Non-standard reinforcement learning for autonomous driving

Lorenzo Menchini & Christian Petruzzella

1 Introduction

1.1 Project Overview

This project investigates a reinforcement learning framework for autonomous vehicle control, with a specialized focus on developing safe overtaking behaviors in multi-obstacle environments. The study implements a lexicographic reinforcement learning approach that hierarchically prioritizes safety constraints while maintaining driving efficiency. The methodology features a progressive training curriculum where the agent advances through three distinct scenario complexities, from basic single-obstacle navigation to challenging multi-vehicle interactions, coupled with an adaptive difficulty mechanism that scales with the agent's performance.

1.2 Objectives and Challenges

The core objective of this project was to train an autonomous agent to handle overtaking maneuvers in the presence of static and dynamic obstacles, while preserving safe and lane-compliant driving behavior. In particular, the agent was expected to learn to initiate lane changes only when necessary and safe, and to prioritize staying in its lane whenever possible.

A key challenge addressed was teaching the agent to decelerate and wait when the overtaking lane is temporarily blocked, instead of attempting unsafe or premature maneuvers. This behavior had to emerge through reinforcement learning in environments where anticipating and reacting to traffic conditions is crucial for long-term success.

To achieve this, a series of progressively complex scenarios were designed. These included situations with a single distant obstacle, critical conditions involving nearby vehicles and limited maneuvering space, and more advanced configurations with multiple vehicles traveling in the opposite lane at varying speeds. Each scenario targeted specific behaviors such as proper use of acceleration, risk-aware decision-making, and the ability to evaluate when and how to perform an overtaking maneuver safely.

The main technical challenges involved the design of scenarios that are realistic yet tractable for simulation, the implementation of reward functions that could balance competing priorities (lane keeping, progress, safety), and the emergence of long-term strategies without explicit supervision. Particularly in multi-vehicle settings, ensuring the agent could distinguish between safe waiting and safe passing required fine control over visual range, obstacle dynamics, and the agent's understanding of temporal opportunity.

1.3 Considered Scenarios

To support the learning of overtaking behaviors and safe decision-making under uncertainty, three main types of scenarios were designed and implemented, each introducing increasing levels of complexity and risk.

1. Easy Scenario – Distant Obstacle

In this configuration, a static vehicle is placed far ahead in the agent's lane, leaving ample space and time for the agent to safely initiate and complete an overtaking maneuver. This scenario aims to reinforce basic overtaking execution while preserving adherence to the ego lane when possible.

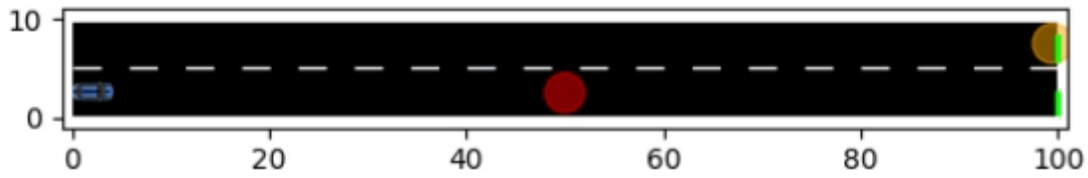


Figure 1: Easy scenario with a distant obstacle.

2. Critical Scenario – Immediate Obstacle

Here, the obstacle is positioned much closer to the pedestrian and to the vehicle, reducing reaction time and spatial flexibility. This setup forces the agent to assess whether the overtaking lane is accessible, and if not, to decelerate and wait. It is instrumental in encouraging the emergence of safe stopping behavior when the overtaking lane is temporarily blocked.

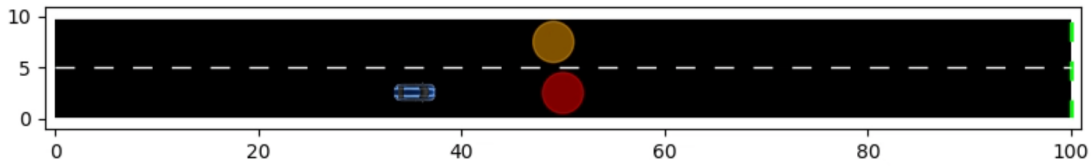


Figure 2: Critical scenario with a close obstacle and blocked lane.

3. Complex Scenario – Multi-Vehicle Interaction

In the most challenging setting, two vehicles are introduced in the oncoming lane, moving towards the agent with variable speeds and randomized positions. This scenario tests the agent's ability to handle uncertainty, anticipate future openings in traffic, and time its overtaking maneuvers without compromising safety or lane discipline. It closely mirrors real-world traffic negotiation problems where decisions must account for dynamic and possibly conflicting elements.

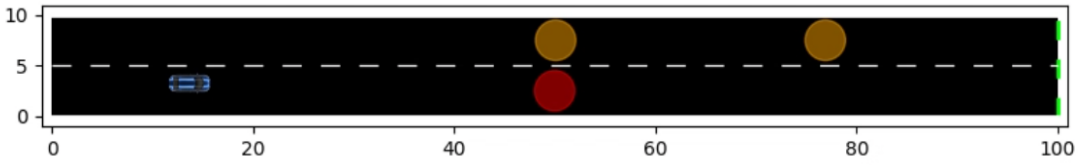
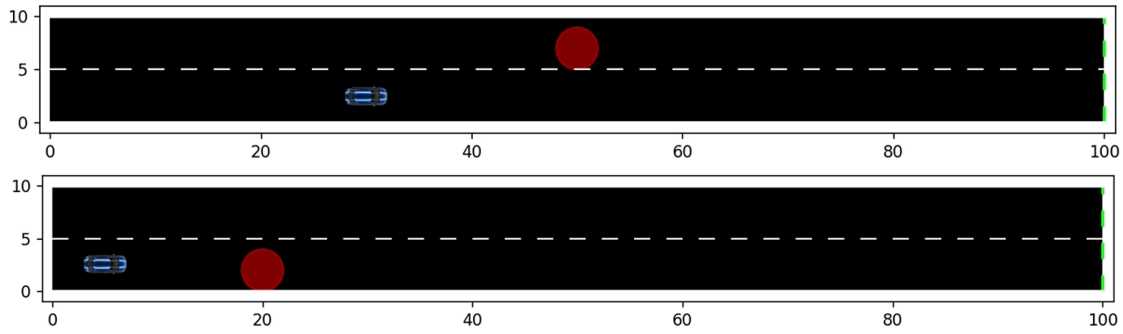


Figure 3: Complex scenario with two oncoming vehicles.

Bonus Scenario - Crossing Jaywalker

This separate scenario introduces the movement of the pedestrian, that crosses the road at various locations and directions, obstructing the agent's trajectory. The agent has to deal with four different crossing configurations, varying both the jaywalker's entry point (far or close to the car) and direction (upward or downward), that are cycled automatically.



By progressively exposing the agent to these configurations, the training process improved understanding of traffic context, situational awareness, and the tactical patience required to execute overtaking actions only when conditions are favorable.

2 Methodology

2.1 Approach

The project employs a lexicographic reinforcement learning approach to train an autonomous vehicle in a simulated environment. Instead of relying on a single aggregated reward signal, the proposed method employs multi-objective RL with lexicographic optimization, ensuring safety (collision avoidance) takes precedence over efficiency (goal-reaching and lane centering).

2.2 Training Pipeline

The training process was structured around a progressive curriculum strategy, designed to expose the agent to increasingly challenging traffic conditions as its policy matured. This curriculum was implicitly driven by the agent's exploration parameter (epsilon), which decayed throughout training, and influenced both scenario selection and the internal dynamics of each environment.

Three distinct environment configurations were defined:

Easy environment: activated during early training when epsilon was high, this scenario involved a single static vehicle placed far ahead of the agent, allowing safe and straightforward overtaking without time pressure or the need to decelerate.

Critical environment: triggered more frequently as epsilon decreased. Here, a vehicle was positioned much closer to the pedestrian and directly in the ego lane, simulating an obstructed path. This setting required the agent to slow down or come to a stop if the overtaking lane was temporarily blocked. Notably, in the later stages of training (epsilon = 0.1), the obstacle's velocity was dynamically reduced based on the agent's average task completion rate using the formula below.

$$v' = v - \text{completed\_rate}$$

where v' is the new velocity for the current episode, v is the base velocity and completed_rate is a parameter $\in [0,1]$ meaning the mean of completed episodes in the last 31.
This mechanic encouraged the agent to wait longer, rather than forcing risky maneuvers.

Complex environment: introduced only when epsilon reached its minimum threshold (`0.01`), this scenario involved two dynamic obstacles traveling in the oncoming lane. Their initial positions were randomized, and their velocities varied within a bounded range. This setup represented the highest level of complexity and required the agent to anticipate openings in traffic, make predictive decisions, and time overtaking maneuvers precisely.

The environment selection mechanism was tightly coupled with the training loop. At each episode, the current value of epsilon and the recent average of completed episodes (`completed_mean`) were used to determine which scenario to present. When exploration was high, simpler situations were favored to build

foundational behavior. As epsilon decreased and the agent exhibited stable performance, more complex and less forgiving environments were presented.

This gradual increase in difficulty allowed the agent to build reliable low-level behaviors before confronting high-risk scenarios.

2.3 State Representation

The state is a 8-dimensional vector constructed as follows:

Index	Component	Description
0	`car.position[1]`	Lateral position (y-coordinate) of the car relative to the road. Tracks deviation from the lane center.
1	`inv_distance_to_jaywalker`	Inverse distance to the jaywalker. Set to zero if the pedestrian is behind the car or beyond the sight range (60 units). Higher values indicate closer proximity.
2	`angle_to_jaywalker`	Angular offset (in radians) between the car's heading direction and the vector to the jaywalker. Returns $-\pi$ when the pedestrian is behind.
3	`car.v`	Current velocity of the car (bounded between ± 20 units/sec). Positive values indicate forward motion.
4	`car.beta`	Slip angle induced by steering. Represents lateral deviation of the car's trajectory. Positive values correspond to right turns.
5	`inv_distance_to_obstacle`	Inverse distance to the closest vehicle ahead in either lane (if visible). Zero if no relevant obstacle is within the observation range (80 units).
6	`angle_to_obstacle`	Angular deviation (in radians) between the car's heading and the closest detected obstacle. Returns $-\pi$ if no obstacle is present or if it is behind.
7	`lane_index`	Index of the current lane the car is occupying (0 or 1). Used to encode discrete lane positioning.

2.4 Reward Function

The reward function is designed as a multi-objective vector that explicitly prioritizes safety over navigation efficiency by combining three critical components in a lexicographic hierarchy:

1. **Collision Avoidance**: penalty (`-10`) applied if the agent collides with the jaywalker or other obstacles in order to discourage unsafe actions and prioritize obstacle avoidance

2. **Forward Progress**: incremental reward scaled by the distance traveled toward the goal that encourages the agent to make steady progress along the road

   ```
   (car.position[0] - car.prev_position[0]) / scale_factor
   ```

3. **Out of Bounds and Lane Keeping**: strong penalty triggered if the car exits the road boundaries

   ```
   - 1000 / (scale factor * 10)
   ```

   or if the episode is completed a penalty proportional to the deviation from the lane center is applied in order to promote lane adherence:

   ```
   - |car.position[1] - goal[1]| / (scale factor * 10)
   ```

2.5 Action Space

The action space consists of a combination of steering angles (df) and acceleration values (a). Specifically:

Steering Actions

- $-\pi/18$ (left turn)
- 0 (no turn)
- $+\pi/18$ (right turn)

Acceleration Actions

- -2 (decelerate)
- 0 (maintain speed)
- $+2$ (accelerate)

Moreover, the vehicle's speed is constrained within a range of `[-20, 20]` and the effective steering angle (`beta`) accounts for the vehicle's geometry (wheelbase `lf` and `lr`), ensuring that the turns are physically plausible.

Action selection follows a lexicographic priority scheme, where the agent first prioritizes collision avoidance, then progress toward the goal, and finally lane centering. During exploration, the agent occasionally selects random actions with probability *epsilon*, which decays over time to shift from exploration to exploitation.

3 Implementation

3.1 Initial Setup

The original implementation served as a foundation framework, focusing on a simplified environment with a single jaywalker obstacle.

3.1.1 Environment Design

The driving scenario is modeled as a 2D rectangular road (100 units long x 10 units wide) with the following elements:

- A car initialized at position (`0, 2.5`) with discrete control over acceleration (`-5, 0, +5` units).
- A static pedestrian represented as a circular obstacle (radius = `2` units) placed midway at (`50, 2.5`).
- Target position for the car at (`100, 2.5`)

3.1.2 Technical Implementation

- RL algorithm: Deep Q-Network (DQN) with experience replay and target network stabilization.
- Neural Network: A 4-layer MLP (128 hidden units per layer) with batch normalization.
- Training Parameters:
  - Learning rate: `1e-2`
  - Discount factor (γ): `0.95`
  - Exploration: *epsilon*-greedy (ϵ decayed from `1.0` to `0.01` over episodes)
  - Batch size: `256`
  - Replay buffer: `10,000` iterations

3.1.3 Limitations of the Original Implementation

Two key limitations of the initial setup warrant particular discussion.

First, the static nature of the jaywalker scenario, while useful to establish baseline performance metrics, the environment failed to capture critical temporal aspects of obstacle avoidance that would become essential in more complex implementations, and limited the system's ability to learn generalized collision avoidance strategies for moving obstacles.

Second, the episode length constraint of just 100 iterations created artificial time pressure that may have influenced the agent's learning priorities. While this compact time horizon enabled rapid experimentation cycles, it potentially discouraged the development of more sophisticated, long-term navigation strategies. In particular, the short episodes particularly affected scenarios where careful speed modulation or temporary stops might be optimal solutions, as the agent was incentivized to reach the goal quickly rather than develop nuanced behaviors.

These constraints were later addressed through learning approaches that progressively introduced dynamic elements and extended operational timeframes.

3.2 Modifications & Improvements

The most substantial changes were made to the `Jaywalker` environment class to create a more realistic and challenging simulation.

- **Dynamic Obstacles:** The original environment featured only a single, static "jaywalker" obstacle. The new version introduces dynamic obstacles, simulated as other cars on the road.
  - These obstacles are managed in a list (`self.obstacles`).
  - They are initialized during the `env.reset()` call and can move towards the agent, forcing it to learn more complex avoidance maneuvers rather than simply bypassing a static point.
- **Multi-Lane Environment:** The road is now explicitly defined as having multiple lanes (`self.n_lanes`). This adds structure to the environment and is crucial for developing lane-changing behaviors.
- **Advanced Collision Detection:** A new method, `collision_with_obstacle()`, was added to handle collisions with the new dynamic obstacles, complementing the original `collision_with_jaywalker()`.

3.2.1 Curriculum Learning Implementation

To manage the increased difficulty, a curriculum learning strategy has been implemented. This allows the agent to learn fundamental skills in a simpler context before being exposed to the full complexity of the task.

- **Two-Stage Training:** The training process is now divided into two stages, managed by the `curriculum_stage` attribute in both the `QAgent` and `Jaywalker` classes.
  - **Stage 1:** The agent is trained on a mix of "easy" and "critical" scenarios. In the "easy" scenario, the obstacles are placed far away, allowing the agent to focus on reaching the goal. In the "critical" scenario, an obstacle is placed closer. This alternation prevents the agent from overfitting to a single, simple situation.
  - **Stage 2:** After the agent has reached a certain level of proficiency (determined by the epsilon value dropping below a threshold), it transitions to a consistently difficult environment with multiple, actively moving obstacles.
- **Adaptive Difficulty:** In Stage 2, the speed of the obstacles is dynamically adjusted based on the agent's recent performance (`self.completed_mean`). If the agent is succeeding, the obstacles move faster, increasing the challenge. This ensures the agent is always learning at the edge of its capabilities.

3.2.2 State Representation Modification

To accommodate the new environmental features, the agent's perception has been updated.

- **Expanded State Representation:** The state vector that the agent receives has been expanded from 5 to 8 elements.
  - **Old State** (`state_size = 5`): `[car_y, inv_dist_jaywalker, angle_jaywalker, car_velocity, car_beta]`
  - **New State** (`state_size = 8`): `[car_y, inv_dist_jaywalker, angle_jaywalker, car_velocity, car_beta, inv_dist_obstacle, angle_obstacle, lane_index]`

  The three new elements provide the agent with crucial information about the nearest dynamic obstacle (its inverse distance and angle) and its own current lane index.

3.2.3 Training Process and Agent Refinements

The `QAgent`'s training loop (`learn` method) has been made more robust and intelligent.

- **Automated Curriculum Transition:** The agent automatically progresses from Stage 1 to Stage 2 of the curriculum when its exploration rate (`epsilon`) falls below a set threshold, indicating that it has stabilized its initial policy.
- **Advanced Model Saving & Early Stopping:** A new mechanism was introduced to save the model's state. It tracks the moving average of the completion rate and saves a model checkpoint if the agent achieves a high completion rate with zero collisions. An early stopping condition terminates the training if the agent maintains a high success rate for a set number of consecutive episodes.
- **Increased Memory:** The `memory_length` of the experience replay buffer was drastically increased from `10,000` to `1,500,000`. This provides the agent with a much larger and more diverse set of past experiences to learn from.

3.2.4 Visualization and Debugging

- **Live Rendering:** The new script includes a `render()` method in the `Jaywalker` class, which uses `matplotlib` to provide a live visual representation of the simulation during training. This is an invaluable tool for debugging and understanding the agent's behavior in real-time.

Summary of Key Differences

Feature	`old`	`new_project` **(Improvements)**
Environment	Static, single jaywalker obstacle.	Dynamic, moving obstacles (cars) and a static jaywalker.
Training Strategy	Standard training on a fixed environment.	Two-stage curriculum learning with adaptive difficulty.
State Size	5	8 (adds vision for obstacles and lane index).
Training Loop	Basic loop, runs for a fixed number of episodes.	Advanced loop with auto-curriculum, early stopping, and best-model saving.
Max Iterations per Episode	`100`	`15000` (150x larger).
Memory Length	`10,000`	`1,500,000` (150x larger).
Visualization	None. Plots generated only after training.	Live rendering of the simulation during training.

3.3 Perception & Vision

A key challenge in autonomous driving is empowering the agent with the ability to perceive its surroundings effectively. This capability is implemented through a dual-channel sensory system, designed to distinguish between the primary static obstacle (the "jaywalker") and other dynamic obstacles on the road. Both systems compute the *inverse distance* and *relative angle* of the detected entity from the vehicle.

The perception is handled by two main functions:

- `vision()`: This function is dedicated to detecting the jaywalker. It calculates the vector from the agent to the jaywalker and returns two key values: the inverse of the distance and the relative angle. This perception is active only if the jaywalker is within a predefined sight radius and is positioned in front of the agent; otherwise, the function returns default numerical values (`0` for inverse distance and $-\pi$ for the angle) to indicate the absence of a relevant threat.
- `vision_obstacle()`: This function manages the perception of dynamic obstacles. It identifies the single closest obstacle to the agent and, similar to the primary function, calculates its inverse distance and relative angle. This sensory channel is more focused, operating within a 90-degree forward-facing cone of vision to simulate the driver's focus area and lie within a broader visual range (`sight_obstacle`). If no obstacles are within this cone and range, it returns default values.

A crucial aspect of the implementation lies in how these two sensory channels are integrated during each simulation step (`step`). The perception of the jaywalker via `vision()` is always active, providing a constant baseline awareness. In contrast, the perception of dynamic obstacles via `vision_obstacle()` is subordinate and conditional. Indeed, the obstacle vision is triggered based on a priority mechanism that compares the proximity of the jaywalker with that of the closest obstacle. Specifically, the obstacle vision activates if:

1. The jaywalker has been detected, or
2. the nearest obstacle is closer than the jaywalker.

This logic ensures that obstacle information only competes for attention when it is more immediately relevant than the pedestrian, helping the agent avoid distraction from less urgent stimuli.

3.4 Collision Handling

The system considers two primary types of collisions:

1. Collision with obstacles (pedestrian or other moving vehicles)
2. Non-object collision (out-of-bounds and lane violations)

3.4.1 Obstacle Collision

Detecting a collision with the jaywalker or other vehicles involves a segment-based intersection test between the car's front trajectory and the jaywalker's bounding box. Importantly, the bounding box is updated in real time as the obstacle moves across the lanes. If a collision is detected, the agent receives a negative reward in the first component of the reward vector, and the episode is terminated. This penalty reflects the high cost of hitting an obstacle and forces the agent to learn safe braking or evasive maneuvers early in the training.

3.4.2 Out-of-Bounds Violations

In addition to physical collisions with obstacles, the system penalizes running off the road, which is treated as another termination condition: a vehicle is considered out-of-bounds if any part of its front or rear crosses the upper or lower limits of the road, or if it moves backwards past the starting point. When this happens, the episode is terminated, and a large negative reward is applied to the third reward component.

3.5 Complex Scenarios Integration

The transition from the simplified initial setup to our implementation involved the introduction of dynamic and multi-object scenarios, which progressively increased in complexity:

- Cars spawning in the opposite lane with configurable speeds.
- Geometric and kinematic variations (position, speed) of parametrized obstacles.
- Obstacles appear at variable distances to prevent memorization.

These changes enabled the agent to handle simultaneous interactions (overtaking, avoiding both the jaywalker and the arriving cars) and learn generalizable policies across obstacle configurations.

4 Results & Evaluation

The trained agent was evaluated across all defined scenarios to assess both the quality of its learned behaviors and its ability to generalize beyond the training conditions. Quantitative metrics such as completion rate, collision frequency, and lane stability were monitored across multiple runs for each scenario, providing a comprehensive picture of policy robustness.

Initial evaluations were conducted separately on each scenario using a fixed policy with epsilon set to zero. The agent demonstrated a high success rate in the easy and critical scenarios, consistently executing safe overtaking maneuvers and avoiding collisions. In the complex multi-vehicle scenario, the agent showed reliable anticipation and hesitation behaviors, often waiting for a safe opening before initiating an overtake, even when this required multiple time steps of inactivity.

To ensure that the agent had not overfitted to specific training dynamics, a final evaluation phase was performed in which all three scenarios were presented in randomized succession. This stress test allowed validation of policy flexibility and adaptability to varying spatial-temporal configurations.

In the end, three final models have been evaluated on the same set of scenarios: the model trained on fixed starting speed (`v=0`), a variant with random starting speed, and finally the original "physical" model taken presented in the paper *Kinematic and Dynamic Vehicle Models for Autonomous Driving Control Design* [1], which served as inspiration for our simplified model.

4.1 Performance Metrics

To monitor the learning process and evaluate the agent's behavioral progression over time, six key metrics were recorded and visualized throughout training. Each of these offers insights into different aspects of policy formation, safety, and task success.

Our Proposed Model

1. Completion Rate (Completed) This metric tracks whether the agent successfully reached the goal in each episode. In the corresponding plot, the blue line shows binary outcomes (1 for success, 0 for failure), while the orange line represents a moving average.

At the beginning of training, the completion rate remains near zero, reflecting an untrained policy. A brief increase occurs early due to simpler scenarios, but performance quickly deteriorates as difficulty increases. Around episode 4000, a sustained improvement is visible. The agent steadily progresses toward more stable and reliable performance, eventually achieving a success rate oscillating around 75% at the cutoff of 10,000 episodes. This demonstrates effective learning and behavioral consolidation.

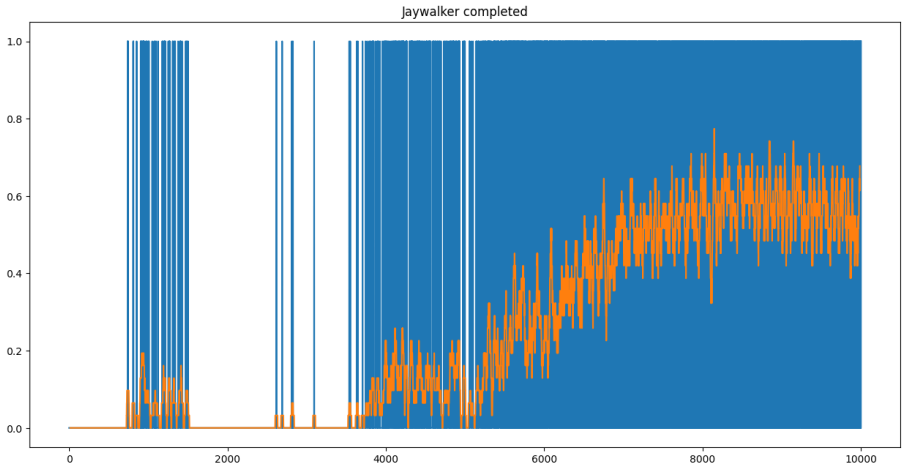


Figure 4: Easy scenario with a distant obstacle.

2. Forward Progress (Distance) This plot shows the average reward associated with forward movement, normalized by episode. Higher values indicate more consistent progress toward the goal.

Initially, the agent exhibits low and highly variable forward movement. Early spikes suggest accidental progress rather than deliberate behavior. As training advances, the forward distance reward stabilizes and rises significantly, peaking around episode 8000, where values consistently exceed 0.9. This trend aligns with the agent's increasing competence in safely navigating toward the goal.

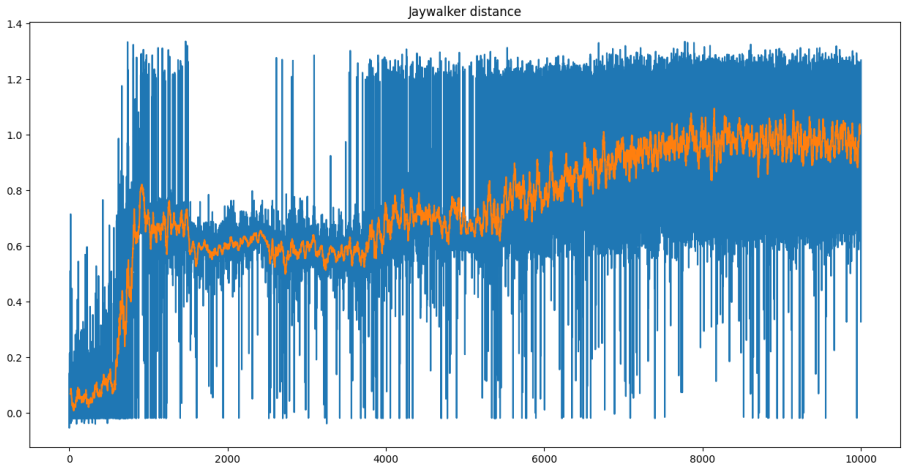


Figure 5: Easy scenario with a distant obstacle.

3. Out-of-Bounds Penalty (OOB) This metric penalizes the agent for leaving the drivable area. A value of -1 corresponds to a full violation, while values closer to 0 indicate better lateral control.

Early in training, the penalty remains near -1, indicating frequent lane departures and instability. Starting around episode 4000, the penalty value gradually rises, showing improved lane discipline. By episode 8000, the agent maintains values consistently above -0.3, demonstrating increased ability to remain within road boundaries while still pursuing the goal.

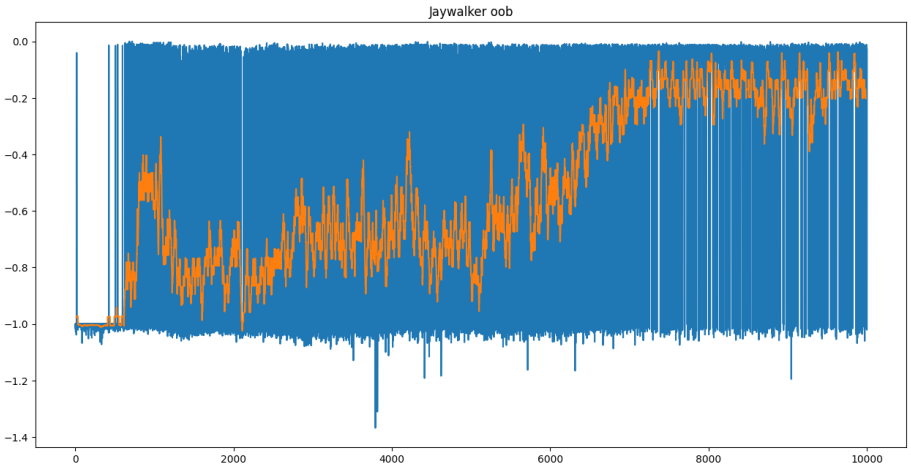
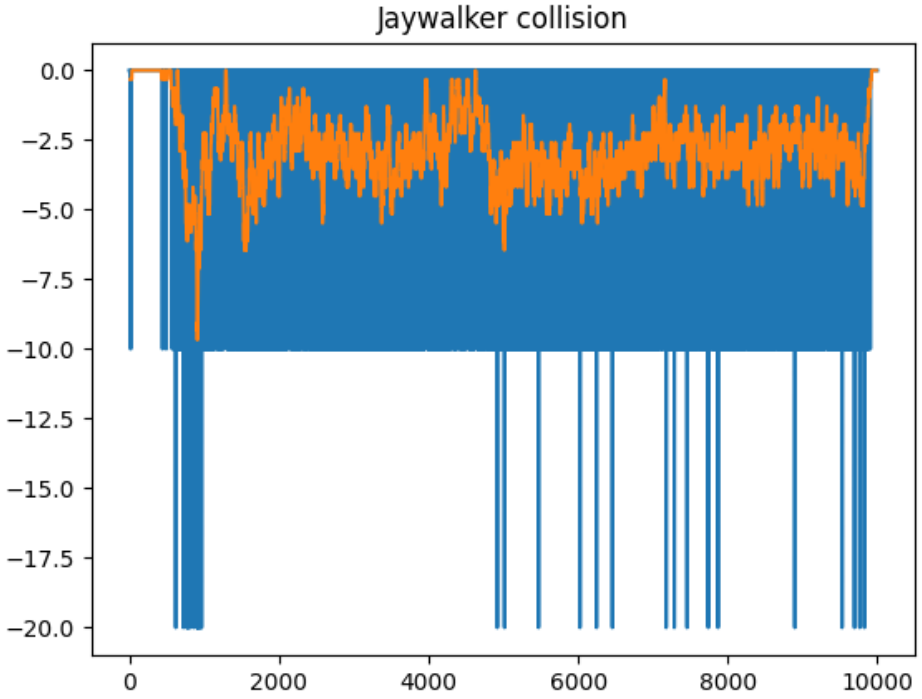


Figure 6: Easy scenario with a distant obstacle.

4. Collision Count The agent initially experiences frequent collisions, as seen in the dense cluster of minimum values in the early training phase. Over time, the frequency of such penalties decreases, although occasional spikes persist, indicating isolated failure cases.

The downward shift in the average penalty during mid-training suggests increased caution and improved obstacle avoidance. However, the long-term stability of the orange curve around -3 to -5, before ending the training approximately with 0 collisions.

This metric is essential for assessing safety and verifying the agent's ability to consistently avoid dangerous interactions.



5. Action Selection This plot summarizes the distribution of actions taken by the agent over time. Patterns in this metric can reflect shifts from exploratory to exploitative behavior and show emerging preferences for safe, efficient maneuvers (e.g., less steering variance, more conservative acceleration in complex scenarios)

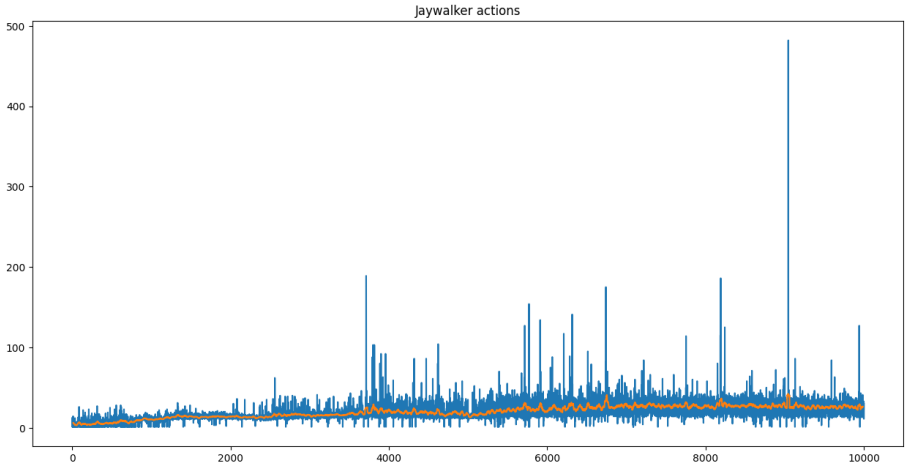


Figure 7: Easy scenario with a distant obstacle.

6. Epsilon Decay This plot reflects the decay of the exploration rate epsilon, which begins near 1 and decays exponentially toward a minimum value (0.01). As epsilon decreases, the agent transitions from random exploration to increasingly deterministic, greedy action selection. This shift is necessary to ensure convergence to a stable policy and is correlated with improved performance in the other metrics.

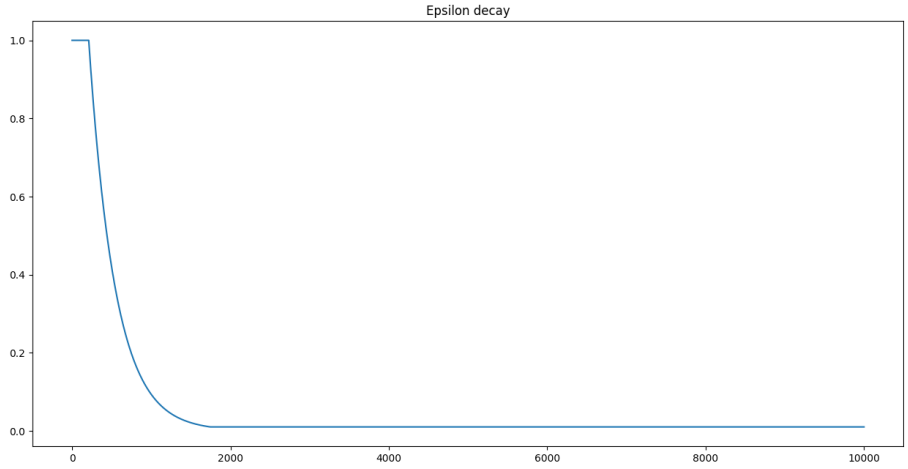
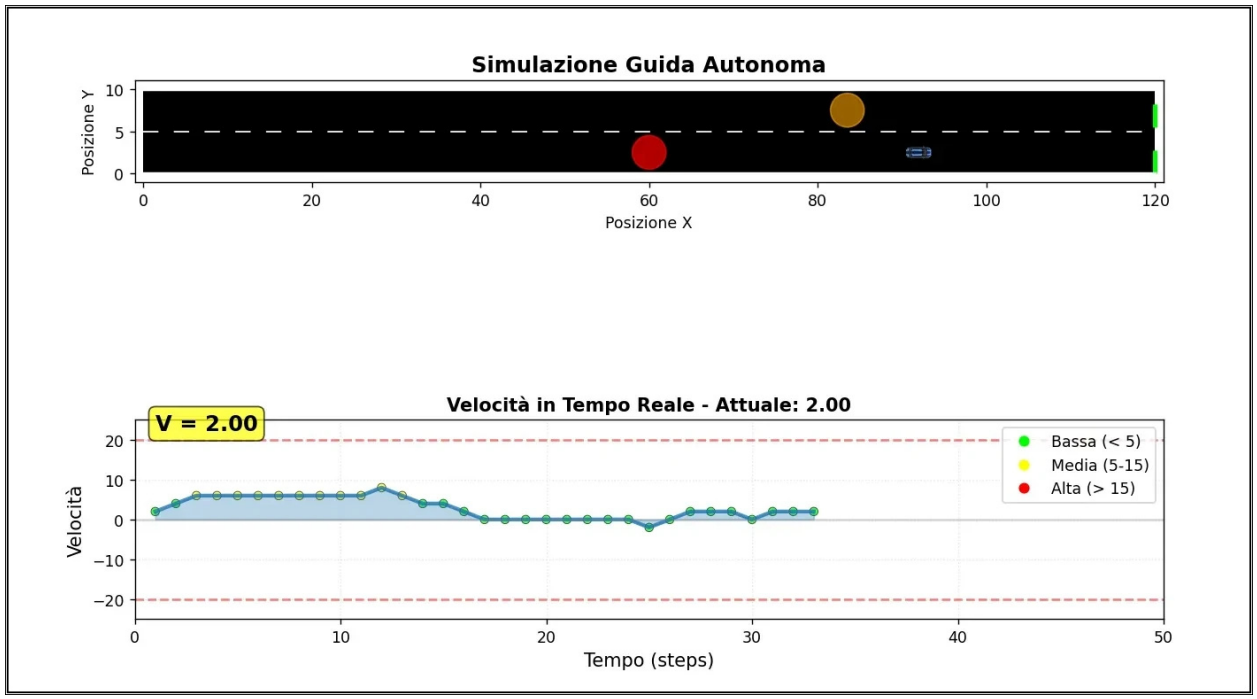


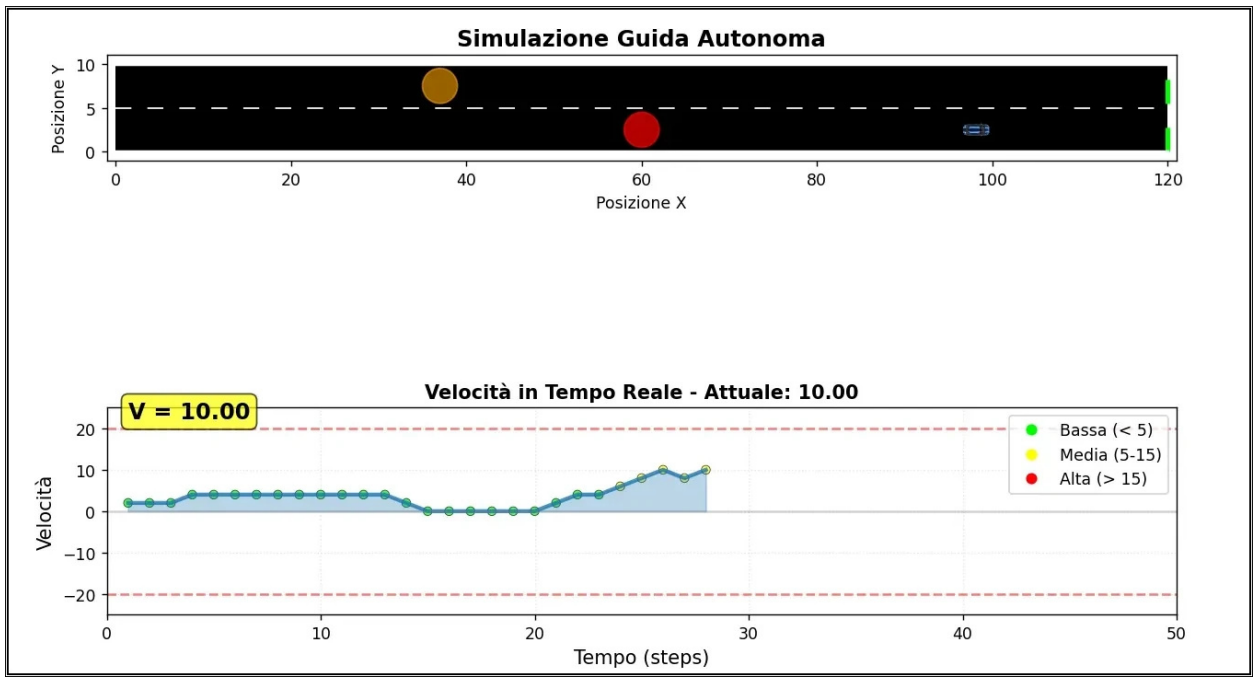
Figure 8: Easy scenario with a distant obstacle.

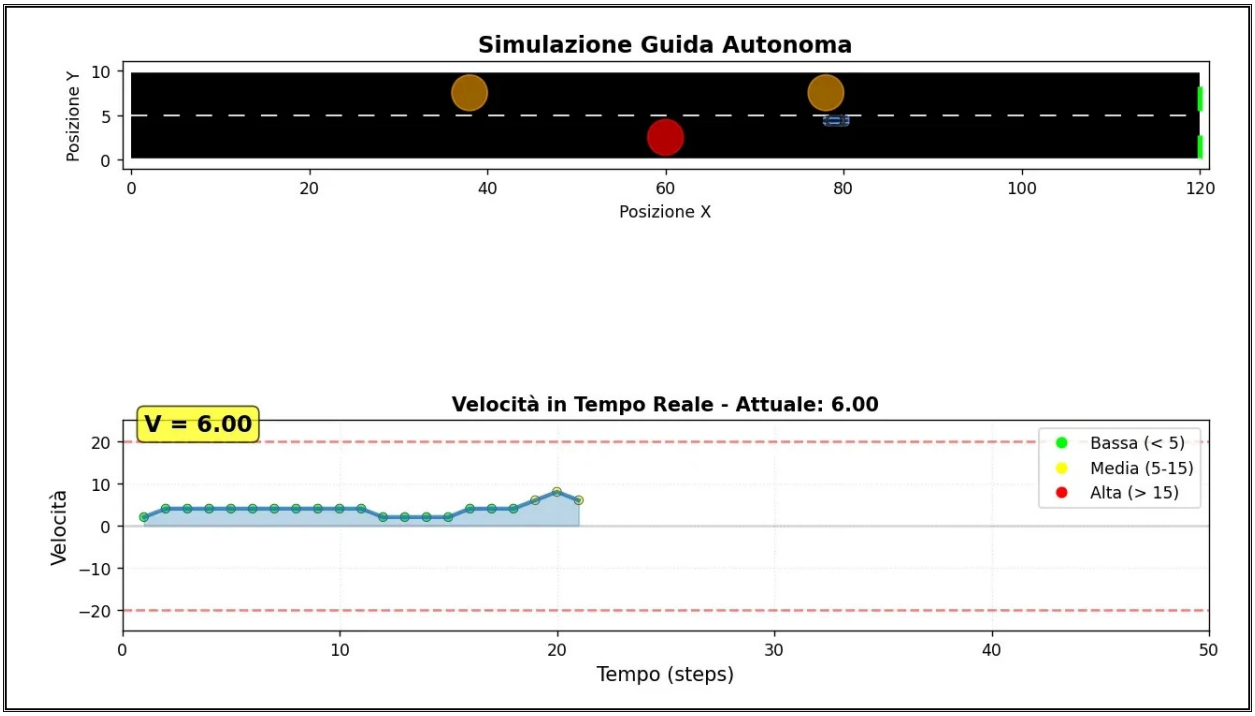
Test Results: The agent maintained high performance levels, confirming that its learned strategy was not merely reactive to fixed patterns but could adaptively handle multiple forms of environmental uncertainty. Moreover, the agent was exposed to additional test conditions involving increased obstacle density and reduced reaction time, which were not part of the original training curriculum. While these configurations

introduced failure cases in some episodes, the agent was still able to generalize core principles such as safe stopping and progressive overtaking behavior, demonstrating policy resilience and strong situational awareness.

- **Scenario 1**: Jaywalker only – 100% completion rate
- **Scenario 2**: Jaywalker with a critical overtaking car – 100% completion rate
- **Scenario 3**: Jaywalker with two oncoming vehicles – 100% completion rate







The results indicate that the agent successfully learned to navigate constrained overtaking tasks with safety and efficiency, and retained robust performance even when deployed in more complex or unfamiliar settings.

Our Proposed Model (varying starting speed)

In this extended version of the model, the initial velocity of the agent is no longer fixed but is instead dynamically sampled from a Gaussian distribution. This change introduces variability in the initial motion conditions, aiming to improve the robustness and adaptability of the learned policy.

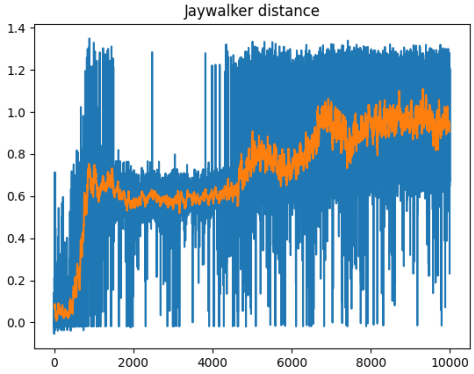
More precisely, the initial velocity v_final is computed as a function of a base velocity and a normally distributed random variable, scaled by the agent's current average task completion rate. The formula ensures that velocity increases both in mean and variance as the agent demonstrates higher proficiency, while avoiding negative values through truncation at zero.

This modification is designed to encourage the agent to learn policies that generalize across a broader range of motion conditions, particularly during overtaking and stopping behaviors. Compared to the previous model, which used a fixed or deterministic speed initialization, this stochastic approach introduces controlled uncertainty that acts as a form of curriculum-dependent domain randomization.

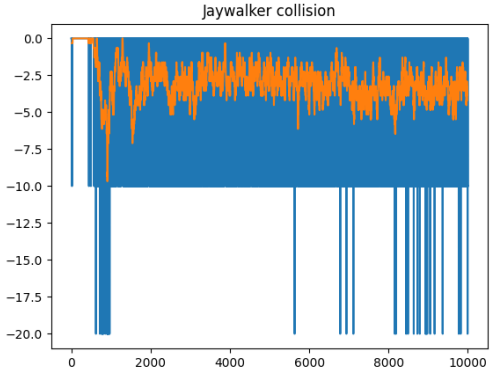
$$v_{\text{final}} = \max\left(0,\ (v_{\text{base}} + \mathcal{N}(0, \sigma)) \cdot \text{completed\_mean}\right)$$

Where:

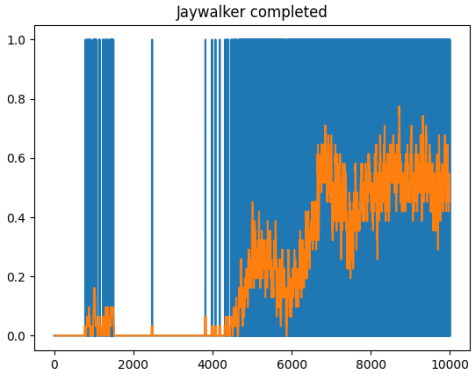
- v_{base} is the base initial speed (set to 2.0),
- $\mathcal{N}(0, \sigma)$ is a normal distribution with mean 0 and standard deviation $\sigma = 1.5$,
- completed_mean $\in [0, 1]$ is the agent's average task completion rate,
- The outer $\max(0, \cdot)$ ensures that the resulting velocity is non-negative.



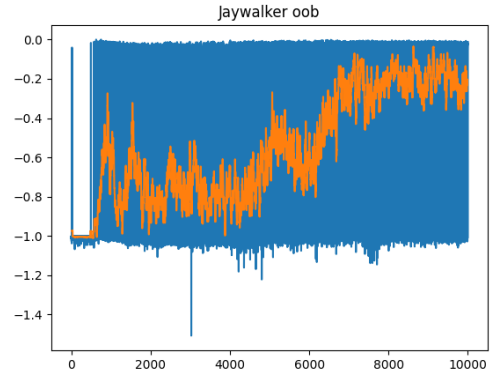
(a) Distance



(b) Collision



(c) Completed



(d) Out of Bounds

Figure 9: Performance metrics across episodes for v random model

Test Results: The results from the test phase indicate that the model is able to generalize its behavior regardless of the randomized initial velocity. Specifically, the initial speed for the test environment was sampled from a normal distribution defined as:

$$v_{\text{final}} = \ v_{\text{base}} + \mathcal{N}(0, \sigma))$$

with $v_{\text{base}} = 2.0$ and $\sigma = 1.5$. The resulting value was directly assigned as the agent's initial speed at the beginning of each episode.

Despite this variability in initial conditions, the model achieved consistent performance across a set of predefined evaluation scenarios. The agent was tested on three environments of increasing difficulty:

- **Scenario 1**: Jaywalker only – 100% completion rate
- **Scenario 2**: Jaywalker with a critical overtaking car – 50% completion rate
- **Scenario 3**: Jaywalker with two oncoming vehicles – 25% completion rate

Notably, these results remained stable across different runs and across varying initial velocities, suggesting that the policy successfully learns to adapt its behavior independently of the starting speed. This demonstrates a degree of robustness and generalization in the learned strategy, even under increased uncertainty in initial conditions.

Original "Physical" Model

The key differences between this model and our simplified implementation are the following:

- The slip angle (`beta`) is updated using

$$\beta = \arctan(\frac{lr}{l_f + l_r} \tan(\delta_f))$$

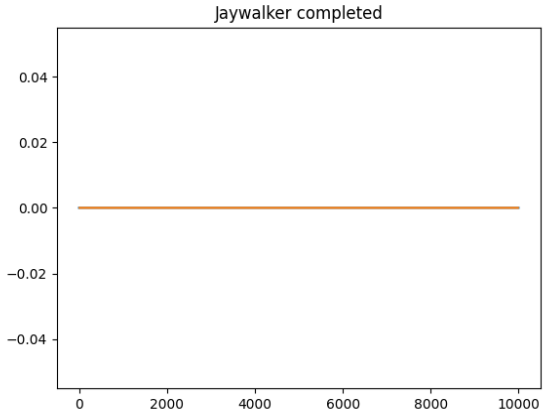
- The heading angle (`phi`) is updated with

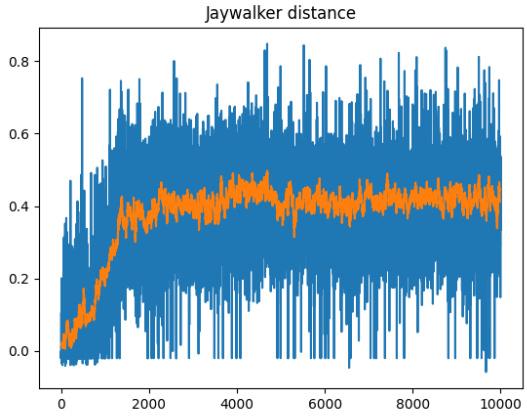
$$\phi = \frac{v}{l_r} sin(\beta)$$

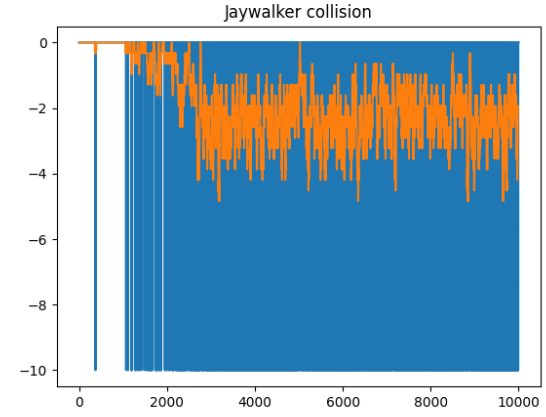
  and it is used to determine the direction the car is pointing, affecting the car's movement when combined with the slip angle (`beta`)

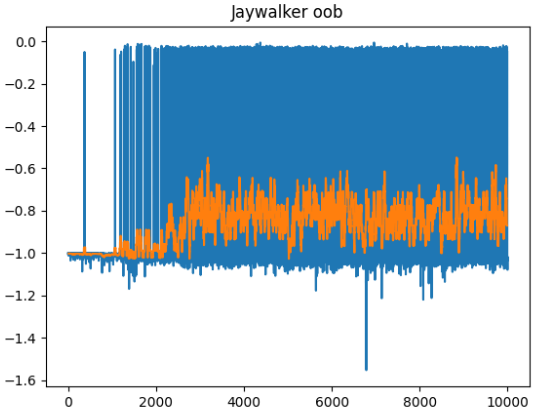
- The previous position of the rear axle of the car (`prev_back`) is used to track the car's movement history, useful when checking for collisions to help determine if the car's trajectory intersected with an obstacle between the last two time steps

Overall, the model struggles significantly with task completion and forward progress. The completion rate is always near zero, indicating that the car rarely reaches the goal. The forward progress metric fluctuates without clear improvement, suggesting that the car moves inefficiently. The collision penalties are around -2 and -4, and the out-of-bounds suggests instability during sharp maneuvers.









4.2 Comparison with Other Existing Approaches

Upon analyzing the Lexicographic Multi-Objective Reinforcement Learning (LMORL) framework [2], we can compare it with our project's approach in terms of methodologies, performance metrics, and outcomes.

4.2.1 Approach

LMORL employs a lexicographic ordering of multiple objectives, prioritizing them hierarchically. This method ensures that higher-priority objectives are satisfied before considering lower-priority ones. Such an approach is particularly beneficial in scenarios where certain safety constraints must be strictly enforced before optimizing for performance or efficiency.

Our Project focuses on training agents across diverse environments to promote generalization and prevent overfitting. By exposing the model to varied scenarios consecutively, we aim to develop a robust policy capable of adapting to unforeseen situations, emphasizing adaptability over strict hierarchical objective satisfaction.

4.2.2 Performance Metrics

LMORL evaluates its agents based on the satisfaction of prioritized objectives, ensuring that primary goals are met before secondary ones. This evaluation is typically qualitative, focusing on whether the agent adheres to the set hierarchy of objectives.

Our Project utilizes quantitative metrics to assess performance comprehensively:

Action: Monitors the agent's decision-making patterns over time.

Collision: Tracks the frequency of collisions, indicating safety levels.

Completed: Measures the success rate in achieving set tasks.

Distance: Calculates the proximity to goals, reflecting efficiency.

Epsilon Decay: Observes the exploration-exploitation balance during training.

Out-of-Bounds (OOB): Counts instances where the agent exits permissible zones, highlighting control precision

4.2.3 Results

LMORL demonstrates strong adherence to prioritized objectives, ensuring that critical constraints are never violated. This results in agents that are reliable in environments where certain rules or safety measures are non-negotiable.

Our Project shows that agents trained across varied scenarios can generalize effectively, performing well even in unfamiliar or more complex environments. The quantitative metrics indicate improvements in safety (reduced collisions and OOB incidents) and efficiency (shorter distances to goals and higher task completion rates) over time.

4.2.4 Conclusion

While LMORL excels in environments where strict adherence to a hierarchy of objectives is paramount, our project's strength lies in fostering adaptability and generalization through exposure to diverse scenarios. The choice between the two approaches depends on the specific requirements of the deployment environment: LMORL for strict constraint satisfaction and our method for versatility and adaptability.

4.3 Limitations & Challenges

Although the agent performs well across different overtaking scenarios, the project still has some limitations and challenges that should be considered.

First, the environment used for training is a simplified version of real-world driving. While it includes important elements like lanes, obstacles, and basic dynamics, it does not capture all the complexity of real traffic, such as curved roads, different types of vehicles, or sensor noise. Because of this, the learned behavior may need further adjustments before being applied in more realistic settings.

Another challenge was the difficulty of learning at the beginning of training. At that stage, the agent rarely completes the task successfully, so it must rely on smaller rewards—like making forward progress or staying in its lane—to improve. Finding the right balance between these different rewards was not easy and required careful tuning.

The action space in this project is discrete and limited to a few choices. This makes training easier and more stable, but it also reduces the precision of the agent's movements. In some situations, especially during overtaking, a more continuous set of actions could lead to smoother and more realistic behavior.

The curriculum used to increase scenario difficulty during training helped the agent learn in a structured way. However, this approach also depends on good timing: if the difficult scenarios appear too early, the agent may struggle to learn and fail to improve.

Lastly, even though the final stage of training includes two moving vehicles in the oncoming lane, these vehicles follow fixed paths and do not react to the agent. The agent is not yet tested in fully interactive multi-agent environments, where other vehicles can also make decisions based on the situation. This remains a direction for future work.

5 Future Work & Possible Extensions

5.1 More Complex Simulators

To better approximate real-world driving conditions, the current simulator could be expanded in several meaningful directions.

- A natural progression would be to model multi-lane highways, where the autonomous vehicle must learn to execute safe lane changes, handle merges, and navigate highway interchanges. This extension would require additional lanes and proper modeling of traffic flow patterns and driver interactions during lane transitions.
- Urban driving scenarios could be incorporated through the addition of intersections with traffic signals and stop signs. Here, the agent would need to understand precedence rules, detect traffic light states, and make appropriate stopping decisions.
- The current straight-road geometry could be generalized to include curved road segments, which would challenge the vehicle's steering controller to maintain precise trajectory tracking. Non-linear paths would better validate the robustness of the learned policies, particularly when combined with variable speeds and adverse weather conditions.
- Pedestrian interactions could be made more realistic through the simulation of crowd dynamics, where groups of jaywalkers cross the road according to more sophisticated movement patterns
- To further stress-test the system, the simulator could introduce adversarial drivers that occasionally violate traffic norms through sudden lane changes, unpredictable braking, or other aggressive maneuvers. These challenging edge cases would help develop more robust collision avoidance strategies.

The progressive introduction of complexity could follow a curriculum similar to the existing approach, allowing the autonomous agent to gradually master basic skills before tackling more sophisticated scenarios.

5.2 Real-World Applicability

The ultimate goal is to create a system where policies trained in a simulation can be reliably transferred to physical autonomous vehicles in complex environments:

- Sensor Integration and Perception: actual sensor data do not assume perfect knowledge of object positions, so camera perception models, occlusion handling for partially observable scenarios, and online mapping and localization components could be incorporated.
- Dynamic Environment Adaptation: real-world traffic features constantly evolving scenarios could be addressed through continual learning frameworks that adapt to novel traffic situations.

- Safety Certification: before deployment autonomous systems must demonstrate formal verification of collision avoidance guarantees and robustness testing against adversarial conditions.

This progressive approach to real-world integration would maintain the benefits of simulation-based development while systematically addressing the challenges of physical deployment. Each component could be tested and validated in isolation before being combined into a complete autonomous system.

5.3 Multi-Agent Reinforcement Learning

The current single-agent framework could be significantly extended by introducing multi-agent reinforcement learning capabilities, enabling the study of cooperative and competitive interactions in traffic scenarios. This extension would transform the simulator into an ecosystem of interacting agents, reflecting the complexity of real-world driving.

Cooperative Multi-Agent Systems would allow investigating:

- Vehicle coordination for efficient highway merging
- Collaborative intersection navigation
- Distributed traffic optimization through local
- Emergency vehicle passage coordination

Competitive Scenarios would introduce more realistic challenges:

- Lane change competitions in dense traffic
- Parking spot allocation conflicts
- Adversarial driver behaviors

Technical Implementation Challenges would include:

- Determining individual contributions to shared outcomes
- Adaptive to other learning agents
- Agent-to-agent information exchange
- Maintaining training stability with increasing agent counts

By embracing multi-agent reinforcement learning, this framework could evolve from a controlled training environment into a powerful platform for studying the emergent behaviors and complex dynamics of real-world traffic systems, ultimately advancing the development of robust autonomous driving solutions.

References

[1] Jason Kong, Mark Pfeiffer, Georg Schildbach, and Francesco Borrelli. Kinematic and dynamic vehicle models for autonomous driving control design. In *2015 IEEE intelligent vehicles symposium (IV)*, pages 1094–1099. IEEE, 2015.

[2] Joar Skalse, Lewis Hammond, Charlie Griffin, and Alessandro Abate. Lexicographic multi-objective reinforcement learning. *arXiv preprint arXiv:2212.13769*, 2022.