust dynamic programming. *Mathematics of Operations Research*, 30(2):257–280.

[52] Juliani, A., Berges, V.-P., Teng, E., Cohen, A., Harper, J., Elion, C., Goy, C., Gao, Y., Henry, H., Mattar, M., et al. (2018). Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*.

[53] Kang, Z., Liu, Q., and Jian, S. (2020). Modeling and simulation of merging behavior at urban expressway on-ramp. *Journal of System Simulation*, 29(9):1895–1906.

[54] Karbasi, A. H., Zheng, N., and O'Hern, S. (2024). Investigating the impact of automated vehicles on the safety and operation of low-speed urban networks. *Journal of Advanced Transportation*.

[55] Kesting, A., Treiber, M., and Helbing, D. (2007). General lane-changing model mobil for car-following models. *Transportation Research Record*, 1999(1):86–94.

[56] Kiran, B. R., Sobh, I., Talpaert, V., Mannion, P., Sallab, A. A. A., Yogamani, S., and Pérez, P. (2021). Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 23(6):4909–4926.

[57] Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526.

[58] Korkmaz, E. (2023). Adversarial robust deep reinforcement learning requires redefining robustness. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(7):8757–8765.

[59] Krajzewicz, D., Hertkorn, G., Rössel, C., and Wagner, P. (2002). Sumo (simulation of urban mobility)-an open-source traffic simulation. In *Proceedings of the 4th middle East Symposium on Simulation and Modelling (MESM20002)*, pages 183–187.

[60] Kwon, T., Di Palo, N., and Johns, E. (2024). Language models as zero-shot trajectory generators. *IEEE Robotics and Automation Letters*, 9(7):6728–6735.

[61] Lazzaroni, L., Bellotti, F., Capello, A., Cossu, M., De Gloria, A., and Berta, R. (2022). Deep reinforcement learning for automated car parking. In *International Conference on Applications in Electronics Pervading Industry, Environment and Society*, pages 125–130. Springer.

[62] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

[63] Leurent, E. (2018). An environment for autonomous driving decision-making. https://github.com/eleurent/highway-env.

[64] Leurent, E. and Mercat, J. (2019). Social attention for autonomous decision-making in dense traffic. *arXiv preprint arXiv:1911.12250*.

[65] Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2016). End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40.

[66] Li, F.-J., Zhang, C.-Y., and Chen, C. P. (2025). Robust decision-making for autonomous vehicles via deep reinforcement learning and expert guidance. *Applied Intelligence*, 55(6):412.

[67] Li, Q., Jia, X., Wang, S., and Yan, J. (2024). Think2drive: Efficient reinforcement learning by thinking with latent world model for autonomous driving (in carla-v2). In *European Conference on Computer Vision*, pages 142–158. Springer.

[68] Li, Q., Peng, Z., Feng, L., Zhang, Q., Xue, Z., and Zhou, B. (2022). Metadrive: Composing diverse driving scenarios for generalizable reinforcement learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

[69] Liang, Y., Sun, Y., Zheng, R., and Huang, F. (2022). Efficient adversarial training without attacking: Worst-case-aware robust reinforcement learning. *Advances in Neural Information Processing Systems*, 35:22547–22561.

[70] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

[71] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. In *International Conference on Learning Representations*.

[72] Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994*, pages 157–163. Elsevier.

[73] Liu, H., Huang, Z., Mo, X., and Lv, C. (2024a). Augmenting reinforcement learning with transformer-based scene representation learning for decision-making of autonomous driving. *IEEE Transactions on Intelligent Vehicles*, 9(3):4405–4421.

[74] Liu, J., Hang, P., Na, X., Huang, C., and Sun, J. (2024b). Cooperative decision-making for cavs at unsignalized intersections: A marl approach with attention and hierarchical game priors. *IEEE Transactions on Intelligent Transportation Systems*, 26:443–456.

[75] Liu, J., Wang, H., Peng, L., Cao, Z., Yang, D., and Li, J. (2022). Pnnuad: Perception neural networks uncertainty aware decision-making for autonomous vehicle. *IEEE Transactions on Intelligent Transportation Systems*, 23:24355–24368.

[76] Liu, L., Lu, S., Zhong, R., Wu, B., Yao, Y., Zhang, Q., and Shi, W. (2020). Computing systems for autonomous driving: State of the art and challenges. *IEEE Internet of Things Journal*, 8:6469–6486.

[77] Liu, Y., Wu, F., Liu, Z., Wang, K., Wang, F., and Qu, X. (2023). Can language models be used for real-world urban-delivery route optimization? *The Innovation*, 4(6).

[78] Lu, C., Willi, T., Letcher, A., and Foerster, J. N. (2023). Adversarial cheap talk. In *Proc. ICML*, pages 22917–22941. PMLR.

[79] Ma, O., Pu, Y., Du, L., Dai, Y., Wang, R., Liu, X., Wu, Y., and Ji, S. (2024). SUB-PLAY: Adversarial policies against partially observed multi-agent reinforcement learning systems. *arXiv preprint arXiv:2402.03741*.

[80] Mandlekar, A., Zhu, Y., Garg, A., Fei-Fei, L., and Savarese, S. (2017). Adversarially robust policy learning: Active construction of physically-plausible perturbations. In *Proc. IROS*, pages 3932–3939.

[81] Matiisen, T., Oliver, A., Cohen, T., and Schulman, J. (2020). Teacher–student curriculum learning. *IEEE Transactions on Neural Networks and Learning Systems*, 31(9):3732–3740. Original preprint arXiv:1707.00183.

[82] Milakis, D., Arem, B., and Wee, B. (2017). Policy and society related implications of automated driving: A review of literature and directions for future research. *Journal of Intelligent Transportation Systems*, 21:324 – 348.

[83] Mirchevska, B., Pek, C., Werling, M., Althoff, M., and Boedecker, J. (2018). High-level decision making for safe and reasonable autonomous lane changing using reinforcement learning. In *2018 21st international conference on intelligent transportation systems (ITSC)*, pages 2156–2162. IEEE.

[84] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

[85] Mohammadhasani, A., Mehrivash, H., Lynch, A., and Shu, Z. (2021). Reinforcement learning based safe decision making for highway autonomous driving. *arXiv preprint arXiv:2105.06517*.

[86] Moos, J., Hansel, K., Abdulsamad, H., Stark, S., Clever, D., and Peters, J. (2022). Robust reinforcement learning: A review of foundations and recent advances. *Machine Learning and Knowledge Extraction*, 4(1):276–315.

[87] Narvekar, S., Sinapov, J., Leonetti, M., and Stone, P. (2020). Curriculum learning for reinforcement learning domains: A framework and survey. *Journal of Machine Learning Research*, 21(181):1–50.

[88] Nilim, A. and Ghaoui, L. E. (2005). Robust control of markov decision processes with uncertain transition matrices. *Operations Research*, 53(5):780–798.

[89] Nordeus, E. (2022). Self driving vehicle. https://github.com/Habrador/Selfdriving-vehicle.

[90] NVIDIA Omniverse (2023). Nvidia physx. https://github.com/NVIDIA-Omniverse/PhysX. GitHub repository.

[91] Ochs, S., Doll, J., Heinrich, M., Schörner, P., Klemm, S., Zofka, M., and Zöllner, J. M. (2024). Leveraging swarm intelligence to drive autonomously: A particle swarm optimization based approach to motion planning. *ArXiv*, abs/2404.02644.

[92] Organization, W. H. (2018). *Global Status Report on Road Safety 2018*. World Health Organization, Geneva.

[93] Paden, B., Čáp, M., Yong, S. Z., Yershov, D., and Frazzoli, E. (2016). A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles*, 1(1):33–55.

[94] Pan, X., Seita, D., Gao, Y., and Canny, J. (2019). Risk averse robust adversarial reinforcement learning. In *Proc. ICRA*, pages 8522–8528. IEEE.

[95] Park, Y., Jun, W., and Lee, S. (2025). A comparative study of deep reinforcement learning algorithms for urban autonomous driving: Addressing the geographic and regulatory challenges in carla. *Applied Sciences*, 15(12):6838.

[96] Pattanaik, A., Tang, Z., Liu, S., Bommannan, G., and Suresh, K. (2018). Robust deep reinforcement learning with adversarial attacks. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 2040–2042.

[97] Pendleton, S. D., Andersen, H., Du, X., Shen, X., Meghjani, M., Eng, Y. H., Rus, D., and Ang, M. H. (2017). Perception, planning, control, and coordination for autonomous vehicles. *Machines*, 5(1):6.

[98] Pérez-Gil, Ó., Barea, R., López-Guillén, E., Bergasa, L. M., Gomez-Huelamo, C., Gutiérrez, R., and Diaz-Diaz, A. (2022). Deep reinforcement learning based control for autonomous vehicles in carla. *Multimedia Tools and Applications*, 81(3):3553–3576.

[99] Petereit, J., Emter, T., Frey, C. W., Kopfstedt, T., and Beutel, A. (2012). Application of hybrid a* to an autonomous mobile robot for path planning in unstructured outdoor environments. In *ROBOTIK 2012; 7th German conference on Robotics*, pages 1–6. VDE.

[100] Pinto, L., Davidson, J., Sukthankar, R., and Gupta, A. (2017). Robust adversarial reinforcement learning. In *Proc. ICML*, pages 2817–2826. PMLR.

[101] Polack, P., Altché, F., d'Andréa Novel, B., and de La Fortelle, A. (2017). The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles? In *2017 IEEE intelligent vehicles symposium (IV)*, pages 812–818. IEEE.

[102] Portelas, R., Colas, C., Hofmann, K., and Oudeyer, P.-Y. (2020a). Teacher algorithms for curriculum learning of deep rl in parameterized environments. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, volume 119 of *Proceedings of Machine Learning Research*, pages 7805–7815.

[103] Portelas, R., Stúrner, C. C., Mary, J., Colas, C., Hofmann, K., and Oudeyer, P.-Y. (2020b). Automatic curriculum learning for deep reinforcement learning: A short survey. *arXiv preprint arXiv:2003.04664*.

[104] Qu, X., Lin, H., and Liu, Y. (2023). Envisioning the future of transportation: Inspiration of chatgpt and large models.

[105] Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. (2021). Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8.

[106] Raju, K. (2020). Autonomous car parking using ml-agents. https://medium.com/xrpractices/autonomous-car-parking-using-ml-agentsd780a366fe46. Medium article.

[107] Rakhsha, A., Radanovic, G., Devidze, R., Zhu, X., and Singla, A. (2020). Policy teaching via environment poisoning: Training-time adversarial attacks against reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, volume 119 of *PMLR*, pages 7974–7984.

[108] Reiter, R., Hoffmann, J., Reinhardt, D., Messerer, F., BaumgÃĪrtner, K., Sawant, S., Boedecker, J., Diehl, M., and Gros, S. (2025). Synthesis of model predictive control and reinforcement learning: Survey and classification. *arXiv preprint arXiv:2502.02133*.

[109] Rummery, G. A. and Niranjan, M. (1994). On-line q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166, University of Cambridge, Department of Engineering. Introduces the SARSA update rule.

[110] Ryou, G., Tal, E., and Karaman, S. (2022). Cooperative multi-agent trajectory generation with modular bayesian optimization. *arXiv preprint arXiv:2206.00726*.

[111] Schulman, J., Levine, S., Moritz, P., Jordan, M. I., and Abbeel, P. (2017a). Trust region policy optimization.

[112] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017b). Proximal policy optimization algorithms. *arXiv:1707.06347*.

[113] Schwarting, W., Alonso-Mora, J., and Rus, D. (2018). Planning and decision-making for autonomous vehicles. *Annual Review of Control, Robotics, and Autonomous Systems*, 1:187–210.

[114] Sharif, A. and Marijan, D. (2022). Adversarial deep reinforcement learning for improving the robustness of multi-agent autonomous driving policies. In *Asia-Pacific Software Engineering Conference (APSEC)*.

[115] Sharma, O., Sahoo, N. C., and Puhan, N. B. (2021). Recent advances in motion and behavior planning techniques for software architecture of autonomous vehicles: A state-of-the-art survey. *Engineering Applications of Artificial Intelligence*, 101:104211.

[116] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., and Hassabis, D. (2017). Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359.

[117] Soviany, P., Ionescu, R. T., Rota, P., and Sebe, N. (2021). Curriculum learning: A survey. *arXiv preprint arXiv:2101.10382*.

[118] Stoler, B., Navarro, I., Francis, J., and Oh, J. (2025). Seal: Towards safe autonomous driving via skill-enabled adversary learning for closed-loop scenario generation. *IEEE Robotics and Automation Letters*, 10(9):9320–9327.

[119] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition.

[120] Taghavifar, H., Hu, C., Wei, C., Mohammadzadeh, A., and Zhang, C. (2025). Behaviorally-aware multi-agent rl with dynamic optimization for autonomous driving. *IEEE Transactions on Automation Science and Engineering*, 22:10672–10683.

[121] Tallec, C., Blier, L., and Ollivier, Y. (2019). Making deep q-learning methods robust to time discretization. pages 6096–6104.

[122] Tamar, A., Glassner, Y., and Mannor, S. (2015). Optimizing the cvar via sampling. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 2993–2999.

[123] Tang, X., Yang, K., Wang, H., Wu, J., Qin, Y., Yu, W.-H., and Cao, D. (2022). Prediction-uncertainty-aware decision-making for autonomous vehicles. *IEEE Transactions on Intelligent Vehicles*, 7:849–862.

[124] Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. (2017). Domain randomization for transferring deep neural networks from simulation to the real world. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[125] Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE.

[126] Towers, M., Terry, J. K., Kwiatkowski, A., Balis, J. U., De Cola, G., Deleu, T., et al. (2023). Gymnasium.

[127] Treiber, M., Hennecke, A., and Helbing, D. (2000). Congested traffic states in empirical observations and microscopic simulations. *Physical review E*, 62(2):1805.

[128] Unreal Engine (2023). Unreal engine. https://www.unrealengine.com/en-US. Website.

[129] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

[130] Wang, C., Li, M., Zhang, M., and Zhang, M. (2025). Deep reinforcement learning for autonomous driving with multiple expert demonstrations. In *2025 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8.

[131] Wang, J., Sun, H., and Zhu, C. (2023a). Vision-based autonomous driving: A hierarchical reinforcement learning approach. *IEEE Transactions on Vehicular Technology*, 72:11213–11226.

[132] Wang, L., Fernández, C., and Stiller, C. (2023b). High-level decision making for automated highway driving via behavior cloning. *IEEE Transactions on Intelligent Vehicles*, 8:923–935.

[133] Wang, T. T., Gleave, A., Tseng, T., Pelrine, K., Belrose, N., Miller, J., Dennis, M. D., Duan, Y., Pogrebniak, V., Levine, S., and Russell, S. (2023c). Adversarial policies beat superhuman go ais. In *Proc. ICML*, pages 35655–35739. PMLR.

[134] Wang, X., Chen, Y., and Zhu, W. (2021). A survey on curriculum learning. *IEEE transactions on pattern analysis and machine intelligence*, 44(9):4555–4576.

[135] Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3-4):279–292.

[136] Wu, F., Li, L., Huang, Z., Vorobeychik, Y., Zhao, D., and Li, B. (2022). Crop: Certifying robust policies for reinforcement learning through functional smoothing. In *International Conference on Learning Representations (ICLR)*.

[137] Wu, Y., Liao, S., Liu, X., Li, Z., and Lu, R. (2021). Deep reinforcement learning on autonomous driving policy with auxiliary critic network. *IEEE transactions on neural networks and learning systems*, 34(7):3680–3690.

[138] Wymann, B., Dimitrakakis, C., Sumner, A., Espié, E., Guionneau, C., and Coulom, R. (2013). Torcs: The open racing car simulator.

[139] Xia, X., Meng, Z., Han, X., Li, H., Tsukiji, T., Xu, R., Zheng, Z., and Ma, J. (2023). An automated driving systems data acquisition and analytics platform. *Transportation Research Part C: Emerging Technologies*.

[140] Xu, C., Ding, W., Lyu, W., Liu, Z., Wang, S., He, Y., Hu, H., Zhao, D., and Li, B. (2022a). Safebench: A benchmarking platform for safety evaluation of autonomous vehicles. *Advances in Neural Information Processing Systems*, 35:25667–25682.

[141] Xu, C., Liu, J., Fang, S., Cui, Y., Chen, D., Hang, P., and Sun, J. (2025). Tell-drive: Enhancing autonomous driving with teacher llm-guided deep reinforcement learning. *arXiv preprint arXiv:2502.01387*.

[142] Xu, D., Chen, Y., Ivanovic, B., and Pavone, M. (2022b). Bits: Bi-level imitation for traffic simulation. *arXiv preprint arXiv:2208.12403*.

[143] Zheng, X., Ma, X., Wang, S., Wang, X., Shen, C., and Wang, C. (2024). Toward evaluating robustness of reinforcement learning with adversarial policy. In *Proc. IEEE/IFIP DSN*. IEEE.

[144] Zhou, M., Luo, J., Villella, J., Yang, Y., Rusu, D., Miao, J., Zhang, W., Alban, M., Fadakar, I., Chen, Z., et al. (2021). Smarts: An open-source scalable multi-agent rl training school for autonomous driving. In *Conference on robot learning*, pages 264–285. PMLR.