# Reinforcement Learning Project: Autonomous Driving


| | | | |
|---|---|---|---|
| Anton Kutsenko | *ID: 20230765* | Mateus Amaral | *ID: 20230595* |
| Maximillian Laechelin | *ID: 20230979* | Diogo Parreira | *ID: 20230758* |



**Abstract**

Autonomous driving lately has appeared as a major area of research and development, which promises to revolutionize safety, efficiency, and convenience in transportation. Reinforcement learning (RL) offers a powerful way to optimize decision-making in vehicles, enabling them to learn complex behaviors through interaction with their environment. This project focuses on developing RL agents to navigate the Highway environment, a simulated setting that captures the dynamics of real-world highways while providing a simplified and controlled testing ground. We explore various RL algorithms, including Q-learning (table and deep) and A2C, and evaluate their performance.


## 1 Introduction

The highway environment, part of Farama's foundation "highway-env" project, presents a challenge for an agent who is required to maximize his speed (within certain bounds) while at the same time avoiding crash risks, and driving as close to the right lane as possible. The environment allows several observation and action types, giving a broad experimentation space.

In this project, we implement four different solutions, where each leverages a different combination of observation and action types, with an appropriate choice of algorithm such as the ones used in [1]. These approaches aim to explore and compare the effectiveness of various RL algorithms. The solutions developed are:

- Q-Learning with Q-table for modified Kinematics with Discrete Meta-actions
- DQN with Discrete Meta-actions and custom functions (CNN and NN) for Time to Collision Observation
- DQN with Discrete Action Type and Kinematic Observations
- A2C with Continuous Actions and Kinematics, allowing for more detailed control over the vehicle's behaviour.

These solutions are evaluated based on their performance on the given default reward system, with the main focus being on maximizing the rewards by balancing speed and crash risk. Additionally, we explore and expand the reward system during training to enhance our agent's decision-making.

The report details the methodology and implementation of each solution as well as challenges that occurred in the process. We evaluated each agent on the main metrics - the cumulative reward during the evaluation phase -, and created some plots to gain insights into their decision-making, such as action distribution and velocity for each time step.

# 2 Methodology and Implementation

For the first two solutions, our main goal was to code everything by hand, so as to gain a deeper understanding of how the models worked, as well as to implement more customizable and out-of-the-box solutions. The latter two solutions were done with the help of the stable-baselines library.

## 2.1 Discrete meta actions with modified kinematics and Q-learning

For the first solution, we faced a multitude of problems. The first was the curse of dimensionality, which posed a significant obstacle, as the high-dimensional state space made it difficult for the agent to learn in human time. This was followed by the reckless driving behaviour of the agent, including double-steering and illegal actions. Finally, and common to all the solutions we experimented with, the speed/crash risk ratio, and the reward to compensate it. This was also accompanied by other hyper-parameter tuning challenges, which we will dissect later on.

### 2.1.1 N-closest neighbours

Transforming a continuous observation space like the one from kinematics into a discrete one, with reasonable dimensionality is definitely a hard task. The first thing we tried was to implement a n-closest neighbours (n-CN) solution, with binned values of x and y coordinates (not including speed). This posed some problems:

1. To ensure a solvable problem, we could have a maximum of 10,000 states (50,000 distinct Q-values). Assuming that only 50% of these states would be visited (some states are impossible) and that we would need to pass through each state 20 times for minimal training, with the environment generating 2 observations per second the training time would be approximately 14 hours. Furthermore, to achieve a sufficient observation space, we would need to consider the 4 closest neighbours while maintaining a bin size of 5 for the x and y coordinates (4 for positive and negative values and 1 for 0). This would result in a space of approximately 400,000 different observations. In general, for the n-CN solution, the observation space size is given by the following expression:

$$\text{Size}_{\text{ncn}} = (x_{\text{bin size}} \cdot y_{\text{bin size}})^{n_{\text{neighbours}}} \tag{1}$$

2. For the agent to be able to drive and to make good predictions, we would need to include somehow the speed of the other cars in relationship to our car, which would further increase the observation space size.

### 2.1.2 Euclidean distance

After taking notice of this, we moved to another approach, which was the binned Euclidean distance to the closest car in the lane ahead, behind, on the left and on the right. For the left and right lanes, in case the car was behind, we would consider the negative Euclidean distance. For this solution, we would have an observation space size of about 225 (with 3 and 5 bins for x,y), given by the following equation:

$$\text{Size}_{\text{euc}} = (x_{\text{bin size}} \cdot y_{\text{bin size}})^2 \tag{2}$$

### 2.1.3 Danger Space

Although the size of the former approach is already manageable, there are still complications, like not accounting for the speed at which the other cars are driving, in comparison to the agent's velocity; not considering cars that are two lanes on the right (or left), which could turn left while the agent is turning right, posing a risk of crash that was not previously accounted for in the observation; disregarding illegal actions and double-steering.

To address all this, we devised the third observation space, which we called the danger-space. It consists of a tuple of 6 entries, the first four of which represent the danger in the front, back, left and right, followed by the turn possibility, which can be 0 if it can freely turn and -1 or 1 if it can't turn left or right. The final entry is 1 if the car has turned in the last n-past actions, or 0 if it hasn't. This helps prevent double steering.

The last two entries needn't be explained, as they are simple, but for the first four, that is not the case. To calculate them, we used the formulae (3) and (4), for the front/back and left/right:

$$\text{front/back} = \begin{cases} 1, & \text{if } \min_{a \in \mathcal{F}/\mathcal{B}} \left( |x_a| + sign(x_a) \cdot v_{x_a} \cdot \beta_{v_x} \right) < D_x, \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

$$\text{left/right} = \begin{cases} 1, & \text{if } \min_{a \in \mathcal{L}/\mathcal{R}} \left( |x_a| + \mathbb{1}_{\text{behind}} \cdot v_{x_a} \cdot \beta_{v_x} + \mathbb{1}_{\text{2lane}} \cdot (5 + sign(y_a) \cdot v_{y_a} \cdot \beta_{v_y}) \right) < D_y, \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

Where $a$ is an automobile on the road, $x_a$ and $y_a$ are the relative coordinates to the agent, meaning they represent the distances for each dimension, $v_{x_a}$ and $v_{y_a}$ the difference in velocity to our car, $\beta_{v_x}$ and $\beta_{v_y}$ are tunable parameters, which dictate the importance of x and y speed in the formulas, and $D_x$ and $D_y$ are the adjusted distances' thresholds. Finally, there is another parameter, $L$, which is the lane threshold. If car $a$ has $y_a = 5.5$, then it is either in the next lane from mine (from $4 - L$ till $4 + L$), or in the next one. This parameter is used to get the sets $\mathcal{F}$, $\mathcal{B}$, $\mathcal{L}$, $\mathcal{R}$, which contain the cars for each direction, and for the $\mathbb{1}_{\text{2lane}}$ indicator function. The other function $\mathbb{1}_{\text{behind}}$ uses a value of $-3$ on the $x_a$ to check if the car is behind (after simulation, we found $-3$ to be the best value).

These formulas work well because they tackle all our problems, while simultaneously reducing our observation space to a mere 96 possible combinations, of which only 31 are likely. For all the adjusted distances, we include the velocity difference to account for the breaking or reaction time. The left and right functions are a little harder, because if a car is in the back with a negative velocity, then it is of no harm, whereas if it's in front of the agent it could be dangerous. That is why we use an indicator function for behind ($\mathbb{1}_{\text{behind}}$). If the car is in the second lane on the left, his $y_a$ is negative, which means $sign(y_a) = -1$, thus accounting for when the lane changing velocity ($v_{y_a}$) is positive, i.e., the car is turning in our direction.

### 2.1.4 Training and evaluation

In order to balance exploration-exploitation as best as possible, we decided to implement a schedule for the epsilon values, so that in the beginning the agent would mostly explore, and towards the end, after the epsilon had already decayed, the agent would exploit what it already knew.

To understand how the training was going, we implemented a solution which consisted of plotting 6 charts with the historical and current information of the training process. This included the historical mean reward, speed and steps, and the action distribution per episode evaluation,

along with the current action distribution on the Q table and the percentage of the Q table explored. We also hard-coded an optimum Q table, and from that calculated the Q measure, which represents the percentage of correct actions predicted for a given state.

We did a random search to find the best hyper-parameters, and then evaluated the models we got to get a grasp of what influences each model's learning ability. We also played a little with other algorithms like SARSA and Monte Carlo control. Finally, the best model was chosen and evaluated to get the mean speed, reward and steps per epoch, so that we would then be able to compare it with our other models.

## 2.2 Discrete meta actions with time to collision and DQN

The objective of this solution was to implement and customize the models used for the prediction of the Q-values given an observation of the type time to collision. We used some CNNs and NNs, but we soon realized the CNNs could have a problem: using this observation type, we would get 3 matrices for the 3 different speeds the car could be driving at, while the car doesn't know its speed. This seemed inefficient, as we increased the input size 3-fold, and harder for the agent to learn. We could either add the speed to the CNN or select one of the 3 matrices for every state using the speed index of the car. The issue was that if we were to keep only 1 matrix, we wouldn't be able to use a CNN.

### 2.2.1 Neural Networks

The architecture chosen for our neural networks was two hidden layers, with dimensions (32,16), for the first neural network, and (128,64) for a more complex one. The CNNs used had only one convolution layer, as the dimensions of the observation space are 3x3xN, and the minimum filter size we considered was of 2x2.

### 2.2.2 Slow training

Q learning needs a lot of samples to learn from, and one gym environment can only generate so much data per second, which in our case was the bottleneck worsening our performance. To fix this, we had to implement a vectorial environment, which allows us to run multiple environments at the same time, thus generating the necessary data for the model to learn. We chose to use from 6 to 10 different environments at the same time, as increasing beyond 10 yields decreases performance.

### 2.2.3 Hyper-parameters

The most important hyper-parameters specific to this solution are the batch size, the frequency to train the main model and update the target model's weights, the horizon for the time-to-collision observation type, which gets stretched if we increase the policy frequency, the replay memory size and the minimum replay memory.

These parameters really matter for the model's training performance, as fitting the main model at every step will be computationally expensive and inefficient. Conversely, updating it too rarely is not effective, as we will through away a lot of our data that took time to be generated. A small target model update frequency will generate instability in the training, while a larger one will slower it.

### 2.2.4 Evaluation

For this approach, finding a way to keep track of the training process proved crucial, as these models can take hours to train. To ease that, every n iterations the model gets evaluated 10 times and 4 plots are displayed: all the past and current mean rewards and steps per episode, the action distribution during the evaluation and the historical cross-entropy of the actions, which was implemented in order to see if the agent was only doing one action. The best model was then evaluated to be compared with the other solutions.

## 2.3 Discrete action space with Kinematics and DQN

In this solution, we employed the kinematics observation type and discrete action space to train our RL agent using a Deep Q-network (DQN). The kinematics observation type included features such as the vehicle's position (x, y) and velocity ($v_x$, $v_y$), providing essential information for decision-making. The discrete action space - a discretized version of continuous actions in the highway environment, allowed us to train DQN, as it requires a discrete action space.

Deep Q-Networks extend the Q-Learning algorithm by approximating the Q-value function (table) with a neural network. This approach is advantageous in environments with large state spaces (6.25x bigger than with discrete meta actions), where tabular Q-Learning becomes impractical. In our implementation, we used a Multilayer Perceptron as the policy network to approximate the Q-values for each discrete action.

### 2.3.1 Discretization of the action space

We experimented with different settings for the steering wheel and acceleration, as well as with action space size. As a final solution, we discretized the action space by dividing the steering range [-$\pi/4$, $\pi/4$] into 5 bins and the acceleration range [-5, 5] into 5 bins, resulting in 25 different action combinations. This discretization provided a balance between granularity and computational feasibility.

Unlike discrete meta actions, discrete actions require the agent to learn combinations of actions to perform tasks such as lane changes, adding complexity to the learning process.

Initially, the lane change reward was set to zero which led the agent to maximize speed by driving in a straight line. To encourage overtaking and more dynamic driving behaviour, we experimented with positive lane change rewards, which effectively incentivized the agent to perform lane changes as implicitly seen in the action distribution of our agent in the notebook.

We encountered an issue with the speed range initially set to [-30, 30], which allowed negative speeds and caused the agent to drive in reverse. To address this, we implemented a wrapper class to penalize negative velocities. Ultimately, we simplified this solution by restricting the speed range to [1, 30], preventing reverse driving.

### 2.3.2 Configuration and evaluation

We configured the highway environment with specific parameters, including 4 lanes, 50 vehicles and a maximum episode duration of 40 seconds. We experimented in changing the reward system, by increasing the penalty for collisions, which led to lower rewards during evaluation. This led us to keep the initial values for collision rewards and high speed. We also tried several custom reward systems as we needed to account for different corner cases, including off-road driving and pointless slowdowns. The final reward structure included penalties for collision and rewards for maintaining high speeds and lane changes (collision_reward=-1, high_speed_reward=0.4, lane_change_reward=0.2).

After training, we evaluated the agent over 5 episodes to assess its performance on the default configuration given by our teacher. Key metrics included cumulative rewards, average speed, and episode length. Various visualizations were generated to analyze the agent's behaviour, such as speed over time and action distribution.

### 2.4 Continuous action space with kinematics and A2C

In this solution, we used a continuous action space to train our agent using the advantage actor-critic algorithm to get a comparison between two algorithms using a similar setup. The kinematics observation space was chosen as before. Although a continuous action space brings about difficulties, it also gives a more fine-grained control, making it suitable for an A2C algorithm, which is designed to handle continuous action spaces effectively.

This on-policy algorithm combines the benefits of value-based and policy-based methods. It consists of two components: the actor, which decides the actions to take, and the critic, which evaluates the actions taken by the car. The advantage function helps the learning process by providing more accurate feedback on the actions taken, which makes the learning process more stable and efficient.

The continuous action space was set up with the same ranges for steering and acceleration as before, while the configuration only differed in that there wasn't a need for lane change rewards. After training the model, it was evaluated on the same metrics as seen before.

## 3 Evaluation results

After training all the solutions and selecting the best model for each one of them, we found that the q-learning with the modified kinematics observation space was the one that performed best, as can be seen in table 2, which summarizes the mean rewards achieved by each algorithm along with their respective action and observation spaces.

For the discrete modified kinematics, the SARSA algorithm was also applied, but it didn't prove itself, as it under-performed consistently. This approach was discontinued later on, as the Q learning provided much better results.

In terms of time to train this approach was also king, as it got to good results in about an hour, with the other solutions, especially the second one, taking up to 6 hours of training to achieve moderate convergence, even though the sampling was greatly sped-up with the use of vectorial environments.

We can see how well our best model is trained in figure 6 by looking at the evaluation plots that are run at every n episodes. Diving deeper into this solution, we obtained the following results from the hyperparameter tunning (check table 1), where we can see that ... influences

For the second solution, the CNNs didn't train well at all, taking up to 10 hours to achieve very poor performance. The neural networks did better in that they were faster, but the results aren't comparable to those of the first solution. Our best model of this solution was obtained with a neural network of architecture (128,64), with a maximum episode duration of 20 steps, a batch size of 32, frequencies to train and update models, respectively, 4 and 100, alpha of 0.05 and a replay memory of 50,000.

The third and fourth solutions didn't fully go according to plan. After multiple modifications of the reward system, we could finally force our cars to stay inside their lanes for a little while but did not achieve consistent performance even with strong penalizations. The agent couldn't learn the concept of lanes, and hence he would drive oftentimes in the middle, resulting in very early crashes, and poor overall performance.

# 4 Conclusion

The results obtained were not the best, with only the first solution proving to be working well, while the second one had the problem of the agent not turning left, even though a reward for changing lanes was provided, which made the agent miss out on potential overtakes that would yield a better cumulative reward for the agent.

We learned mostly how to program reinforcement learning models from scratch and also how to train them using a stable baselines library.

# 5 Challenges and future directions

As we already see autonomous driving using reinforcement learning is a complicated task in terms of realisation and computation. The main challenges we faced were the huge space of possible states (for discrete solutions) which had to be reduced wisely and the exhaustive search for optimal combinations of rewards and the model's hyper-parameters, as there are many of them and because each model takes a lot of time to be trained, further hindering progress. Future directions include deeper exploration of continuous actions and more advanced RL models. The reward system can also be improved by adding custom penalties and finding a trade-off between different parts of the reward system, in order to, for example, achieve an even distribution of the different actions (in the case of discrete actions).

# References

[1] El Sallab, A., Abdou, M., Perot, E., Yogamani, S. (2017). Deep Reinforcement Learning framework for Autonomous Driving. arXiv preprint arXiv:1704.02532. Retrieved from https://arxiv.org/pdf/1704.02532.

# 6 Annex

| Model | Mean Reward | Mean Steps | Mean Reward/Step | Mean Speed | Std Reward | Std Steps | $\alpha$ | $\gamma$ | past_action_len | $\beta_{v_x}$ | $\beta_{v_y}$ | to_right_skewness |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 20240612-1606 | 38.62 | 101.00 | 0.38 | 27.16 | 21.96 | 42.49 | 0.15 | 0.90 | 2 | 1.20 | 0.60 | 8 |
| 20240611-1522 | 33.15 | 83.50 | 0.40 | 27.68 | 21.87 | 51.85 | 0.10 | 0.99 | 3 | 1.25 | 0.50 | 6 |
| 20240611-2133 | 23.41 | 86.67 | 0.27 | 27.13 | 16.24 | 47.14 | 0.05 | 0.99 | 3 | 1.25 | 0.50 | 8 |
| 20240612-0056 | 21.61 | 54.17 | 0.40 | 29.15 | 21.10 | 49.56 | 0.10 | 0.99 | 3 | 1.15 | 0.75 | 8 |
| 20240611-2248 | 15.91 | 35.50 | 0.45 | 28.33 | 20.13 | 40.14 | 0.05 | 0.95 | 3 | 1.30 | 0.50 | 8 |
| 20240612-0149 | 11.52 | 86.17 | 0.13 | 22.59 | 5.20 | 47.85 | 0.10 | 0.98 | 3 | 1.05 | 0.80 | 8 |
| 20240612-0430 | 10.02 | 87.33 | 0.11 | 21.98 | 7.05 | 46.90 | 0.13 | 0.98 | 2 | 1.05 | 0.50 | 8 |
| 20240612-1434 | 7.57 | 56.00 | 0.14 | 24.22 | 4.35 | 47.34 | 0.18 | 0.85 | 3 | 1.15 | 0.50 | 8 |

Table 1: Tabular Q-learning results

| **Algorithm/ActionSpace/ObsSpace** | **Mean Reward** |
|---|---|
| Q-Table/DiscreteMeta/Kinematics | 38.62 +- 21.96 |
| DQN/DiscreteMeta/TimeToCollision | 17.23 +- 11.05 |
| DQN/DiscreteActions/Kinematics | 26.01 +/- 12.67 |
| A2C/ContinuousActions/Kinematics | 44.99 +/- 16.26 |

Table 2: Mean Rewards for the Algorithms, evaluated on default environment for 5 episodes


Figure 1: Results for the best tabular Q-learning model

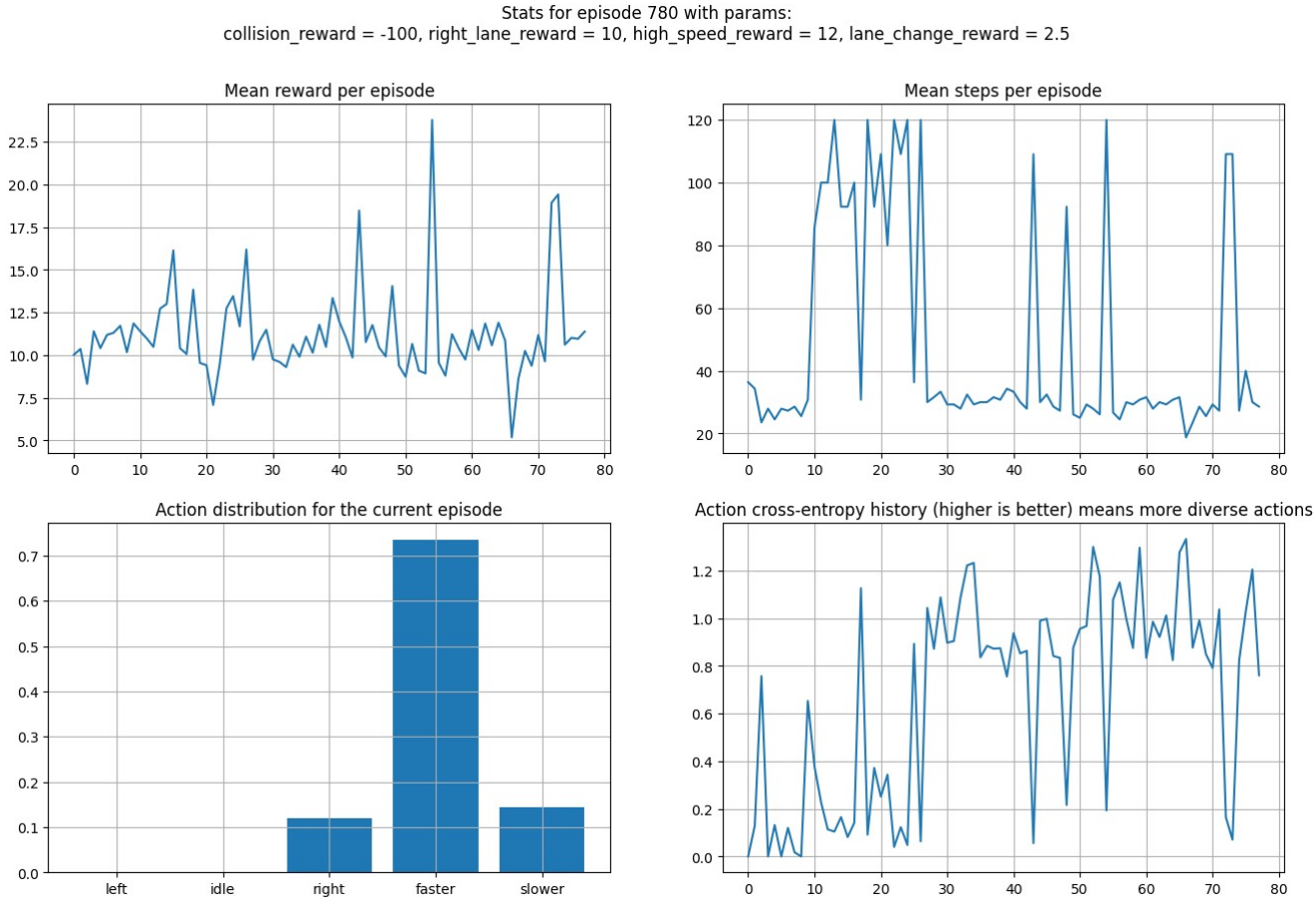


Figure 2: Best DQN model learning history