CentraleSupélec

Report

Mention IA

RL class Project

Reinforcement Learning for Autonomous Driving

Paul Massey

Students:
Paul Massey
paul.massey@student-cs.fr
Samuel Sithakoul
samuel.sithakoul@student-cs.fr

Supervisors:
Haji Hediji

April 24, 2025

1 Introduction

Reinforcement Learning (RL) has emerged as a powerful framework for developing autonomous agents capable of making complex decisions in dynamic environments. One particularly relevant application is autonomous driving, where agents must learn to navigate safely and efficiently through a variety of road scenarios. In this report, we investigate the training and evaluation of RL-based car agents using the highway-env simulation framework, which provides diverse and customizable driving environments.

Our study focuses on three distinct environments within highway-env: Highway, Roundabout, and Racetrack. Each presents unique challenges that test different aspects of the agent's learning and decision-making capabilities. By training agents in these environments, we aim to assess how well RL methods generalize across different driving tasks and road structures. We explore the performance of various algorithms, analyze their learned behaviors, and highlight the strengths and limitations of current approaches in handling diverse driving scenarios.

As the work was split in our duo with Samuel Sithakoul, this report covers the Task 2 with a DDPG algorithm, the Task 3 with a DQN, and the extra experiment of Task 4. On Samuel side, he describes the Task 1 (DQN), the Task 2 (SAC) and the Task 3 (PPO). As we work together, the description of the environment in the Task 2 and 3 are common for both of us.

Our code is available publicly at this repository `https://github.com/Rubiksman78/BatmoobileRL`.

2 Task 1: Highway

This task consists in training a DQN agent from stratch on the highway environment. This part is handled by my teammate Samuel Sithakoul. This DQN algorithm will be called *DQN1* later in this project.

3 Task 2: Racetrack

3.1 Task description

Racetrack is a continuous control environment where the agent vehicle has to follow the tracks of a looping circuit. The agent's objective is to avoid collisions with neighbouring vehicles while staying centered on the track.

The configuration used is the default one (see Table 1).

There is also a *action_reward* which penalizes when the agent makes too many corrections leading to unstable driving.

3.2 DDPG

Deep Deterministic Policy Gradient (DDPG) is an off-policy reinforcement learning algorithm designed for environments with continuous action spaces. It combines ideas from Deep Q-Networks (DQN) and Deterministic Policy Gradient (DPG) methods. From DQN, it borrows the use of target networks and experience replay for stabilizing training, while from DPG, it adopts the actor-critic architecture, where a deterministic policy (the actor) is trained using gradients derived from a Q-function (the critic) estimated via the Bellman equation.

It is composed of :

- A neural network that learn the deterministic policy and outputs a specific action given a state (actor).

Parameter	Value
observation.type	OccupancyGrid
observation.features	presence, on_road
observation.grid_size	[[-18, 18], [-18, 18]]
observation.grid_step	[3, 3]
observation.as_image	False
observation.align_to_vehicle_axes	True
action.type	ContinuousAction
action.longitudinal	False
action.lateral	True
simulation_frequency	15 [Hz]
policy_frequency	5 [Hz]
duration	300 [s]
collision_reward	-1
lane_centering_cost	4
action_reward	-0.3
controlled_vehicles	1
other_vehicles	1
screen_width	600 [px]
screen_height	600 [px]
centering_position	[0.5, 0.5]
scaling	7
show_trajectories	False
render_agent	True
offscreen_rendering	False

Table 1: Configuration for Racetrack

- A neural network that estimates the Q-value function, and evaluates how good an action is in a given state (critic).
- A replay buffer to store transitions *(state, action, reward, next state)* in order to break some correlations between consecutive samples and stabilizes training.
- Target networks for the actor and the critic, that are updated using a soft update rule, in order to reduce the variance and the stability of the learning process.
- Some exploration using noise to the action generated by the actor network to have enough exploration of the action space.

The detailed algorithm can be found in the appendice (see Algorithm 1) in the end of this report.

3.3 Training

We trained the algorithm described in the previous section. For simplicity, we are taking a Gaussian noise $\mathcal{N}$. The hyperparameters we use for the training of the DDPG algorithm are shown in the following table

Parameter	Symbol	Value
Discount factor	γ	0.9
Soft update coefficient	τ	0.2
Actor learning rate	α_a	2×10^{-5}
Critic learning rate	α_c	2×10^{-5}
Optimizer LR (Actor)	α'_a	1×10^{-4}
Optimizer LR (Critic)	α'_c	1×10^{-3}
Scheduler step size	s	10000
Scheduler gamma	γ_{sched}	0.95
Number of episodes	$N_{episodes}$	1000
Max steps per episode	T_{max}	1500
Replay buffer batch size	B	128

Table 2: DDPG Hyperparameters

During the training, the reward and the length of each episode was plotted in order to see if the model converges, and those graphs are visible below (Figure 1).

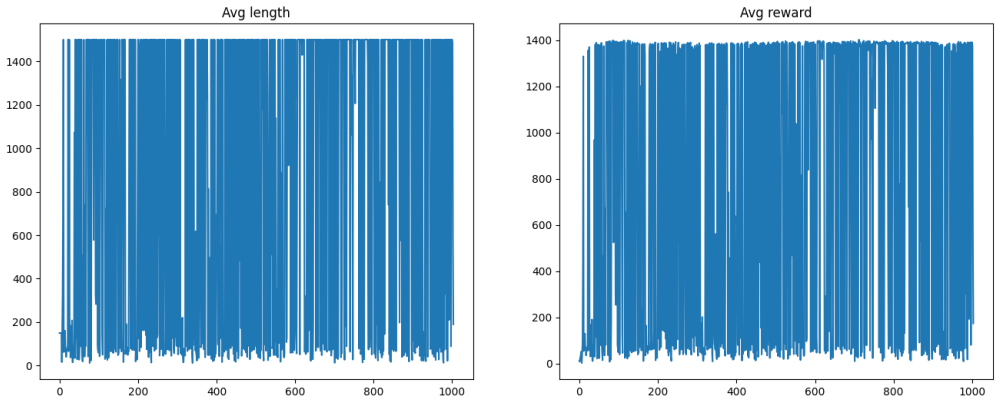

Figure 1: Evolution of total rewards obtained during an episode and episode length during training

3.4 Results

To have a better idea of the result of the model, the agent was simulated 100 times during a testing phase, and the distribution of the reward is shown on Figure 2.

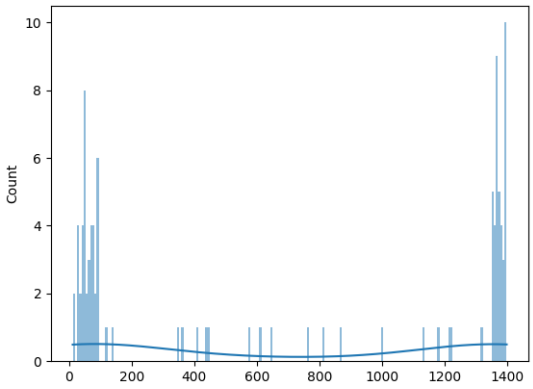

Figure 2: Evolution of total rewards obtained during an episode and episode length during training

The first thing to note for this agent is that it has not converged to something good (especially when compared to the other algorithm used for this task: SAC (see Samuel Sithakoul's report)).

A few specific observations that can be done for the results of the DDPG agent is that, first, the car can follow the track and stay in the center of one of the two available lanes, explaining its high rewards in the reward distribution plot.

A second important observation is that the car driven by the agent has a lot of trouble to overtake. This issue can be even more easily seen in the low rewards, because the car crashed in the first third of the lap not being able to overtake, or even to avoid other cars.

When the car has gone through the first third of the first lap, it seems that it can avoid other cars, sometime by overtaking normally (but at the very last moment), but most of the time by leaving the racetrack and coming back on it driving the wrong way around. This last behavior can be explained looking at the behavior of the other cars: they are changing of lane when there is something in front of them, that can be either another car to overtake or a car driving the wrong direction. In this case, because other cars can avoid the agent, it can survive to the end of the simulation and having a reward similar to the highest reward.

The reason for the non-convergence of this algorithm can be explained by the fact that DDPG

takes more time than SAC to converge. The reason for this is that DDPG uses noise to help explore the action space. On the other hand, this noise can disturb the agent from taking the correct action in a specific situation.

4 Task 3: Roundabout

4.1 Task description

In this case, the agent vehicle is approaching a roundabout with flowing traffic. It follows its planned route automatically but has to handle lane changes and longitudinal control to pass the roundabout as fast as possible while avoiding collisions.

We modified the initial configuration to have a better representation of a real case scenario and based on suggestions from the library author.

Parameter	Config 1 (Kinematics)	Config 2 (GrayscaleObservation)
`observation.type`	Kinematics	GrayscaleObservation
`observation.absolute`	True	-
`observation.observation_shape`	-	(128, 128)
`observation.stack_size`	-	4
`observation.weights`	-	[0.2989, 0.5870, 0.1140]
`observation.scaling`	-	1.75
`observation.features_range`	`x:[-100,100], y:[-100,100]` `vx:[-15,15], vy:[-15,15]`	-
`action.type`	DiscreteMetaAction	DiscreteMetaAction
`action.target_speeds`	[0, 8, 16]	-
`incoming_vehicle_destination`	None	None
`collision_reward`	-1	-4
`high_speed_reward`	0.2	0.2
`right_lane_reward`	0	0
`lane_change_reward`	-0.05	-0.05
`duration`	11 [s]	11 [s]
`simulation_frequency`	-	15 [Hz]
`policy_frequency`	-	1 [Hz]
`screen_width`	600 [px]	600 [px]
`screen_height`	600 [px]	600 [px]
`centering_position`	[0.5, 0.6]	[0.5, 0.6]
`scaling`	-	5.5
`normalize_reward`	True	True

Table 3: Comparison Between Kinematics and GrayscaleObservation Configurations

In our modified configuration, we use image observations in grayscale with stacking of four consecutive frames as done in many classical algorithms with image inputs. This choice was encouraged by issues encountered with the development of RL algorithms in this environment and recommandations from the author in this issue `https://github.com/Farama-Foundation/HighwayEnv/issues/99`. The motivation is that Kinematics or OccupancyGrid contains ordered information about the vehicles which is not prefered with Fully connected Neural Networks which are not invariant by permutations of the inputs. However the combination of image input and CNN might be better as it is not dependant on the order of vehicles and better associated with the notion of a field of view for the agent. We also put more weight to the penalty for collision as the model might tend to prioritize its speed which gives an overall higher total reward across an episode.

4.2 DQN

In addition of the PPO algorithm made by Samuel, this problem was addressed using the Deep Q Learning algorithm as seen in class.

4.3 Training

We trained the DQN algotirhm using this set of parameters (see Table **??**), leading to the reward evolution graph on Figure 3.

Parameter	Symbol	Value
Total timesteps	T_{total}	1×10^5
Batch size	B	64
Discount factor	γ	0.99
ϵ start	ϵ_{start}	1.0
ϵ end	ϵ_{end}	0.02
ϵ decay	τ_ϵ	$0.1T_{\text{total}}$
Target update rate	τ	0.005
Learning rate	α	2×10^{-4}
Warmup steps	t_{warmup}	200
Update interval	f_{target}	50
Replay buffer size	$\lvert\mathcal{B}\rvert$	2×10^5
Hidden dimension	N_{hidden}	64

Table 4: DQN Hyperparameters

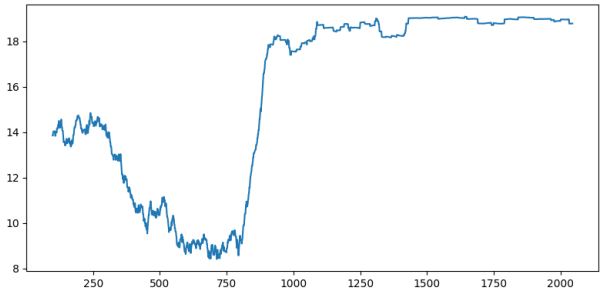

Figure 3: Evolution of the rewards, moving average is taken over 100 steps

4.4 Results

For this algorithm, contrary to the previous one, it converges successfully, as the distribution reward graph show (see Figure 4). When looking in detail, the car can reach the end of the simulation, but in a very specific way. Even though there is a *high_speed_reward* in the config, the car stops before entering the roundabout until all other vehicles have exited. Once the roundabout is clear, the agent proceeds, but sometimes brakes again later, likely to avoid a vehicle in front of it.

In this case, the agent appears to be engaging in a form of *reward hacking*, prioritizing survival over maximizing all rewards. Since the negative reward for a collision is greater (in absolute value) than the reward for high speed (-4 compared to 0.2, it is understandable that the car prioritizes avoiding collisions. However, the strategy it uses is surprising. In a way, the car follows driving guidelines more exactly than the other vehicles, which may even crash into each other.

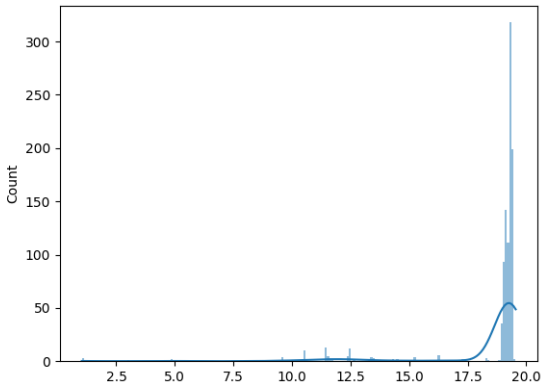

Figure 4: DQN reward distribution on 1000 simulations

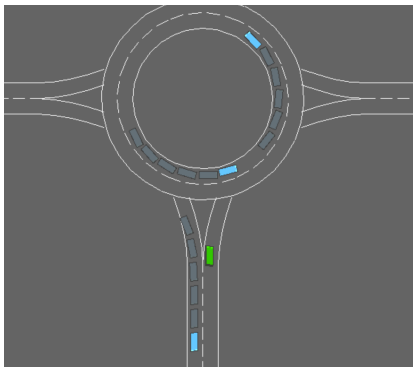

Figure 5: The car stop first before entering the roundabout not to crash in any other cars

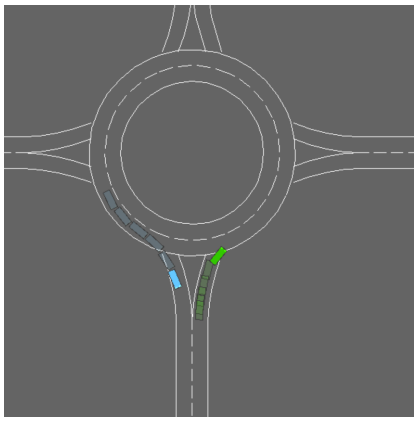

Figure 6: Then, the car re-accelerate to engage into the roundabout when every other cars have left it

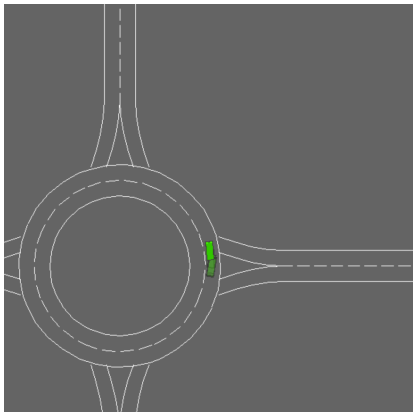

Figure 7: Even if there is nobody around it, the car tends to re-brake later in the simulation

5 Task 4: Extra experiment

For Task 4, the chosen experiment was to test whether an agent trained for a specific task could perform well on a different task. The agent trained on the roundabout environment from Task 3 (called *DQN3* in the following parts) was reused in the highway environment, with performance evaluated using the reward function from Task 1 (called *DQN1*). For more details, about the configuration of the highway environment, check on Samuel's report.

The distribution of rewards of *DQN3* is represented on the Figure 8, whereas the distribution of the DQN agent trained during Task 1 (*DQN1*) is represented on Figure 9 as a remind (imported from Samuel's report).

The first observation is that the *DQN3* is not as good as *DQN1*, what is logical because the first one was not trained on the same task. When looking in details to the rewards of *DQN3*, there are a large part in the lowest part, because of early crashes. The car goes quickly on the right lane and keeps a high speed compared to the other vehicles, and after a few iterations, if it has not crashed yet, it breaks and let the other vehicles overtaking before accelerating progressively (and still remaning on the right lane).

The behavior of the car is similar to that in the roundabout task, which is expected since it is the same agent. The agent's behavior, learned to navigate the roundabout, helps it make some progress on the highway. However, it struggles at the beginning because it doesn't know how to act in this

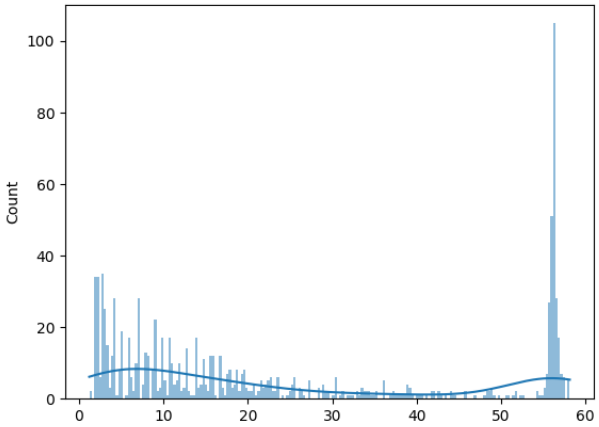

Figure 8: *DQN3* reward distribution on 1000 simulations for the highway env

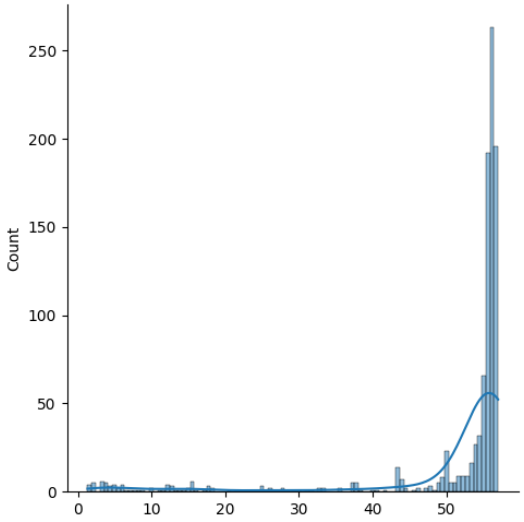

Figure 9: *DQN1* reward distribution on 1000 simulations

new context. This is similar to how the car initially waits before entering the roundabout: it doesn't engage until other vehicles are on the roundabout. On the highway, there is constant interaction with other cars, and without training in this environment, the agent simply repeats familiar behaviors rather than adapting to the new situation.

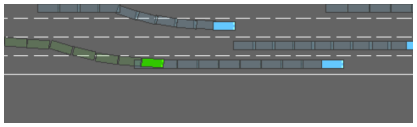

Figure 10: The car is going as soon as possible on the right lane

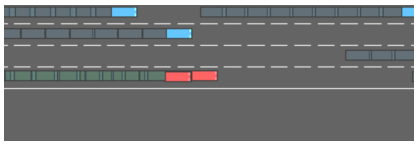

Figure 11: The car tends to crash on the back of cars on the right lane

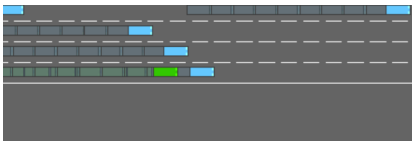

Figure 12: The car is not anticipating necessary overtake, or does not break before a crash

6 Conclusion

In this project, we investigated reinforcement learning approaches across multiple autonomous driving environments of varying complexity. While simpler scenarios allowed agents to converge efficiently, more dynamic settings, such as racetrack, posed significant challenges due to increased unpredictability. The experiments also highlighted the sensitivity of agent behavior to reward design, occasionally leading to reward exploitation rather than goal-oriented learning. Future work can focus on algorithm that respects the reward, such as incorporating vision-based inputs, or training more complex algorithm to tackle the issue encountered in this report.

7 Appendices

7.1 Task 2

Here is the detailed DDPG algorithm mentioned in the Task 2 for the racetrack:

Algorithm 1 DDPG Algorithm

Initialize: Actor network $\mu(s|\theta^\mu)$ and critic network $Q(s,a|\theta^Q)$ with random parameters θ^μ, θ^Q
Initialize: Target networks μ' and Q' with weights $\theta^{\mu'} \leftarrow \theta^\mu$, $\theta^{Q'} \leftarrow \theta^Q$
Initialize: Replay buffer $\mathcal{D}$

for each episode **do**
    Initialize a random process $\mathcal{N}$ for action exploration
    Receive initial state s_0
    for each time step t **do**
        Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$
        Execute action a_t, observe reward r_t and next state s_{t+1}
        Store transition (s_t, a_t, r_t, s_{t+1}) in replay buffer $\mathcal{D}$
        Sample a minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from $\mathcal{D}$
        Compute target: $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
        Update critic by minimizing loss: $L = \frac{1}{N}\sum_i (Q(s_i, a_i|\theta^Q) - y_i)^2$
        Update actor using the policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{a=\mu(s)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)$$

        Soft update target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

    end for
end for