Decentralized Reinforcement Learning for Multi-Agent Navigation in Unconstrained Environments

Felix Förster[1,2] Qadeer Khan[1,2] Daniel Cremers[1,2]

***Abstract*—Supervised learning has demonstrated to be an effective strategy in training neural networks for vehicle navigation. However, it requires labeled data, which may not be available when a large number of vehicles need to be controlled simultaneously. In contrast, Deep Reinforcement Learning (DRL) circumvents the necessity for ground truth labels through environmental exploration. However, most concurrent DRL approaches either tend to operate in the discrete action/state space or do not consider the vehicle kinematics. In this paper, we use DRL to control multiple vehicles while also considering their kinematics. The task is for all the vehicles to reach their desired destination/target while avoiding collisions with each other or static obstacles in an unconstrained environment. For this, we propose a decentralized Proximal Policy Optimization (PPO) based DRL agent that independently provides control commands to each vehicle. The agent is based on two separate PPO models. The first is used to drive each vehicle to the proximity of its target. Once within the target's proximity, the second model is used to park that vehicle at the correct position and orientation. The decentralized nature of the algorithm allows each agent to rely only on information about its current state and target, along with details regarding the closest obstacle/agent. By scaling this approach to all vehicles, simultaneous navigation of multiple vehicles can be achieved. Experimental results show a collective strategy that allows consistent results across a wide range of scenarios while scaling to situations with up to 20 vehicles and 12 stationary obstacles.**

***Index Terms*—Multi-Agent Navigation, Multi-Vehicle Environment, Reinforcement Learning**

I. Introduction

In recent years, Deep Learning has established itself as a viable method for autonomous vehicle control, as demonstrated in [1], [2], [3]. However, scaling these methods tends to be constrained by the requirement of having supervised labels recorded by an expert driver. This matter is further complicated when not just one but multiple vehicles have to be controlled simultaneously. The labels for these multiple vehicles can be obtained using optimization-based algorithms [4]. However, the optimization suffers from scalability issues when control commands for an increasing number of vehicles must be determined.

[1]Computer Vision Group, School of Computation, Information and Technology, Technical University of Munich. Contact: {f.foerster, qadeer.khan, cremers}@tum.de

[2]Munich Center for Machine Learning

A viable alternative to the aforementioned problem is to use Deep Reinforcement Learning (DRL), which has shown increasingly good results across various problem domains [5], [6], [7]. Of particular interest to this paper is where DRL has been used for multi-agent navigation tasks [8], [9], [10], [11], [12], [13]. However, an issue of concern with these methods is that they either operate in the discrete action space, consider the vehicles to be holonomic, or both. This does not necessarily align with reality, where a car's longitudinal and lateral controls operate in the continuous action space. Moreover, the non-holonomic nature of the vehicles necessitates that the vehicle kinematics be considered when determining the control commands. Otherwise, it may lead to erratic driving.

In this paper, we also propose a DRL approach. It is meant for the task of navigating multiple vehicles to their desired destination while avoiding collisions with other vehicles and static obstacles. The distinguishing feature of our method from previous DRL approaches is that the vehicles navigate in continuous state and action space, and the non-holonomic behavior of the vehicles is integrated with the control commands to render realistic driving characteristics. In particular, our approach calculates control commands for each vehicle independently. It uses two separate Proximal Policy Optimization (PPO) models to simplify the navigation problem. The first covers large distances until the vehicle reaches a set distance threshold of the target. The other model then parks the vehicle on the target with precise controls. The models control each vehicle separately, which allows for decentralized calculation of all actions. The approach considers the current vehicle's state and destination as well as that of the closest vehicle/obstacle to determine the next action. This information is in the individual first-person view of each vehicle.

To evaluate the effectiveness of our framework, we benchmark against [4]. It operates on a continuous state/action space and models the vehicle kinematics using the bicycle model [14] and is therefore used as a baseline in the experiments. It utilizes a centralized attention-based Graphical Neural Network (A-GNN) controller to retrieve information from the surrounding vehicles and the control commands. The centralized nature of the A-GNN architecture necessitates redundant computations even for a pair of agents that do not necessarily influence each other's actions. These redundant computations

may slow down the real-time inference. In contrast, our method uses decentralized computation across the vehicles and does not require supervised labels for training, allowing for better scalability and faster real-time inference. In this regard, the contributions of our work are as follows:

1) Unlike other Reinforcement Learning (RL) methods, our framework for multi-agent navigation operates on the continuous state/action space while also considering the vehicle kinematics in the motion model.
2) In contrast to the supervised A-GNN baseline, our method scales better as the number of vehicles/obstacles increases without needing labeled data for training.
3) The decentralized nature of our algorithm allows inference of multiple vehicles in parallel. This results in a speedup that is an order of magnitude faster than the A-GNN baseline.

The application areas of our method in unstructured environments include warehouses where transportation platforms carry goods from a starting point to a destination.

II. Related Work

There are many facets to autonomous driving, focusing on single or multiple agents, navigating in structured traffic or unstructured/unconstrained road environments, selecting the appropriate input state representation, deciding on the mode of training, or even platforms for assessing performance. In this section, we refer to related works that allude to the aspects above. It is pertinent to mention that this work is meant for multi-vehicle control in unconstrained environments using RL, while also considering vehicle kinematics. Furthermore, our approach does not need any supervised ground truth labels.

BEV State Representation: In [15] and [16], a bird's eye view (BEV) state representation is used for single vehicle control in structured environments. They use an extensive amount of labeled data for training. This results in a comprehensive effort to create supervised ground truth labels. We also use a BEV-inspired state representation wherein the current, target, and closest obstacle state on a 2D plane are known to the network.

Evaluation Platforms: It is critical to evaluate vehicle control algorithms to assess their driving performance. While preliminary experiments in the real world might be tedious, many platforms simulate real-world road traffic to evaluate autonomous driving performance. Depending on the platform, they might include other cars, pedestrians, traffic regulations, and much more. Examples are StreetLearn [17], [18] and Carla [19]. Some approaches, such as [20], [21], [22], use Carla for evaluating single vehicle control algorithms in structured environments. Furthermore, the racing environment of Gymnasium also focuses on single vehicle control [23]. In our experiments, the simulation platform from [4] is used for evaluating multi-vehicle control in unconstrained environments.

Multi-Agent Trajectory Prediction and Control: Approaches using Graphical Neural Network (GNN) architectures have shown promising results in multi-agent trajectory prediction [24], [25], [26], [27] in structured environments. This work, in contrast, is meant for multi-agent control in unstructured/unconstrained environments using RL. The work of [8], [9], [10], [11], [12], [13], [28] also use RL for multi-agent navigation, but they either do not consider vehicle kinematics, act in discrete state/action space, or both. Meanwhile, [29], [30] can also navigate multiple agents towards their desired destination, as in this work. However, the critical difference is that they either consider the agents as particles or do not consider the non-holonomic constraints of the vehicles.

Formation Control: Furthermore, there have been approaches that control multiple vehicles through formation [31], [32], [33]. However, rather than being independent of each other, they use a lead vehicle to build up a formation to guide other vehicles.

Baseline: We use [4] as the baseline for comparisons in the evaluation section. The approach discusses an Attention Based Graphical Neural Network (A-GNN) paradigm that controls multiple vehicles. It considers the state and target of all vehicles and stationary obstacles to calculate each vehicle's action. For training, it uses labeled data based on an optimization-based procedure. With this data, a model can be trained to achieve good results in reaching targets with only a few collisions. It even generalizes to higher numbers of vehicles with decent results while outperforming other GNNs. Two main differences exist between the baseline A-GNN approach and the RL one presented in this project. First, the A-GNN uses data with extensive labeling effort. This limits the training data to only 1-3 vehicles and 0-4 obstacles due to the computational complexity of obtaining labels for a larger number of vehicles. Furthermore, the A-GNN chooses its action based on the information on all vehicles and obstacles. This information is inefficient and excludes many application areas where this data is unavailable. Moreover, all the actions are calculated by a centralized network rather than offering the possibility of decentralization [4].

III. Framework

There are two main components regarding the approach that we propose. First, a custom RL environment is designed to simulate vehicle kinematics and return feedback to the agent. Second, the agent architecture determines the fundamental navigation capabilities of the agent. The following section will go into the details of the respective design decisions.

A. Reinforcement Learning Environment

Vehicle Kinematics: For each vehicle, there are two variables with which the vehicle can be controlled: pedal acceleration (p) and steering (φ). The state of a vehicle is determined

by the x and y position, its orientation ζ, and velocity v. The actions can change this state based on the equations and order:

$$\begin{aligned} \zeta_{t+1} &= \zeta_t + v_t \cdot \frac{\tan(\varphi_t) \cdot \Delta t}{L} \\ v_{t+1} &= \beta \cdot v_t + p_t \cdot \Delta t \\ x_{t+1} &= x_t + v_t \cdot \cos(\zeta_t) \cdot \Delta t \\ y_{t+1} &= y_t + v_t \cdot \sin(\zeta_t) \cdot \Delta t \end{aligned} \tag{1}$$

where β represents environmental friction and L the wheel base. Using these equations results in a fast and accurate way of calculating the future car's states. Note that this calculation method is only reliable for a small Δt. For this reason, $\Delta t = 0.1$ is a good trade-off between accuracy and computational expenses. In addition, this allows the agent to update its actions with a quick reaction time.

Initialization and Termination: The initialization of an episode is specialized to generate difficult scenarios. To achieve a challenging scenario, we place obstacles in the path of the training vehicle. In addition, other vehicles are placed so that it is more likely that they cross paths with the primary training vehicle.

There are two conditions that could lead to the termination of the current episode. First, the maximum number of timesteps avoids unnecessarily long episodes where the vehicle is standing still or moving away from the desired area. In addition, there is the success state, which terminates the episode when reached. The exact conditions are discussed in detail later.

B. Reward Function

The reward function is another key part of the environment. It gives the agent feedback on its performance. The reward is constructed from the perspective of the environment and is the same for both the long and short-range models. More details on Task Splitting can be found in subsection III-C. Introducing subgoals is a great way to avoid sparse rewards. Sparse rewards would be highly data inefficient because reaching the goal by coincidence is almost impossible in such a challenging environment. The overall received reward at time t ($r_{total,t}$) is a combination of 4 sub-rewards: position $r_{dis,t}$, angle $r_{angle,t}$, time $r_{time,t}$, and collision avoidance $r_{obs,t}$. Each sub-reward constitutes achieving a particular subgoal. For example, $r_{dist,t}$ is the reward received for getting closer to the correct target position. We discuss this in detail in the subsequent paragraphs.

$$r_{total,t} = \begin{cases} r_{dis,t} + r_{obs,t} + r_{angle,t} + r_{time,t}, \\ \qquad\qquad \text{if } S_t \in \mathcal{S} \land t \leq t_{max} \\ r_{succ}, \\ \qquad\qquad \text{if } S_t \notin \mathcal{S} \land t \leq t_{max} \\ r_{trunc}, \\ \qquad\qquad \text{if } t > t_{max} \end{cases} \tag{2}$$

The combinations of all reward components represent the total reward r_{total}. Note that $\mathcal{S}$ is the set of all nonterminal states and S_t the current vehicle state at timestep t. All components are carefully balanced so that the model tries to maximize all of them instead of a subset. This is why all components contain an individual constant scaling factor α to balance them against each other (Equations 3, 4, 5, and 6). The respective subscript depicts the corresponding reward. The following subsections will describe all reward components in more detail.

The success reward sets the basic motivation for the agent to reach a goal state. Meanwhile, truncation provides additional punishment in case no goal state is reached after t_{max} timesteps. If an agent reaches a goal state in time, the model receives a high success reward r_{succ}, which results in $r_{total} = r_{succ}, S_t \notin \mathcal{S} \land t \leq t_{max}$. When reaching the maximum number of timesteps t_{max}, the model receives a truncation reward r_{trunc}. This results in $r_{total} = r_{trunc}, t > t_{max}$. Note that r_{trunc} is usually negative and punishes the agent.

Distance Reward, $r_{dis,t}$: The distance reward incentivizes the agent to minimize the distance to the target position. It increases the closer the vehicle is to the target. In the case that it is really far away, a minimal positive reward γ is maintained.

$$\begin{aligned} \hat{r}_{dis,t} &= \max\{\gamma, \alpha_{dis} \cdot (c_{dis} - d_t)\} \\ r_{dis,t} &= \begin{cases} \hat{r}_{dis,t}, & \text{if } d_t < d_{t-1} \lor d_t < d_c \\ -\hat{r}_{dis,t}, & \text{if } d_t \geq d_{t-1} \land d_t \geq d_c \end{cases} \end{aligned} \tag{3}$$

Equation 3 formulates this linear distance reward concept, where d_t is the distance of the vehicle from its target destination at the timestep t and c_{dis} is a constant determined by the environment size. In addition, we introduce the requirement that the agent only receives a positive distance reward if the vehicle decreases its target distance in the corresponding timestep d_t. If this is not the case, the agent receives the same reward but negatively. This condition is disabled if the agent is closer than d_c to the target because the agent is sometimes forced to move away to improve its angle.

Angle Reward, $r_{angle,t}$: One of the subgoals is orienting the vehicle at the correct angle when stopping on the target. The agent is motivated by the angle reward to achieve this. This reward component is mainly based on the angle difference $\zeta_{diff}, 0 \leq \zeta_{diff} \leq \pi$ between the car's and target's orientation. Based on this angle difference, the angle reward is calculated as described in equation 4.

$$r_{angle,t} = \begin{cases} \alpha_{angle} \cdot (0.5 - \frac{\zeta_{diff,t}}{\pi}) \cdot \left| r_{dis,t} \right|, \\ \qquad\qquad \text{if } d_t < d_{a\_min} \\ 0, \\ \qquad\qquad \text{if } d_t \geq d_{a\_min} \end{cases} \tag{4}$$

The angle reward improves proportionally to the angle difference. In addition, the angle reward is more important the closer the vehicle is to the target. This is considered

by multiplying by the absolute distance reward. It is important to note that the reward is only used in case the vehicle is closer to the target than d_{a_min} because there is no point in rewarding an accurate angle if the vehicle is far away.

Obstacle Reward, r_{obs}: The obstacle reward tells the agent to avoid collisions. There are three possible scenarios. First, the vehicle is far away from all obstacles. In this case, the agent is doing everything right. Because this will be the case most often, it is rewarded with a neutral score of $r_{obs} = 0$. If the vehicle is close to an obstacle but has not yet collided, the environment punishes the agent in proportion to its proximity to the obstacle. Last but not least, the agent receives a high negative reward r_{coll} in case of a collision. A collision appears if the center points of the vehicle and obstacle are less than d_{coll} apart. The combined function can be seen in equation 5 with $d_{o,i}$ being the distance to the obstacle i, n the maximum number of obstacles (including vehicles), d_{min} being the minimal distance to receive no punishment, and d_{coll} the distance at which the vehicle collides. $d_{o,0}$ is the vehicle itself and therefore excluded.

$$r_{obs} = \sum_{i=1}^{n} \begin{cases} r_{coll}, & \text{if } d_{o,i} \leq d_{coll} \\ \alpha_{obs} \cdot \frac{r_{coll}}{d_{o,i}}, & \text{if } d_{coll} < d_{o,i} < d_{min} \\ 0, & \text{if } d_{o,i} \geq d_{min} \end{cases} \tag{5}$$

Time Reward: If the rewards are not balanced appropriately, there is a risk that the RL agent tries to find the easiest way to gain as much reward as possible. For example, if the α_{angle} is high, the vehicle might orient itself to the target and then remain stationary, as this will grant it huge rewards at every subsequent timestep. Therefore, we introduce a time reward to disincentivize this behavior, as given in Equation 6. The longer it takes the vehicle to reach the target, the more negative reward it receives.

$$r_{time} = -\alpha_{time} \cdot \mu \cdot t \tag{6}$$

Note that μ is the step size in the environment. These unintended strategies run for the entire duration, while intended strategies that reach a successful state do not. The environment uses this difference and punishes the agent the longer it takes to reach a successful state, which results in a massive punishment for these trivial techniques. In addition, it introduces a motivation to finish quickly.

C. Reinforcement Learning Agent

We train the RL agent using the PPO-Clip on-policy algorithm, based on the following objective [5]:

$$q_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$$

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min(q_t(\theta)\hat{A}_t, \text{clip}(q_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right] \tag{7}$$

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t \left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 \kappa[\pi_\theta](s_t) \right] \tag{8}$$

where κ is an entropy bonus that encourages the algorithm to explore, and $q_t(\theta)$ is the probability ratio of the old and new policy based on the parameters θ, and ϵ is a hyperparameter used for clipping. Furthermore, c_1 and c_2 are coefficients, L_t^{VF} represents the learned state-value function, and $\hat{A}_t$ is an estimator of the advantage function at timestep t. A more detailed explanation of the PPO-Clip optimization objective can be found in the original work [5].
The task of each agent is to output the pedal and steering action command at every timestep.

Observation Space: The input values to a model need to represent the state of the environment sufficiently. However, redundant information could add to the complexity. The most important information necessary to fully display the environment is the position of the car, the target, the speed and angle of the car, and additional information about obstacles and other vehicles.
Using the absolute position and angle values as input leads to additional complexity for the agent because different values represent situations that are the same from the agent's perspective. This is why the information the agent gets is in the respective first-person perspective. This means that the agent's vehicle corresponds to $(x, y) = (0, 0)$ and $\zeta_{car} = 0$. This transformation reduces the number of input values because the vehicle's x and y positions and its angle are fixed. Most obstacles are likely to be irrelevant because they are not close to the vehicle being controlled. We therefore only consider information about the obstacle closest to the vehicle of interest. The position and angle of obstacles are also adapted based on the agent's first-person perspective.

Model Size: Both PPO models contain an input layer corresponding to the observation space, two linear layers with 256 neurons each, and an output layer with the two respective actions. This results in ∼137k parameters for each model. While trying different model sizes, models with this size have empirically demonstrated that they can generalize well while maintaining stability when exposed to shifts in the training data distribution, like a varying number of obstacles and vehicles. It strikes the right balance by being computationally manageable and expedient for the task.

Task Splitting: The central task to solve can be decomposed into two subtasks, which are both solved by a separate policy. The first subtask is navigating a vehicle to its target's proximity without collisions. For this task, we utilize the first policy π_{θ_l}, which navigates a vehicle across large distances while avoiding collisions. The second subtask is precisely reaching and stopping at the target with the correct orientation and position. To address this, we use the second policy π_{θ_s}, which handles short distances that require precise vehicle navigation. A threshold radius R_{switch} from the target position is used to

decide when to use each policy. Equation III-C defines this strategy more formally:

$$\pi_t(d_t) = \begin{cases} \pi_{\theta_s}, & \text{if } d_t < R_{switch} \\ \pi_{\theta_l}, & \text{else} \end{cases}$$

with π_t being the policy used at timestep t and θ_s, θ_l representing the short and long-range policy parameters, respectively.

IV. Experiments

Training Process: The final short-range and long-range PPO models were trained for 40 million and 30 million timesteps, respectively. All timesteps are only seen once because of the on-policy property of the PPO algorithm. The long-range version is trained exclusively on scenarios with 4 vehicles and 3 stationary obstacles, while the short-range model is based on 1-8 vehicles and 0-8 stationary obstacles. One of the vehicles is used as the primary training vehicle. This means we update its policy every timestep based on the reward for its action. To simulate the other cars, we frequently copy the primary vehicle policy to all other cars in the environment. This way, the primary training vehicle interacts with vehicles based on almost the same policy.

Evaluation: The evaluation is based on the datasets used in the baseline A-GNN paper [4]. Furthermore, other test datasets containing more vehicles and stationary obstacles were added. The complete test suite includes 1-20 vehicles and 0-12 stationary obstacles. Each test case is tested on 4062 scenarios. Note that cases with 6 or more vehicles only contain up to 8 stationary obstacles because otherwise, there would not be enough space for the vehicles to navigate. None of the test datasets were ever used during training. The goal is that all vehicles stop at their target position, including the correct angle, without colliding. Two criteria track the performance:

1) The success-to-goal rate defines the percentage of vehicles reaching their target state without collisions. A vehicle is in a target state if its center is closer to its target than 1.25 meters and the angle difference is less than 0.2 radians. In addition, the vehicle has to be slower than 0.05m/s. For this metric, a higher number is better.
2) The collision rate is the total number of collisions divided by the total distance traveled. For this metric, a lower number is better.

The quantitative results of the evaluation can be seen in table I. One preference of the model is evading obstacles by moving to the left. This seemingly unimportant detail plays an enormous role when scaling the number of vehicles. It is difficult to notice this preference for less than four vehicles, but it shows massive benefits for scenarios of ten or more vehicles. In these cases, the individual preference of each agent contributes to a collective strategy. This collective strategy is driving in a clockwise circle until the agent's target is reached. Because all agents are doing this, the number of collisions can be minimized.

While a heuristic approach could imitate this clockwise movement, introducing and exiting this motion is not as simple to achieve. In addition, vehicles potentially encounter stationary obstacles or parked vehicles, which have to be avoided. Considering these challenges, a complex heuristic approach would be necessary in combination with manual fine-tuning. In contrast, the ability to manage these challenges emerges naturally for our RL based approach.

The success-to-goal rate of the proposed Proximal Policy Optimization based DRL approach is relatively stable across all test cases. A consistent performance emerges if the number of other vehicles and stationary obstacles increases. The group behavior of the agents can explain this. In dense situations, the vehicles align themselves in a structure that allows them to move relatively safely. This explains the relatively high success-to-goal and low collision rates for many vehicles. This is especially impressive, considering the models were only trained on up to eight vehicles and obstacles, and only have information about themselves and their closest obstacle.

Comparison: Table I compares our approach against the baseline Attention-based Graphical Neural Network. This A-GNN Network was trained on labeled data generated by an optimization algorithm. In addition, it uses information about all vehicles, targets, and obstacles for its calculations rather than only the ego-vehicle, its target, and the closest obstacle. There is one difference in the physics simulation of the A-GNN. The physics simulation adapts the coordinates every timestep based on the old velocity and steering angle. Afterward, the respective vehicle velocity and steering angle will be updated. This means the actions taken do not affect the imminent resulting coordinates but only the ones after them. Because the RL agent receives an immediate reward for its action, it is important that the action that is taken actually affects the coordinates that the reward is based on. For this reason, the order of the physics simulation was changed so that the RL agent could update velocity and steering angle and then update the vehicle position. The old order was used to evaluate the baseline A-GNN to keep consistency with the physics used in training.

While the A-GNN performs better on a small number of vehicles and stationary obstacles, it rapidly drops when they are increased. Our approach performs more consistently and clearly outperforms in cases of ten or more vehicles. It is also more resilient to a high number of stationary obstacles.

Sometimes, the collision rate is higher compared to the A-GNN model. The most severe difference is evident in cases with a large number of obstacles. In these scenarios, the A-GNN model follows a conservative approach, resulting in vehicles not moving and, therefore, leading to relatively low collision rates.

The RL model performs consistently regarding the collision and success-to-goal rates for more than ten vehicles, while the A-GNNs drop rapidly in performance. There are two plausible reasons for that. The first one is that the DRL model only consists of 274k parameters compared to the 474k of the A-GNN. Small models tend to generalize better

		success to goal rate ↑		collision rate ↓	
Number of Vehicles	Number of Stationary Obstacles	Our Agent	Baseline A-GNN	Our Agent	Baseline A-GNN
1*	0*	0.93822	**0.9963**	-	-
1*	4*	**0.8943**	0.6829	1.07E-03	**6.74E-04**
1	8	**0.8223**	0.8114	3.83E-03	**4.69E-04**
1	12	0.7657	**0.792**	6.87E-03	**7.73E-04**
2*	0*	0.9121	**0.9974**	2.69E-04	**0.00E+00**
2*	4*	**0.9226**	0.6384	2.46E-03	**3.62E-04**
2	8	**0.9221**	0.3832	2.10E-03	**9.86E-04**
2	12	**0.8895**	0.2768	4.06E-03	**1.60E-03**
4*	0*	0.9084	**0.9905**	5.68E-04	**5.09E-05**
4	4	**0.8839**	0.8564	2.18E-03	**8.42E-04**
4	8	**0.8896**	0.7052	**1.77E-03**	1.91E-03
4	12	**0.8789**	0.5871	**2.61E-03**	3.35E-03
6*	0*	0.8824	**0.9665**	9.78E-04	**3.60E-04**
6	4	**0.8760**	0.8689	**1.12E-03**	1.38E-03
6	8	**0.867**	0.7654	**1.64E-03**	2.77E-03
10	0	0.85	**0.852**	**1.02E-03**	1.91E-03
10	4	**0.842**	0.7724	**1.31E-03**	3.04E-03
10	8	**0.8399**	0.6978	**1.52E-03**	4.33E-03
15	0	**0.839**	0.6029	**8.29E-04**	5.12E-03
15	4	**0.8356**	0.5231	**9.88E-04**	6.42E-03
15	8	**0.7711**	0.4689	**3.05E-03**	7.54E-03
20	0	**0.8297**	0.3161	**8.10E-04**	9.99E-03
20	4	**0.7925**	0.2563	**1.74E-03**	1.15E-02
20	8	**0.7567**	0.2153	**2.83E-03**	1.29E-02

TABLE I: Shows the performance of our proposed decentralized RL model and the baseline A-GNN approach [4]. As metrics, the success-to-goal rate and the collision rate are used. The test datasets are based on 4062 scenarios per case, including 1-20 vehicles and 0-12 obstacles. This test data has never been seen before by either model. Note that datasets marked with an asterisk (*) are test datasets from the baseline evaluation [4], while all others are newly generated test datasets.

on test cases that deviate from the training data. Another factor is the information gap between the two models. The input data is mostly the same from the long-range PPO model perspective. The A-GNNs have to process information on all vehicles and stationary obstacles. There is a massive difference between managing three cars and twenty. The change in data distribution does not severely affect our model because of the simplicity of the agent's input.

Average Run Time: Run time is a crucial factor in critical real-world applications such as driving. The presented RL approach offers the possibility to efficiently calculate the respective action on the vehicle itself without any centralized computation. Table II shows the runtimes for 1-20 vehicles and 0-8 obstacles. Our approach's performance evaluation is based on batch processing on a single CPU. Our approach is able to calculate actions in a fraction of a millisecond and is between 10 and 30 times faster compared to the baseline A-GNN. This is due to the small model size of the agent and the fact that the agent's input does not gain complexity with an increasing number of entities.
The agent can calculate thousands of actions per second, which is tremendously faster than necessary in real-world applications. Sufficient speeds could even be achieved with less capable hardware. This allows cheap onboard computation.

V. Future Work

An idea for improving the existing method is the introduction of intelligent obstacle selection. The obstacle with the highest collision risk with the ego-vehicle is chosen to be fed into the model, instead of selecting based on which is closest in proximity. This avoids cases where information about the closest obstacles with low collision risk is used.

VI. Conclusion

We proposed a Proximal Policy Optimization based Deep Reinforcement Learning Model that provides vehicle control commands. Our model acts in a continuous action space and takes vehicle kinematics into account. The initial RL problem was simplified using a first-person perspective and Task Splitting. It performs consistently well across various scenarios while being computationally and information efficient when compared with the baseline. It does not depend on labeled data and only relies on the information about the agent's vehicle and its closest obstacle. The actions of each agent can be calculated using its own processing unit. This decentralization allows for conveniently handling more vehicles than originally trained for. This behavior enables consistent performance for up to 20 vehicles and 12 stationary obstacles while maintaining similar collision rates as opposed to the centralized A-GNN. Despite the information disadvantage and decentralized action selection, the approach clearly outperforms the comparable baseline approach in high-density cases.

		Step Time (CPU) ↓	
Num. Veh.	Num. Obs.	Our Agent	Baseline A-GNN
1	0	**1.69E-04**	1.93E-03
1	8	**2.12E-04**	2.61E-03
2	0	**2.20E-04**	2.41E-03
2	8	**2.17E-04**	2.80E-03
4	0	**2.20E-04**	2.71E-03
4	8	**2.20E-04**	3.19E-03
6	0	**2.33E-04**	2.99E-03
6	8	**2.27E-04**	3.74E-03
10	0	**2.37E-04**	3.83E-03
10	8	**2.37E-04**	4.82E-03
15	0	**2.52E-04**	5.09E-03
15	8	**2.47E-04**	6.31E-03
20	0	**2.47E-04**	6.50E-03
20	8	**2.54E-04**	7.67E-03

TABLE II: Shows the average runtime per step for the proposed RL architecture and the baseline A-GNN in seconds. Each step includes the calculation of each vehicle's control commands while only considering the model's forward paths. The agent uses batch processing to calculate all vehicle actions at once. CPU: AMD Ryzen 7 7800X3D

Acknowledgements: We thank Yining Ma for his helpful insights regarding the baseline A-GNN.

REFERENCES

[1] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," 2016. [Online]. Available: https://arxiv.org/abs/1604.07316

[2] M. Toromanoff, E. Wirbel, F. Wilhelm, C. Vejarano, X. Perrotton, and F. Moutarde, "End to end vehicle lateral control using a single fisheye camera," in *2018 IEEE/RSJ IROS*, 2018.

[3] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," in *NeurIPS*, vol. 1, 1989.

[4] Y. Ma, Q. Khan, and D. Cremers, "Multi agent navigation in unconstrained environments using a centralized attention based graphical neural network controller," in *2023 IEEE 26th ITSC*. IEEE, Sep. 2023. [Online]. Available: http://dx.doi.org/10.1109/ITSC57777.2023.10422072

[5] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *preprint arXiv:1707.06347*, 2017.

[6] V. Mnih, "Playing atari with deep reinforcement learning," *preprint arXiv:1312.5602*, 2013.

[7] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proceedings of the 35th ICML*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80, 2018. [Online]. Available: https://proceedings.mlr.press/v80/haarnoja18b.html

[8] S. Nayak, K. Choi, W. Ding, S. Dolan, K. Gopalakrishnan, and H. Balakrishnan, "Scalable multi-agent reinforcement learning through intelligent information aggregation," in *ICML*, 2023.

[9] Z. Li, Y. Yang, and H. Cheng, "Efficient multi-agent cooperation: scalable reinforcement learning with heterogeneous graph networks and limited communication," *Knowledge-Based Systems*, 2024.

[10] Y. Hu, J. Fu, and G. Wen, "Graph soft actor–critic reinforcement learning for large-scale distributed multirobot coordination," *IEEE transactions on neural networks and learning systems*, 2023.

[11] C. et al., "Graph neural network and reinforcement learning for multi-agent cooperative control of connected autonomous vehicles," *Computer-Aided Civil and Infrastructure Engineering*, 2021.

[12] A. et al., "Learning transferable cooperative behavior in multi-agent teams," *arXiv preprint arXiv:1906.01202*, 2019.

[13] J. Jiang, C. Dun, T. Huang, and Z. Lu, "Graph convolutional reinforcement learning," *arXiv:1810.09202*, 2018.

[14] D. Wang and F. Qi, "Trajectory planning for a four-wheel-steering vehicle," in *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)*, vol. 4. IEEE, 2001.

[15] M. Bansal, A. Krizhevsky, and A. Ogale, "Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst," 2018.

[16] D. Chen, B. Zhou, V. Koltun, and P. Krähenbühl, "Learning by cheating," 2019. [Online]. Available: https://arxiv.org/abs/1912.12294

[17] P. Mirowski and et al., "Learning to navigate in cities without a map," in *NeurIPS*, 2018.

[18] P. W. Mirowski, A. Banki-Horvath, K. Anderson, D. Teplyashin, K. M. Hermann, M. Malinowski, M. K. Grimes, K. Simonyan, K. Kavukcuoglu, A. Zisserman, and R. Hadsell, "The streetlearn environment and dataset," *ArXiv*, vol. abs/1903.01292, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:67855375

[19] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[20] K. Chitta, A. Prakash, B. Jaeger, Z. Yu, K. Renz, and A. Geiger, "Transfuser: Imitation with transformer-based sensor fusion for autonomous driving," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, 2022.

[21] D. Chen and P. Krähenbühl, "Learning from all vehicles," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

[22] Q. Khan, I. Sülö, M. Öcal, and D. Cremers, "Learning vision based autonomous lateral vehicle control without supervision," *Applied Intelligence*, vol. 53, no. 16, pp. 19 186–19 198, 2023.

[23] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. D. Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, H. Tan, and O. G. Younis, "Gymnasium: A standard interface for reinforcement learning environments," 2024. [Online]. Available: https://arxiv.org/abs/2407.17032

[24] Y. Wang, Z. Liu, H. Lin, J. Li, R. Li, and J. Wang, "Vif-gnn: A novel agent trajectory prediction model based on virtual interaction force and gnn," in *2023 IEEE IV*. IEEE, 2023.

[25] X. M. et al., "Graph and recurrent neural network-based vehicle trajectory prediction for highway driving," in *2021 IEEE ITSC*, 2023.

[26] D. Zhu, Q. Khan, and D. Cremers, "Multi-vehicle trajectory prediction and control at intersections using state and intention information," *Neurocomputing*, vol. 574, p. 127220, 2024.

[27] Y. Liu, X. Qi, E. A. Sisbot, and K. Oguchi, "Multi-agent trajectory prediction with graph attention isomorphism neural network," in *2022 IEEE IV*. IEEE IV, 2022.

[28] G. Sartoretti, J. Kerr, Y. Shi, G. Wagner, T. K. S. Kumar, S. Koenig, and H. Choset, "Primal: Pathfinding via reinforcement and imitation multi-agent learning," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, 2019. [Online]. Available: http://dx.doi.org/10.1109/LRA.2019.2903261

[29] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," *NeurIPS*, vol. 30, 2017.

[30] S. Zhang, O. So, K. Garg, and C. Fan, "Gcbf+: A neural graph control barrier function framework for distributed safe multi-agent control," *IEEE Transactions on Robotics*, pp. 1–20, 2025.

[31] S. M. et al., "Distributed leader-following formation control for multiple nonholonomic mobile robots via bioinspired neurodynamic approach," *Neurocomputing*, vol. 492, 2022.

[32] S. He, R. Xu, Z. Zhao, and T. Zou, "Vision-based neural formation tracking control of multiple autonomous vehicles with visibility and performance constraints," *Neurocomputing*, vol. 492, 2022.

[33] R. Dutta, H. Kandath, S. Jayavelu, L. Xiaoli, S. Sundaram, and D. Pack, "A decentralized learning strategy to restore connectivity during multi-agent formation control," *Neurocomputing*, vol. 520, 2023.