 https://doi.org/10.3311/PPtr.24729

Periodica Polytechnica Transportation Engineering, 54(3), pp. 262–269, 2026

Multi-agent Reinforcement Learning-based Control in a Yield-sign Controlled Intersection

Dániel Tamás Gujgiczer[1], Ádám Szabó[1,2*]

[1] Department of Control for Transportation and Vehicle Systems, Faculty of Transportation Engineering and Vehicle Engineering, Budapest University of Technology and Economics, Műegyetem rkp. 3., H-1111 Budapest, Hungary
[2] Systems and Control Laboratory, HUN-REN Institute for Computer Science and Control (SZTAKI), Kende utca 13–17., 1111 Budapest, Hungary
* Corresponding author, e-mail: szabo.adam@kjk.bme.hu

Received: 24 December 2023, Accepted: 23 September 2025, Published online: 25 February 2026

Abstract

Both single- and multi-agent reinforcement learning have been widely used to solve different problems of autonomous driving. Single-agent methods have already outperformed traditional rule-based algorithms in several areas, but recent research shows that multi-agent algorithms have an even greater potential. Encouraging cooperative behavior and communication between the agents leads to their altruistic behavior, which results in safer and more reliable driving strategies while applying techniques such as parameter sharing and curriculum learning helps to deal with the increased complexity of the problem. This paper aims to compare the performance and reliability of single- and multi-agent reinforcement learning algorithms through the example of a yield-sign controlled intersection.

Keywords

autonomous vehicles, multi-agent reinforcement learning, proximal policy optimization, reinforcement learning

1 Introduction

Research related to Deep Reinforcement Learning (DRL) has prospered since the enormous success of Google Deepmind's AlphaGo Zero (Mnih et al., 2013; Mnih et al., 2015). Since then, it has been successfully deployed in various applications, such as video games (Berner et al., 2019) and board games (Silver et al., 2017). Due to DRL's proficiency in solving sequential decision-making problems, it also became an area of interest in research related to road transportation, such as autonomous driving, route optimization, and traffic signal control.

Some works attempt to provide an end-to-end framework to solve the tasks of autonomous driving (Ly and Akhloufi, 2021). On the other hand, most researchers propose the applications of modular systems and focus only on sub-tasks of the problem, such as environmental sensing (Farooq et al., 2023) especially with a pedestrian. To avoid collision with pedestrians, the vehicle requires the ability to communicate with a pedestrian to understand their actions. The most challenging task in research on computer vision is to detect pedestrian activities, especially at nighttime. The Advanced Driver-Assistance Systems (ADAS) and local trajectory planning (Moghadam et al., 2021).

Reinforcement learning algorithms performed outstandingly in tasks related to the higher-level decision-making problems of autonomous vehicles. For instance, in (Wang and Chan, 2018), a reinforcement learning algorithm is proposed to complete the merging process on a highway successfully. In (Wolf et al., 2018), a deep reinforcement learning algorithm is proposed to learn maneuver decision making in a highway environment. The algorithm uses a compact semantic state representation, which ensures consistent modeling of the environment and a behavior adaptation function. The research presented in (Leurent and Mercat, 2019) proposes an attention-based architecture to handle various nearby vehicles in an intersection. Then, it is compared against fully connected and convolutional neural networks using DQN.

Single-agent algorithms work exceptionally well in scenarios where the aim is to maximize their rewards. In these applications, other drivers are mostly

Cite this article as: Gujgiczer, D. T., Szabó, Á. (2026) "Multi-agent Reinforcement Learning-based Control in a Yield-sign Controlled Intersection", Periodica Polytechnica Transportation Engineering, 54(3), pp. 262–269. https://doi.org/10.3311/PPtr.24729

modeled using algorithms such as the Intelligent Driver Model (IDM) (Treiber et al., 2000) and Minimizing Overall Braking Induced by Lane change (MOBIL) (Kesting et al., 2007) models.

While, according to the applied rewarding strategy, single-agent algorithms could learn both selfish and selfless behavior (Li and Chen, 2024; Schwarting et al., 2019), multi-agent reinforcement learning has several advantages in controlling autonomous vehicles, such as better scalability in mixed traffic and the possibility of communication and collaboration between agents.

Kaushik et al., (2018) proposes a novel MARL architecture utilizing parameter sharing. It can learn multiple driving behaviors simultaneously, while the results indicate a more generic behavior and faster convergence. A scalable MARL framework is presented in (Chen et al., 2023) with an action masking scheme. It can be efficiently used in dynamic traffic scenarios, where the communication topology can be time-varying.

Multi-agent algorithms have already been successfully deployed to navigate vehicles through busy intersections. In (Kalantari et al., 2016), a distributed cooperative intelligence-based system is proposed, which routes vehicles to their destination safely and on time. A delay-aware algorithm is presented in (Chen et al., 2020), where the framework of delay-aware Markov games is formalized and solved using centralized training and decentralized execution.

This paper compares single- and multi-agent reinforcement learning algorithms in a yield-sign controlled intersection. Two different reinforcement learning algorithms (namely Deep Q-Network and Proximal Policy Optimization) and rewarding strategies are utilized along with parameter-sharing and curriculum learning.

The paper is organized as follows: Section 2 presents the algorithms applied and the hyperparameters used for training. Section 3 introduces the simulation environment and the modifications necessary for multi-agent reinforcement learning. Section 4 describes the environment configurations and the hyperparameters of the agents for the training scenarios. Section 5 deals with the simulation results and the analysis of the algorithms' performance. Section 6 shows some concluding remarks.

2 Methodology

Reinforcement learning is a rapidly developing branch of machine learning, which learns from a continuous interaction with its environment through trial and error. Reinforcement learning problems are formulated mathematically as a Markov Decision Process (MDP) described by a $\{S, A, P, R\}$ tuple.

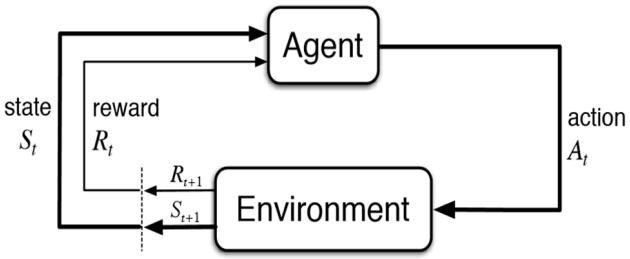

Fig. 1 Reinforcement learning training loop

The training loop of reinforcement learning algorithms is shown in Fig. 1. At each i discrete time step, the agent receives the $s_i \in S$ state of the environment, then it chooses an $a_i \in A$ action based on its policy π_i. The action executed by the agent induces a $P(S_i, a_i|S_{i+1})$ state transition in the environment, and the agent receives a $r_i \in R$ reward based on the quality of the selected action.

During training, the agent aims to develop a policy which maximizes its cumulative reward:

$$G = \sum_{t=1}^{T} \gamma^t r_t \ . \tag{1}$$

2.1 Deep Q-network

Q-learning, which was first defined by (Watkins, 1989), is a value-based, model-free, off-policy variant of Temporal Difference (TD) methods (Sutton and Barto, 2018). It directly approximates the optimal action-value function, q_*, with Eq. (2):

$$\begin{aligned} &Q(S_t, A_t) \leftarrow Q(S_t, A_t) \\ &+\alpha\left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right], \end{aligned} \tag{2}$$

where Q is the approximated function and $R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$ is the target for the updates of Q, which comes from the Bellman optimality equation. The earliest Q-learning methods used a tabular approach to store Q values for each state-action pair. However, the increase in observation and action space sizes can lead to a serious increase in computational costs. This opened the way to approximate the Q function with neural networks. In this case, Q values become dependent of the parameters of the network stored in θ_t, which modifies the target of the updates as Eq. (3):

$$Q_t^* = R_{t+1} + \gamma Q\left(S_{t+1}, \arg\max_a Q(S_{t+1}, a; \theta_t); \theta_t\right). \tag{3}$$

This nonlinear approximation of the Q function is known to lead to instability or even divergence, caused by the correlation between observations, the relation between the actual and estimated Q values, and even small updates of the Q-function can induce significant changes in the policy. These problems have been tackled by (Mnih et al., 2015)

in their algorithm called Deep Q-Network (DQN). Their implementation used an experience replay memory, where the experiences of the agent are stored, and random samples are taken in batches to update Q values, reducing correlations in the observation sequence. The other two issues have been dealt with through using a target network, whose θ_t^- parameters are updated periodically after a certain amount of timesteps to the actual, online Q network's θ_t parameters. The optimal Q value can then be determined as Eq. (4):

$$Q_t^{*DQN} = R_{t+1} + \gamma Q\left(S_{t+1}, \arg\max_a Q\left(S_{t+1}, a; \theta_t^-\right); \theta_t^-\right). \tag{4}$$

2.2 Proximal policy optimization

Policy gradient methods obtain an estimator of the policy gradient and incorporate it into a stochastic gradient ascent algorithm. The most widely used gradient estimator is written as Eq. (5):

$$\hat{g} = \hat{\mathbb{E}}_t\left[\nabla_\theta \log \pi_\theta\left(a_t \middle| s_t\right)\hat{A}_t\right]. \tag{5}$$

Implementations using automatic differentiation software construct an objective function whose gradient is the policy gradient estimator, and then $\hat{g}$ is obtained by differentiating the following objective function:

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t\left[\log \pi_\theta\left(a_t \middle| s_t\right)\hat{A}_t\right]. \tag{6}$$

Performing multiple optimization steps on L^{PG} using the same trajectory would be appealing; it often leads to excessively large policy updates, thus destabilizing the learning process.

On the other hand, Trust Region Policy Optimization (TRPO) algorithms aim to maximize an objective function subject to a constraint on the size of the policy update:

$$\max_\theta \hat{\mathbb{E}}_t\left[\frac{\pi_\theta\left(a_t \middle| s_t\right)}{\pi_{\theta_{\text{old}}}\left(a_t \middle| s_t\right)}\hat{A}_t\right], \tag{7}$$

$$\text{subject to } \hat{\mathbb{E}}_t\left[KL\left[\pi_{\theta_{\text{old}}}\left(\cdot \middle| S_t\right), \pi_\theta\left(\cdot \middle| S_t\right)\right]\right] \le \delta. \tag{8}$$

While the theory justifying TRPO suggests using a penalty instead of a constraint, TRPO implementations use a hard constraint because it is difficult to generalize the penalty well for different problems or even within a single issue, where its characteristics change over the learning process.

The Proximal Policy Optimization (PPO) algorithm uses a modified "surrogate" objective, which penalizes significant changes to the policy:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min\left(r_t(\theta)\hat{A}_t, clip\left(r_t(\theta), 1-\varepsilon, 1+\varepsilon\right)\hat{A}_t\right)\right], \tag{9}$$

$$r_t(\theta) = \frac{\pi_\theta\left(a_t \middle| S_t\right)}{\pi_{\theta_{\text{old}}}\left(a_t \middle| S_t\right)}. \tag{10}$$

The first term of the min function is the same as the objective of the TRPO algorithm, while the second term discourages updates that would move r_t outside of the $[1-\varepsilon, 1+\varepsilon]$ interval. As the minimum of the two terms is calculated, the final objective becomes a lower bound of the unclipped objective.

Utilizing a penalty instead of constraints enables using first-order algorithms, such as the Gradient Descent method, to optimize the objective. While the constraint may sometimes be violated, the computation is significantly simplified.

In the cases of neural network architectures that share parameters between the policy and the value function, the loss function must combine the policy surrogate and the value function error term. Furthermore, an entropy bonus can be added to the objective, which encourages exploration and prevents premature convergence to suboptimal strategies:

$$L(\theta) = \hat{\mathbb{E}}_t\left[L^{\text{CLIP}}(\theta) - c_1 L^{VF}(\theta) + c_2 S\left[\pi_\theta\right]\left(S_t\right)\right]. \tag{11}$$

Furthermore, PPO handles both discrete and continuous action spaces, hence it is applicable for a wide variety of RL problems.

2.3 Multi-agent reinforcement learning

To extend the sequential decision-making tasks of RL to multiple agents, Markov Games were introduced (Littman, 1994). It is a generalization of MDPs to numerous agents simultaneously interacting within a shared environment. For a discrete-time setting, it is formalized by the $\{N, S, \{A^i\}P, \{R^i\}\}$ tuple. Where $N = \{1, \ldots, n\}$ denotes the set of $n > 1$ interacting agents, S is the set of states observed by all agents, and the joint action space $A = A^1 \times \ldots \times A^n$ is the collection of individual action spaces from the agents. The transition probability function P describes the chance of a state transition, and each agent owns an associated reward function R^i.

Multi-Agent Reinforcement Learning (MARL) algorithms are categorized based on different factors, such as the designed reward structure. In the case of a fully cooperative setting, all agents receive the same $R = R^1 = \ldots = R^n$ reward for the state transitions. This reward structure motivates agents to collaborate and try to maximize the performance of the team instead of their individual performance.

Fully competitive settings are described as zero-sum Markov Games, where $R = \sum_{i=1}^{n} R^i = 0$. Therefore, these agents aim to maximize their rewards while minimizing the rewards of others.

In the case of mixed settings, also known as general-sum games, the setting is neither fully competitive nor fully cooperative. More generally, a setting can be called cooperative if the agents are encouraged to work together despite not sharing the same reward. In contrast, in cooperative settings, the agents are encouraged to work against each other, but the sum of rewards does not equal zero.

Besides the rewarding strategy, other taxonomies are also used, such as the taxonomy presented by (Claus and Boutilier, 1998) differentiating independent and joint-action learners. Independent learners do not consider other agents and have no information about the observations and actions of other agents. In contrast, joint-action learners observe the actions taken by other agents a-posteriori. In addition to the training paradigm, (Gronauer and Diepold, 2022) also differentiates two execution schemes. Hence, they divide MARL algorithms into three categories: centralized training, centralized execution (CTCE), decentralized training, decentralized execution (DTDE), and centralized training, decentralized execution (CTDE).

In this research, "parameter sharing" is utilized for training, a particular case within the CTDE strategy. In this case, all agents own and update the same policy network. The results of (Gupta et al., 2017) showed that agents trained using parameter sharing could be trained more efficiently and outperform independent learners. While this approach is the most beneficial if all agents share the same or similar goal, different agent behaviors can emerge due to different observed states.

2.4 Curriculum learning

Besides parameter sharing, curriculum learning (CL) (Gupta et al., 2017; Long et al., 2020) is also utilized in this research. CL aims to start the agent training using only a subset of the original problem with reduced complexity. As the training progresses, it gradually increases the difficulty of the training scenarios.

In MARL, tasks become more challenging as the number of agents increases. Hence, CL is used to learn an optimal policy. The training starts with a small number of agents, and then the number of agents is increased throughout the training.

3 Environment description

This paper presents the advantages and challenges of multi-agent reinforcement learning through the example of a yield-sign controlled, four-way intersection. The environment is part of the Farama Foundation's Highway-env project (Leurent, 2018a).

3.1 Intersection environment

The environment is shown in Fig. 2. In the scenario, the horizontal road has the right of way. Hence, the levels of priority are as follows:

1. horizontal straight lanes and right-turns;
2. horizontal left-turns;
3. vertical straight lanes and right-turn;
4. vertical left-turns.

This structure provides the possibility for future extensions, including e.g. the prioritization of vehicles transporting high numbers of passengers.

Of the available observation types, KinematicObservation is used in this paper. It is a $V \times F$ array, which lists the F features of the ego vehicle and the V–1 nearest vehicles. In this research, the default observed features are the following:

- presence;
- vehicle position on the x-axis;
- vehicle position on the y-axis;
- lateral velocity of the vehicle;
- longitudinal velocity of the vehicle;
- cosine of the vehicle's heading;
- sine of the vehicle's heading.

While most features describe the state of the vehicles surrounding the agent, the aim of the presence feature is different. The observation space must have a fixed size; hence, if fewer vehicles are observed, the array's last rows are filled with zeros. Thus, the presence feature disambiguates agents at zero offsets from non-existent agents.

The environment contains a low-level controller for both the longitudinal and lateral control. Their setpoints can be defined by the agent using the DiscreteMetaAction

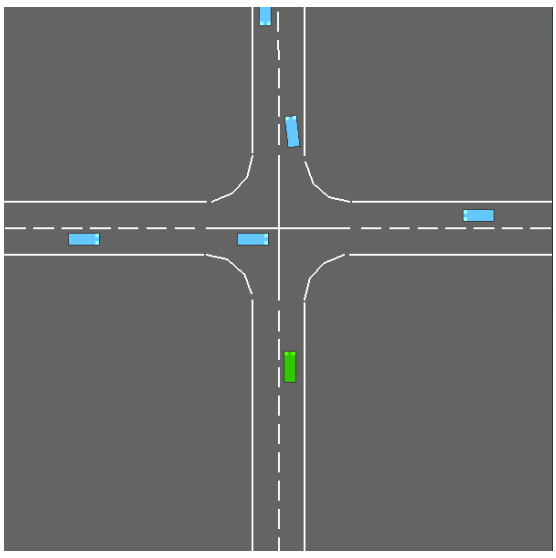

Fig. 2 Intersection environment

action type. In this case, the agent only controlled the longitudinal behavior of the vehicle through the actions: slower, idle, or faster. The lateral position control was handled by the environment.

3.2 Application programming interface

The environment's application programming interface (API) follows the rules provided by the gym API. Therefore, the environment is compatible with popular reinforcement learning tools like Stable-Baselines3 (SB3). SB3 supports the development of multi-agent reinforcement learning algorithms using parameter sharing. A multi-agent version is already provided to most environments of Highway-env, although in this case, its interface differs from SB3.

Thus, an API has been developed to adapt the environment's interface to enable learning using SB3. This API handles the input/output conversion between the environment and the agent while handling agent death. In MARL, the two most common strategies are to stop the episode if a single agent is terminated or if all of them are. In the first case, the truncated flag can prevent the otherwise surviving agents from thinking they have reached a terminal state. In the second case, the varying number of agents must be handled. The latter approach has been used to prevent the early termination of episodes at the start of the learning process.

If a single agent reaches its destination or collides with another vehicle, it is removed from the environment, but the episode will continue for the remaining agents. The observations of the removed agents are substituted with zero vectors of the proper size to provide observations with consistent size to the algorithm.

4 Training

4.1 Benchmark agents

The first step of the research is to reproduce a single-agent benchmark performance described in (Leurent and Mercat, 2019) with an SB3 DQN agent. The development of the rewards during the approximately 1-hour training process, which can be seen in Fig. 3, matches the results presented in the original paper, which indicates that the performance of the agent can be used for comparison.

After that, the environment was extended to a multi-agent setting, containing only four vehicles, each controlled by an agent. Cooperative rewarding has been used to model the task more precisely, where all agents receive a joint average reward. In both environments, agents have been trained with 1 and 5 Hz policy update frequencies, which are "1 fps" and "5 fps". The hyperparameters of these agents and the configurations for each environment have been determined according to (Leurent, 2018b), and are summarized in Table 1 and Table 2. The training

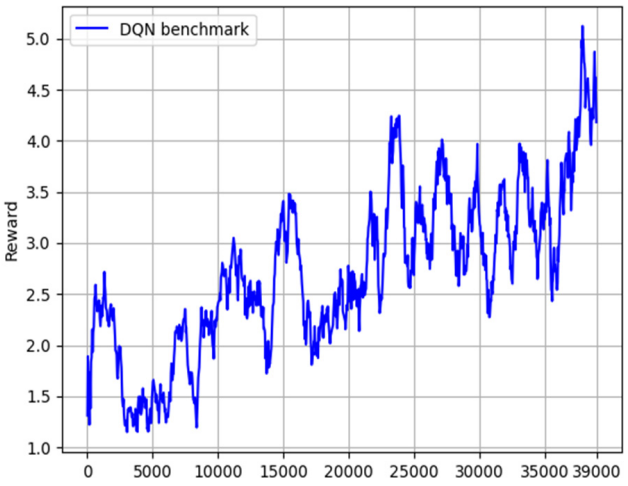

Fig. 3 Average rewards during the benchmark training process

Table 1 Hyperparameters of the benchmark agents

Parameter	DQN 1 fps	DQN 5 fps
Type of neural network	MLP	MLP
Activation function	ReLU	ReLU
Num. of hidden layers	2	2
Num. of neurons per layer	128, 128	128, 128
Optimizer	Adam	Adam
Learning rate	0.0005	0.0005
Discount factor	0.95	0.9
Batch size	64	64
Buffer size	15,000	15,000
Online network update interval	1	3
Target network update interval	512	50
Epsilon max	1	1
Epsilon min	0.05	0.05

Table 2 Configurations of the training environments

Parameter	Single-agent	Multi-agent
Absolute or relative values	Absolute	Absolute
Order of observation	1 fps: shuffled 5 fps: sorted	1 fps: shuffled 5 fps: sorted
Destination	Left lane	Left lane
Observed features	Original	Original
Duration [s]	13	13
Controlled vehicles	1	4
Initial (other) vehicles	10	0
Spawn probability	0.6	0
High speed reward range [m/s]	[7, 9]	[7, 9]
High speed reward	1	1
Arrived reward	1	1
Collision reward	-5	-5

process of each algorithm lasted 39,000 timesteps, about 3,000 episodes, which took around 15 min in real life.

4.2 PPO implementation

After training the benchmark agents, a PPO algorithm with a simple neural network was implemented and trained in the 5 fps multi-agent environment configuration. Then, the agents have been given a more challenging task by randomizing their destination.

To improve their performance, the reward values have been updated to represent the importance of each term more accurately. Besides that, the complexity of the neural network has been increased by extending it with more hidden layers and neurons. The modifications are presented in Table 3. The training process for the PPO agents lasted 30,000 timesteps, which required between 6 and 10 real life min in each case.

5 Simulation results

The performance of the trained agents has been evaluated in test batches consisting of 100 episodes. Due to the different rewards and number of training steps, examining the training diagrams would not accurately represent the differences in the agents' behaviors. With that in mind, the following metrics were chosen to compare the achieved efficiency by different agents:

Table 3 Hyperparameters of the PPO agents

Parameter	Original PPO	Updated PPO
Type of neural network	MLP	MLP
Activation function	ReLU	ReLU
Num. of hidden layers	policy network: 2 value network:2	policy network: 4 value network: 4
Num. of neurons per layer	policy n.: 128, 128 value n.: 128, 128	policy n.: 64, 512, 512, 64 value n.: 32, 256, 256, 32
Optimizer	Adam	Adam
Learning rate	0.0005	0.001
Discount factor	0.99	0.99
Batch size	64	64
Horizon (	128	256
Num. of epochs	10	10
Environment configurations:		
Destination	Left lane	Random
High speed reward range [m/s]	[7, 9]	[4.5, 9]
High speed reward	1	0.1
Arrived reward	1	1
Collision reward	-5	-5

- Success rate indicating the number of episodes in which agents avoided collision.
- The average number of agents who have reached their goal.

The results of the tests are summarized in Table 4, which are separately analyzed for each agent in the subsequent sections.

5.1 Single agent benchmark

First, the benchmark agent has been evaluated. The results are summarized in the first two rows of Table 4. As described in (Leurent and Mercat, 2019), the agents obtain a "risky and blind" policy. The ego-vehicle learns to accelerate through the intersection with the highest possible speed to maximize its speed-based reward. However, this often leads to collisions, or sometimes, the other vehicles controlled by the IDM model adjust their speeds so collisions can be avoided. Therefore, there is a slight increase in the success rate in the 5 fps environment because, in this case, other vehicles can adjust their speeds more accurately. Another factor that shows the assertive behavior of the agent is that no episode has been truncated, so they were only terminated by reaching the goal or colliding. When put in a multi-agent environment, the performance of the benchmark agent becomes questionable, causing more collisions and, thus, unable to reach the goal.

5.2 Multi-agent results

As mentioned earlier, the environment has been modified to contain only 4 vehicles, all controlled by separate agents

Table 4 Results of the evaluation for each agent

Agent	Success rate	Avg. reached goal
Single-agent DQN 1 fps	47%	0.47
Single-agent DQN 5 fps	55%	0.55
Single-agent DQN 5 fps Multi-agent environment	12%	1.85
Multi-agent DQN 1 fps	11%	2.07
Multi-agent DQN 5 fps	17%	2.19
Multi-agent PPO original, certain destination	62%	2.05
Multi-agent PPO original, random destinations	40%	2.04
Multi-agent PPO updated reward, original network	48%	2.93
Multi-agent PPO updated reward and netwoek	58%	3.08
Multi-agent PPO 1 fps using Curriculum Learning	81%	3.14

using the same policy. By that, they had to learn to cooperate to reach their goals. The rudimentary multi-agent training used the previously described DQN method, which resulted in an almost identical behavior. The worsening success rate, seen in the fourth and fifth row of Table 4, can be traced back to the increased number of interactions between vehicles controlled by an agent, which have all learned a selfish policy. Considering this, the cooperative reward did not help to improve the results. One cause of this problem can be the weights of the reward terms. By rewarding each timestep's high speeds with the same value as the return received when reaching the goal, agents might learn a suboptimal policy by accelerating early in the episode. This issue was later considered in PPO training.

Next, the performance of the simple PPO algorithm has been evaluated with both specific and randomized destinations for the agents. As shown in the sixth and seventh rows of Table 4, the agents achieved a significantly increased success rate. However, the other examined metric demonstrates that the agents learned to avoid entering the intersection in some scenarios. This policy resulted in notably fewer collisions, but on the other hand, fewer agents have reached their goals, which is far from the desired behavior. By randomizing the destinations, agents received a task that models real-life scenarios more accurately. However, as expected, this modification slightly increased the number of collisions.

Agents have been trained with randomized destinations to increase the performances they achieve. On top of that, the high-speed reward term has been significantly decreased due to the previously explained weighting issue. The results from evaluating these policies can be found in the eighth and ninth rows of Table 4. These modifications helped slightly lower the number of collisions but, more importantly, increased the average passage rate, even with the original, simple neural network. By increasing the complexity of the neural network, the performance of the agents surpassed the benchmark results in terms of success rate and achieved a noteworthy average of reached goals.

The last row of Table 4 presents the results achieved using Curriculum Learning, where the training was divided into three parts. In the first part, a single agent must learn to reach its randomized destination in minimal traffic. Other drivers were controlled by IDM and MOBIL models. The agent got a 100% success rate in these scenarios. The second part consisted of 4 agents. Collision was not penalized during this segment of the training, although early termination of the episodes due to collision results in a decreased cumulative reward. The reward strategy has also been slightly modified. Instead of using purely cooperative rewards, each agent received 50% of its reward and 50% of the average reward of the agents. These modifications resulted in a 69% success rate, and 2.37 agents reached their goal on average. In the third part, the collision was also penalized, resulting in an 81% success rate, and 3.14 agents got their goals on average.

6 Conclusions and future work

Single-agent algorithms may learn to achieve the highest possible rewards, but often by developing a selfish policy, which is unacceptable in real-life scenarios such as the example of an intersection. In this paper, a multi-agent PPO algorithm was presented to overcome the results of a single-agent benchmark policy in an intersection, which was achieved by first modifying the environment configurations to be able to model real-life traffic situations more accurately, then by optimizing the training hyperparameters. With that, the demonstrated algorithm solved the complex, cooperative decision-making task with increased success. At last, curriculum learning was utilized to enhance the performance of the agents further, and it was able to surpass agents even with higher control frequency.

Acknowledgement

Supported by the ÚNKP-23-1-I-BME-214 New National Excellence Program of the Ministry for Culture and Innovation from the source of the National Research, Development and Innovation Fund.

References

Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., …, Zhang, S. "Dota 2 with Large Scale Deep Reinforcement Learning", [preprint] arXiv, arXiv:1912.06680, 10 March 2021. https://doi.org/10.48550/arXiv.1912.06680

Chen, B., Xu, M., Liu, Z., Li, L., Zhao, D. "DelayAware MultiAgent Reinforcement Learning for Cooperative and Competitive Environments", [preprint] arXiv, arXiv:2005.05441, 29 August 2020. https://doi.org/10.48550/arXiv.2005.05441

Chen, D., Hajidavalloo, M. R., Li, Z., Chen, K., Wang, Y., Jiang, L., Wang, Y. (2023) "Deep Multi-Agent Reinforcement Learning for Highway On-Ramp Merging in Mixed Traffic", IEEE Transactions on Intelligent Transportation Systems, 24(11), pp. 11623–11638. https://doi.org/10.1109/TITS.2023.3285442

Claus, C., Boutilier, C. (1998) "The Dynamics of Reinforcement Learning in Cooperative Multiagent Systems", In: Fifteenth National Conference on Artificial Intelligence (AAAI-98), Madison, WI, USA, pp. 746–752. ISBN 9780262510981 [online] Available at: https://cdn.aaai.org/AAAI/1998/AAAI98-106.pdf [Accessed: 31 July 2025]

Farooq, M. S., Khalid, H., Arooj, A., Umer, T., Asghar, A. B., Rasheed, J., Shubair, R. M., Yahyaoui, A. (2023) "A Conceptual Multi-Layer Framework for the Detection of Nighttime Pedestrian in Autonomous Vehicles Using Deep Reinforcement Learning", Entropy, 25(1), 135. https://doi.org/10.3390/e25010135

Gronauer, S., Diepold, K. (2022) "Multi-agent deep reinforcement learning: a survey", Artificial Intelligence Review, 55(2), pp. 895–943 https://doi.org/10.1007/s10462-021-09996-w

Gupta, J. K., Egorov, M., Kochenderfer, M. (2017) "Cooperative Multi-agent Control Using Deep Reinforcement Learning", In: Autonomous Agents and Multiagent Systems, Sao Paulo, Brazil, pp. 66–83. ISBN 978-3-319-71681-7 https://doi.org/10.1007/978-3-319-71682-4_5

Kalantari, R., Motro, M., Ghosh, J., Bhat, C. (2016) "A distributed, collective intelligence framework for collision-free navigation through busy intersections", In: 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC), Rio de Janeiro, Brazil, pp. 1378–1383. ISBN 978-1-5090-1890-1 https://doi.org/10.1109/ITSC.2016.7795737

Kaushik, M., Phaniteja, S., Krishna, K. M."Parameter Sharing Reinforcement Learning Architecture for Multi Agent Driving Behaviors", [preprint], arXiv, arXiv:1811.07214, 17 November 2018. https://doi.org/10.48550/arXiv.1811.07214

Kesting, A., Treiber, M., Helbing, D. (2007) "General Lane-Changing Model MOBIL for Car-Following Models", Transportation Research Record: Journal of the Transportation Research Board, 1999(1), pp. 86–94. https://doi.org/10.3141/1999-10

Leurent, E. (2018a) "An Environment for Autonomous Driving Decision-Making", [online] Available at: https://github.com/eleurent/highway-env [Accessed: 31 July 2025]

Leurent, E. (2018b) "rl-agents: Implementations of Reinforcement Learning algorithms", [online] Available at: https://github.com/eleurent/rl-agents [Accessed: 31 July 2025]

Leurent, E., Mercat, J. (2019) "Social Attention for Autonomous Decision-Making in Dense Traffic", [preprint] arXiv, arXiv:1911.12250, 27 November 2019. https://doi.org/10.48550/arXiv.1911.12250

Li, N., Chen, P. (2024) "Research on a Personalized Decision Control Algorithm for Autonomous Vehicles Based on the Reinforcement Learning from Human Feedback Strategy", Electronics, 13(11), 2054. https://doi.org/10.3390/electronics13112054

Littman, M. L. (1994) "Markov games as a framework for multi-agent reinforcement learning", In: Machine Learning Proceedings 1994 - Proceedings of the Eleventh International Conference, New Brunswick, NJ, USA, pp. 157–163. ISBN 978-1-55860-335-6 https://doi.org/10.1016/B978-1-55860-335-6.50027-1

Long, Q., Zhou, Z., Gupta, A., Fang, F., Wu, Y., Wang, X. "Evolutionary Population Curriculum for Scaling Multi-Agent Reinforcement Learning", [preprint] arXiv, arXiv:2003.10423, 23 March 2020. https://doi.org/10.48550/arXiv.2003.10423

Ly, A. O., Akhloufi, M. (2021) "Learning to Drive by Imitation: An Overview of Deep Behavior Cloning Methods", IEEE Transactions on Intelligent Vehicles, 6(2), pp. 195–209. https://doi.org/10.1109/TIV.2020.3002505

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M. "Playing Atari with Deep Reinforcement Learning", [preprint] arXiv, arXiv:1312.5602, 19 December 2013. https://doi.org/10.48550/arXiv.1312.5602

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., …, Hassabis, D. (2015) "Human-level control through deep reinforcement learning", Nature, 518(7540), pp. 529–533. https://doi.org/10.1038/nature14236

Moghadam, M., Alizadeh, A., Tekin, E., Elkaim, G. H. (2021) "A Deep Reinforcement Learning Approach for Long-term Short-term Planning on Frenet Frame", In: 2021 IEEE 17th International Conference on Automation Science and Engineering (CASE), Lyon, France, pp. 1751–1756. ISBN 978-1-6654-1873-7 https://doi.org/10.1109/CASE49439.2021.9551598

Schwarting, W., Pierson, A., Alonso-Mora, J., Karaman, S., Rus, D. (2019) "Social behavior for autonomous vehicles", Proceedings of the National Academy of Sciences, 116(50), pp. 24972–24978. https://doi.org/10.1073/pnas.1820676116

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., …, Hassabis, D. "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm", [preprint] arXiv, arXiv:1712.01815, 05 December 2017. https://doi.org/10.48550/arXiv.1712.01815

Sutton, R. S., Barto, A. G. (2018) "Reinforcement Learning: An Introduction", Bradford Books. ISBN 978-0-262-03924-6

Treiber, M., Hennecke, A., Helbing, D. (2000) "Congested traffic states in empirical observations and microscopic simulations", Physical Review E, 62(2), pp. 1805–1824. https://doi.org/10.1103/PhysRevE.62.1805

Wang, P., Chan, C.-Y. (2018) "Autonomous Ramp Merge Maneuver Based on Reinforcement Learning with Continuous Action Space", [preprint], arXiv, arXiv:1803.09203, 25 March 2018. https://doi.org/10.48550/arXiv.1803.09203

Watkins, C. J. C. H. (1989) "Learning from Delayed Rewards", PhD thesis, King's College, Cambridge University.

Wolf, P., Kurzer, K., Wingert, T., Kuhnt, F., & Zöllner, J. M. (2018) "Adaptive Behavior Generation for Autonomous Driving using Deep Reinforcement Learning with Compact Semantic States", In: 2018 IEEE Intelligent Vehicles Symposium (IV), Changshu, China, pp. 993–1000. ISBN 978-1-5386-4453-9 https://doi.org/10.1109/IVS.2018.8500427