

Transferring Multi-Agent Reinforcement Learning Policies for Autonomous Driving using Sim-to-Real

Eduardo Candela^{*1}, Leandro Parada^{*1}, Luis Marques^{*1},
Tiberiu-Andrei Georgescu¹, Yiannis Demiris², Panagiotis Angeloudis¹

Abstract— Autonomous Driving requires high levels of coordination and collaboration between agents. Achieving effective coordination in multi-agent systems is a difficult task that remains largely unresolved. Multi-Agent Reinforcement Learning has arisen as a powerful method to accomplish this task because it considers the interaction between agents and also allows for decentralized training—which makes it highly scalable. However, transferring policies from simulation to the real world is a big challenge, even for single-agent applications. Multi-agent systems add additional complexities to the Sim-to-Real gap due to agent collaboration and environment synchronization. In this paper, we propose a method to transfer multi-agent autonomous driving policies to the real world. For this, we create a multi-agent environment that imitates the dynamics of the Duckietown multi-robot testbed, and train multi-agent policies using the MAPPO algorithm with different levels of domain randomization. We then transfer the trained policies to the Duckietown testbed and compare the use of the MAPPO algorithm against a traditional rule-based method. We show that the rewards of the transferred policies with MAPPO and domain randomization are, on average, 1.85 times superior to the rule-based method. Moreover, we show that different levels of parameter randomization have a substantial impact on the Sim-to-Real gap.

I. INTRODUCTION

Coordination of Autonomous Vehicles (AVs) in traffic is a hard problem with a non-trivial optimization objective. On the one hand, rule-based policies can provide acceptable solutions to specific and heavily-simplified situations, but cannot possibly cover all scenarios that can happen in the real world [1], and require manual crafting of the rules. Moreover, rule-based methods cannot achieve effective coordination between agents and cannot adapt to changing environments. On the other hand, multi-agent learning algorithms can achieve effective agent-agent coordination and can potentially explore a great number of different environment configurations.

Multi-Agent Deep Reinforcement Learning (MARL) is potentially a powerful scalable framework for developing control policies for AVs. MARL uses Deep Neural Networks to represent complex value functions or agents policies that would otherwise require specific hand-crafted rules. However, MARL cannot be trained in live Autonomous Driving systems due to safety concerns [2], [3]. Furthermore, these algorithms must be trained using thousands or even

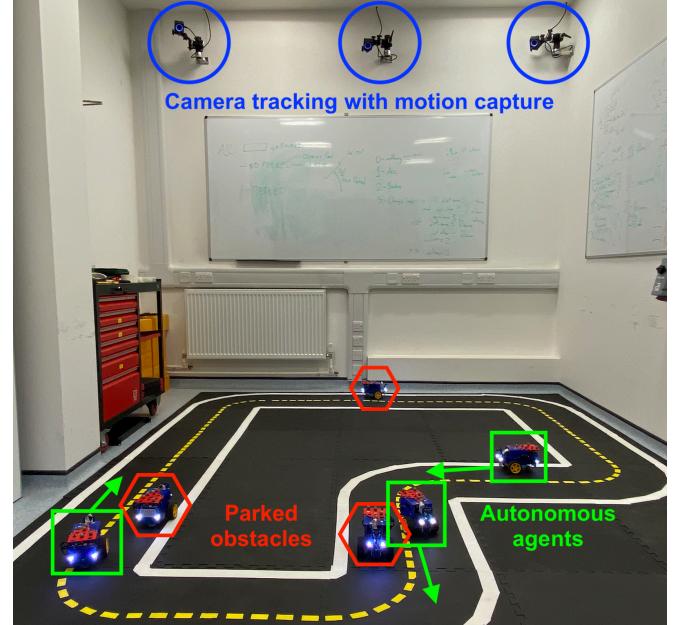


Fig. 1. The goal is to train autonomous vehicles to go around a track as fast as possible while avoiding collisions, parked obstacles, and leaving the track. Policies are trained using Multi-Agent Deep Reinforcement Learning in a simulated environment with a kinematic model and different levels of domain randomization that reduce the *sim-to-real gap* by adding noise and uncertainty to the simulations. The trained policies are then transferred to a real fleet of Duckiebot robots, with a motion capture camera system that obtains the vehicles' real pose.

millions of steps, which cannot be achieved in the real world. Therefore, MARL algorithms must be trained in simulation environments, and then the policies can be transferred to the real system.

Simulations have been an essential tool in the field of reinforcement learning, providing a flexible environment to train and test new algorithmic developments efficiently. However, the performance of policies trained in simulation is not likely to be replicated in the real world. The difference between a simulated environment and its real counterpart is called *reality gap* [4], and is usually observed through the overfitting of the learned policy to the unique characteristics of the simulations.

The *reality gap* can become even larger for multi-agent systems, such as those involving AVs. These systems introduce additional complexities to the Sim-to-Real problem, such as agent collaboration, environment synchronization and limited agent perception. Little work has been done on

^{*}These authors contributed equally.

¹E. Candela, L. Parada, L. Marques, T. Georgescu and P. Angeloudis are with the Centre for Transport Studies, Department of Civil and Environmental Engineering, Imperial College London, UK
e.candela-garza19@imperial.ac.uk

²Y. Demiris is with the Personal Robotics Laboratory, Department of Electrical and Electronic Engineering, Imperial College London, UK

bridging the gap between simulation and reality in multi-AV systems. Attempts have been made to make MARL more robust to uncertainty [5], [6], although these methods have not been tested in the real world. Therefore, in this study, we propose a method to train multi-AV driving policies in simulation and effectively transfer them to reality. For this purpose, we present a multi-agent autonomous driving gym environment that resembles the Duckietown testbed [7], which consists of small differential drive robots with two conventional wheels and a passive caster wheel. We then train different multi-agent policies in simulation and transfer them to the Duckietown using Domain Randomization [4]. We compare the method against a traditional rule-based method on a case study with 3 agents and 3 stationary vehicles which act as obstacles, and show the benefit of training policies with domain randomization when transferring them to reality.

The rest of this paper is structured as follows. Section II presents the background and related work. Section III proposes the methodology including the MARL modeling framework, the Duckietown testbed, the gym environment and the Sim-to-Real transfer mechanism. Section IV presents the main results and Section V concludes with insights to future work.

II. RELATED WORK

A. Multi-Agent Learning for Autonomous Vehicles

A taxonomy for multi-agent learning in Autonomous Driving was presented in [3], consisting of five levels, where the first level **M0** represents rule-based planning with no learning, and the last level **M5** represents agents with a high degree of forward-planning, working to optimize the *Price of Anarchy* of the overall traffic scenario [8], [9]. Most multi-agent learning paradigms find it difficult to reach level **M5**, as most traffic scenarios would be cataloged as massive multi-agent games. Fortunately, a significant amount of algorithms fit either in **M3**, allowing agents to behave and expect in return partially cooperative behavior [10], and **M4**, where a local Nash equilibrium is achieved through the grouping of agents [11].

A few studies have addressed multi-AV problems using MARL. [3] proposed an open-source MARL simulation platform, that includes several traffic scenarios with the possibility of choosing different AV controllers, environment configurations and MARL algorithms. Similarly, [2] presented MACAD-gym, an Autonomous Driving multi-agent platform based on the CARLA [12] simulator. [13] modeled the multi-agent problem as a graph and used Graph Neural Networks in combination with Deep Q Network to control lane-changing decisions in environments with multiple Connected Autonomous Vehicles (CAVs).

B. Sim-to-Real

To bridge the gap between simulation and reality, known as *reality gap*, various domain-adaptation approaches have been developed [14]. To this date, the area with the highest

amount of contributions in Sim-to-real is *robotic manipulation*. The methods that have been used include imitation learning, data augmentation and real world reinforcement learning. The latter-most case is the most difficult one to replicate due to lack of resources, safety concerns and difficulty in resetting training runs. There are simple ways of automating the reset in real world robotic manipulation [15], or to continue training efficiently without resetting [16], [17]. However, in the case of AVs, human interference is needed in order to reset the environment or to prevent catastrophic events. Therefore, most of the techniques used in robotic manipulation are impractical to adapt to environments with AVs.

For AV scenarios, [18] proposed ModEL, a modular infrastructure which considered perception, planning and control, each being trained using reinforcement learning. They used vision as the agent's main sensorial perception, and the CARLA simulator [12] during training for data augmentation and domain generalization, in order to improve overall agent robustness.

Furthermore, in the case of multi-agent systems, there are more reality gaps compared to single-agent settings. Three significant gaps have been reported in [5]: *the control architecture* gap, which relates to the tendency of simulators to synchronize the actions of all agents at each time step; *the observation* gap, which relates to the limited perception of agents in scaled-out environments; and *the communication* gap which, similar to the previous one, relates to the highly limited and inconsistent communication which in multi-agent systems. Given traditional methods of software simulation, the aforementioned gaps are non-trivial to overcome, even with a redesign of the simulation itself. The study suggests that more robust modeling of the interaction between agents would be more beneficial. [5] proposed a method called Agent Decentralized Organization (ADO), which encourages agents to share a board of information provided at certain time frames, without the necessity for the information to be complete or up to date.

Although research in Sim-to-Real for AV learning is extensive, the main focus of the literature is generalizability over diverse environments, not different actors. This can make progress in the literature slower, since the hardware is usually significantly different and not open source. Thankfully, there are attempts to standardize the research hardware, with open source sets like Duckietown [7] and DeepRacer [19], which ensure that more prior research becomes easier to reproduce consistently. However, practically no study has focused on transferring multi-agent policies for autonomous driving to the real world.

III. METHODOLOGY

A. Multi-Agent Deep Reinforcement Learning

The MARL problem can be modeled as a Decentralized Partially Observable MDP (Dec-POMDP), which can be defined by the tuple $\langle N, S, A^i, T, R, \Omega^i, O \rangle$, where N is a finite set of agents, S is the set of states, A^i is the set of actions for agent i , $T : S \times A \rightarrow S$ is the transition function,

$R : S \times A \times S \rightarrow \mathbb{R}$ is the reward function, Ω^i is the set of observations for agent i and $O : S \times A$ is the set of observation probabilities. The goal of Dec-POMDP is to find the joint optimal policy π^* that maximizes the expected return.

B. Multi-Agent Proximal Policy Optimization (MAPPO):

MAPPO [20] is an extension of the Proximal Policy Optimization algorithm to the multi-agent setting. As an on-policy method, it can be less sample efficient than off-policy methods such as MADDPG [21] and QMIX [22]. Despite this fact, MAPPO has been found to be a strong method for cooperative environments, outperforming state-of-the-art off-policy methods and achieving similar sample efficiency in practice. In addition, knowledge of prior states gets recalled through the usage of recurrent neural networks (RNNs) for the actor and critic networks.

The architecture of MAPPO consists of two separate networks: an actor and a critic network. The actor network uses the observation of each agent, while the critic network observes the whole global state. Therefore, it is directly related to the Centralized Training Decentralized Execution (CTDE) paradigm. The centralized critic can foster cooperation among agents during training, but during execution agents' policies (actors) are implemented in a decentralized way. We assume that all agents share both networks for simplicity, even though MAPPO allows for different networks for each individual agent.

C. Duckietown testbed

Duckietown [7] is a multi-robot testbed that consists of several small autonomous robots (Duckiebots) [23] that transit inside a city (Duckietown). Each Duckiebot is a differential drive vehicle equipped with wheel encoders, a monocular camera, an Inertial Measurement Unit (IMU) and a time of flight sensor.

For positioning, we use a NaturalPoint OptiTrack MoCap system, with 8 PrimeX13 cameras tracking at 120 Hz, to obtain the real-time pose of the Duckiebots. A unique asymmetric passive marker configuration is used for each car, and the system was calibrated to be accurate up to 0.5 mm. The 3D pose of each car is broadcasted to the simulator through the NatNet SDK such that position (x, y) and orientation (θ) are directly obtained after a coordinate transformation, and velocities are calculated from applying finite differences to consecutive measurements.

D. Duckietown Multi-Agent Autonomous Driving Environment (Duckie-MAAD)

For the purpose of training and evaluating multi-AV policies, we introduce a new environment called Duckietown Multi-Agent Autonomous Driving Environment (Duckie-MAAD), which is based on the Gym-Duckietown environment [24]. The original Gym-Duckietown is a single-agent lane-following environment that can be used to test different Reinforcement Learning methods. We extend this environment to the multi-agent setting and allow agents to

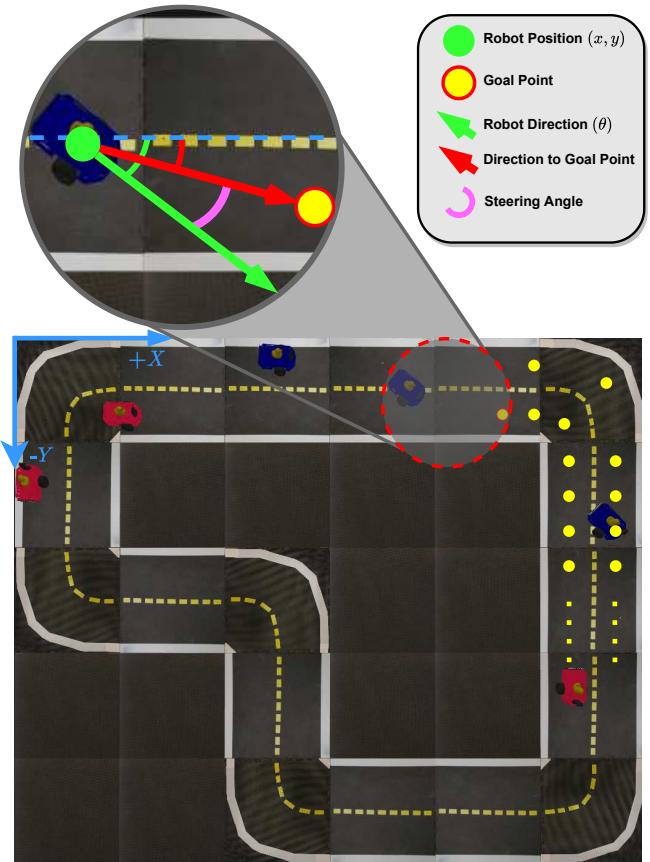


Fig. 2. Test track, used in sim and reality, for the Duckie-MAAD gym environment. Yellow circles represent the waypoints along each lane, which extend all the way around the loop. The *goal point* is determined by the Path Following function and the steering angle required to reach it is the difference between the car's direction in world coordinates and the angle between the car's position and the *goal point*.

take high-level decisions based on local observations. The agents (Duckiebots) can move around a track, following predefined lane paths and taking decisions to accelerate, brake, or change lanes. In addition, each agent can perceive nearby agents and objects in the track within a given radius. Based on perception, proximity and collision penalties can be assigned to agents, as well as a penalty for leaving the track. Although the agents follow a path predefined by waypoints, they have to learn to stay inside the track, because going too fast will take them off. Figure 2 shows a test track with 3 agents (the moving cars) for the Duckie-MAAD environment.

Upon a new environment step, the MAPPO algorithm takes in the observations of the agents and outputs a high-level action A for each car. Using this as input, a Path Following logic is implemented that determines for each car its next goal waypoint (within a pre-defined list that extends along each lane), and calculates the required linear and angular velocities (v, ω) to reach it. These are passed to the differential drive Inverse Kinematics models which produce the left and right wheel velocities (V_l, V_r) that lead to this motion. Said velocities are then either fed to the

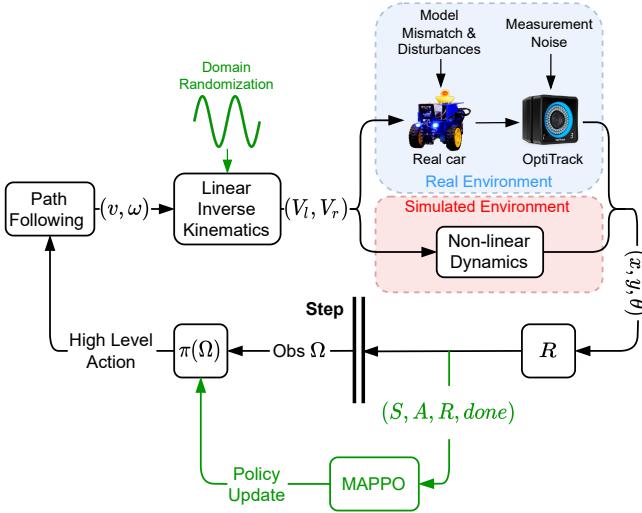


Fig. 3. Duckie-MAAD architecture. The step update loop is the following: 1) the agent takes an action following policy π ; 2) using this, the Path Following function selects the next target waypoint and calculates the linear and angular velocities required to reach it; 3) the wheel velocities that lead to such speeds are calculated using (linear) differential drive Inverse Kinematics and sent to the agents (and domain randomization is added if training MARL); 4) the resulting pose for each agent is obtained either from real life or in the simulator, together with a reward R ; If training MARL, 5) the policy is updated using MAPPO. The green elements, only when training MARL.

simulated vehicles, where the pose is updated via a Non-linear Dynamics model [25], or to the real cars, where the pose is then updated via the OptiTrack MoCap system. The new robot poses are then used to calculate the Reward R and the end of the step is reached. A detailed diagram illustrating this step update in the environment can be found in Figure 3.

We define the individual elements of the Duckietown Multi-Agent Autonomous Driving Environment as follows:

- Agents N : Each Duckiebot is treated as an individual agent. All agents are independent, have a local observation of the environment and obtain a unique reward.
- Observation Ω : The local observation of each Duckiebot is composed by the steering angle, the distance to the center of the lane, the angle with respect to a tangent to the current lane, local distances to the closest Duckiebots in the same and opposite lane, their respective longitudinal velocities, and a binary variable that shows whether the agent is off the track or not.
- Action A : A discrete action space with four high-level possible decisions – accelerate (0.25 m/s^2); brake (0.25 m/s^2); change lane; keep previous velocity (no acceleration). In each step agents can accelerate, brake, change lane or do nothing.
- Reward function R : $v - 5c - 5t - 0.5l$; where v is the agent's measure velocity (m/s), and binary variables c representing a collision, t capturing if the agent is out of the track, and l if the agent is changing lanes.

E. Simulation to reality transfer

There is a significant gap between the simulation environment and reality due to several reasons. First, the agent dynamics within the Duckie-MAAD gym environment are based on a dynamical model, which is based on various assumptions, such as a symmetrical mass distribution of the Duckiebots and that the motors operate in steady-state mode. Second, in the simulator, agent steps are performed sequentially according to the environment's internal clock, meanwhile in reality the agents may take their own step asynchronously. Third, there are certain inaccuracies related to the OptiTrack positioning system. Fourth, although all the Duckiebots share the same architecture, in practice they all behave differently due to small differences in their individual components. All of the above create a significant Sim-to-Real gap that must be addressed in order to successfully implement multi-agent policies trained in simulation.

To tackle this gap, we use domain randomization, which is a technique that consists in training a DRL model over a variety of simulated environments with randomized parameters. The objective is to train a model that can adapt to the real world environment, which is expected to be a sample of the space of environments created through the randomization procedure.

We employ uniform domain randomization of the following parameters within the Duckie-MAAD environment:

- Steering factor (*unitless*)
- Motor constant (K) ($\text{Nm}/\sqrt{\text{W}}$)
- Gain (*unitless*)
- Trim (*unitless*)
- Steering error (rad)

IV. RESULTS

To implement and validate the proposed methodology, we made use of the Duckietown Multi-Agent Autonomous Driving Environment for first training and then testing in simulation various policies. Afterward, experiments in real life were conducted following the same goals, rewards and settings. The specific environment utilized consisted of 3 agents and 3 permanently parked cars randomly spawned at the beginning of each episode throughout the test track shown in Figure 2. Agents had the goal to maximize their reward by going around the track as fast as possible while avoiding leaving the bounds of the track and colliding with other cars (either parked or other moving agents). A minimum speed of 0.1 m/s was set for all agents; and a maximum speed of 0.3 , 0.4 and 0.5 m/s for each agent, respectively. Both the simulations and real experiments were run at discrete time-steps with a frame rate of 10 Hz .

We trained three policies using the MAPPO RL method with the following parameters:

- N° episodes for training: 2000
- N° steps per episode: 400
- Learning Rate: 5×10^{-4}
- Critic Learning Rate: 5×10^{-4}
- PPO epochs: 15

- Entropy coefficient: 0.01
- Actor Network: $MLP(64, 64, 4)$
- Critic Network: $MLP(64, 64, 1)$

Each policy was trained using a different level of domain randomization (none, medium, high) with the distributions shown in Table I. All policies converged after approximately 500 episodes, as can be seen in the reward plot in Figure 4. For training, a workstation was used with an AMD Ryzen Threadripper 3990x 64-core processor and a NVIDIA RTX 3090 GPU, which resulted in a training time of approximately 3 hours for each policy.

TABLE I
DOMAIN RANDOMIZATION DISTRIBUTIONS

| Policy | Rule-based | No D.R. | Med D.R. | High D.R. |
|--------------|------------|---------|-------------|---------------|
| Steer factor | 1 | 1 | U(0.8,1.2) | U(0.5,1.5) |
| K | 27 | 27 | U(22,32) | U(14,40) |
| Gain | 1 | 1 | U(0.8,1.2) | U(0.5,1.5) |
| Trim | 0 | 0 | U(-0.1,0.1) | U(-0.15,0.15) |
| Steer error | 0 | 0 | N(0,0.1) | N(0,0.5) |

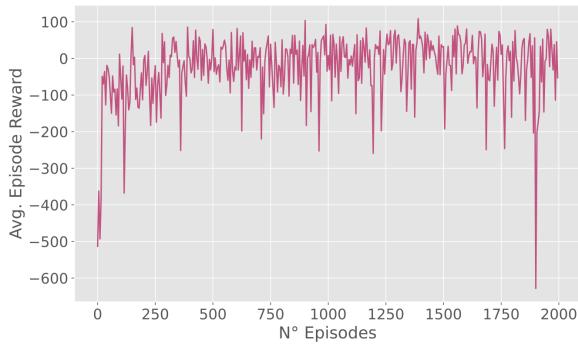


Fig. 4. Reward convergence during training of Med D.R. policy.

As a baseline, we implemented Gipps’ lane changing model [26] – which is a commonly used rule-based algorithm for describing driving behavior – and set the safe following distances according to the RSS model introduced in [27].

The four policies were tested first in the simulated environment and then in the Duckietown testbed (real life) to measure and compare their performance and transferability to real life. To achieve statistical significance and reduce the variance of the average of the metrics to be recorded, 30 runs were executed in the simulator and then another 30 in real life for each policy, with all cars being randomly spawned at the beginning of each run. Each run lasted 400 steps, which granted enough time to the agents to go around the track at least once.

The average rewards achieved by each policy in the simulated environment and real life are presented in Figure 5, where it can be seen that policies on average clearly perform worse in reality than in the simulator. This is due to the previously discussed *reality gap*, which in this case is significant mainly because of the high unreliability of

the Duckiebot robots. The rule-based policy performed the worst, as was expected, because despite following safety distances, it cannot adapt to other agents not precisely following their lanes. The policy trained with MARL and no domain randomization achieved the best rewards in simulation, but the worst in real life among the MARL-trained policies. Note that, despite slightly decreasing the simulated performance, the addition of domain randomization improves the rewards in real life. The policy with a medium level of domain randomization performed better than the one with a high level, because the latter learned to be more conservative, as can be appreciated in the following figures. On average, the med D.R. policy provides almost twice as large reward as the rule-based method. The poor performance of the untrained rule-based algorithm can be used as a proxy of the reality gap caused mainly by the unreliability and unpredictability of the real Duckiebots.

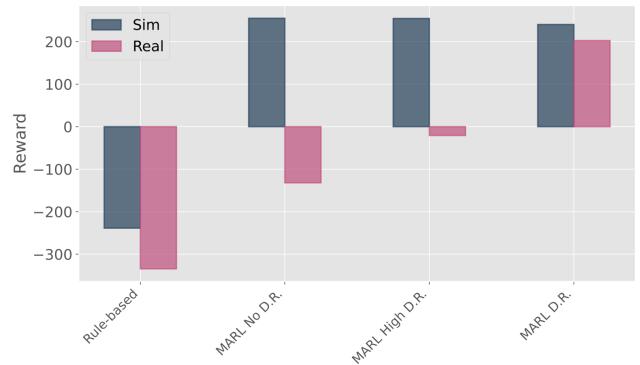


Fig. 5. Average rewards for policies in simulation and real life. MARL policies clearly outperform in real life the rule-based baseline, which can be used as a proxy of the reality gap.

To better understand the resulting rewards of the policies, we analyze the four aspects that compose the reward function: speed, staying within the designated track, collisions, and lane changes. The numerical results of both the simulation and real tests are presented in 6. The average speed is the highest for the rule-based policy, closely followed by the med D.R. one. The policy with the slowest average measured speed is the one with high D.R. – confirming that this is the most conservative policy, especially in real life. Similarly, for the times an agent goes outside the track bounds, the rule-based presents the highest number and the high D.R. the lowest; notice that this phenomenon happens mainly in real life and not in the simulated environment. For collisions, note that for all three MARL policies there are virtually no collisions in simulation, and that the policies trained with domain randomization tend to have lower collisions compared to the other policies. Finally, the average number of times an agent changes lanes is significantly larger in real life compared to simulation for policies trained with domain randomization, which can be interpreted as the agents choosing to switch lanes more to avoid the potential crashes that happen more often in real life.

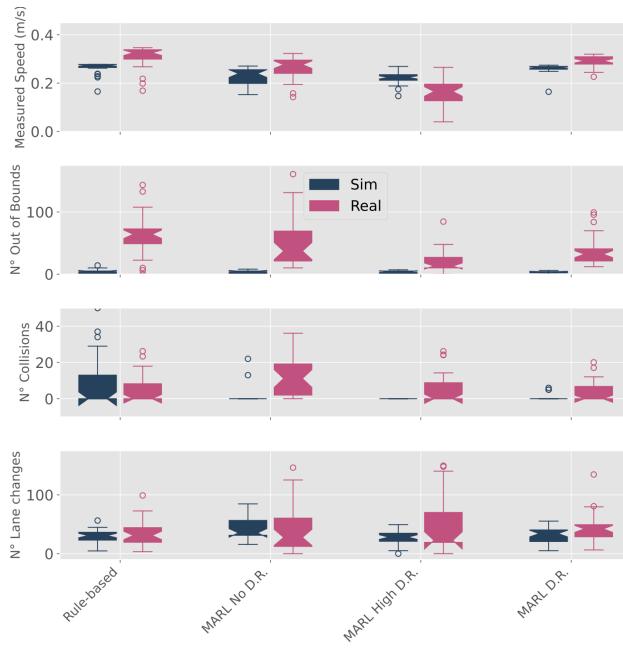


Fig. 6. Distribution of performance metrics across different policies: 1) measured speed; 2) number of times a vehicle left the track; 3) number of collisions between vehicles; 4) number of times a vehicle changed lane.
INTERPRETATION: The rule-based method tends to present the highest speed, track violations, and collisions; closely followed by the MARL policy with no D.R. In general, the policies trained with D.R. perform better in real life, this is partly because they change lanes significantly more times (compared to simulation) to avoid hazardous situations. The policy the medium D.R. (right-hand side policy) is the best one, followed by the one with high D.R. which tends to be more conservative in terms of speed – leading to the least number of track exits and collisions, especially in real life.

V. CONCLUSIONS

Autonomous Vehicles hold an enormous potential to improve safety and efficiency in various fields and applications. However, achieving reliable solutions will require solving the hard problem of coordination and collaboration of AVs. MARL is a tool that can help solve this problem, for it is an optimization-based method that can successfully allow agents to learn to collaborate by using shared observations and rewards. Nevertheless, the training of policies using MARL and other RL methods is usually heavily dependent on accurate simulation environments, which is hard to achieve due to *reality gaps*.

We present a method to train policies using MARL and to reduce the *reality gap* when transferring them to the real world via adding domain randomization during training, which we show has a significant and positive impact in real performance compared to rule-based methods or policies trained without different levels of domain randomization.

It is important to mention that despite the performance improvements observed when using domain randomization, its use presents diminishing returns as seen with the overly conservative policy, for it cannot completely close the *reality gap* without increasing the fidelity of the simulator. Additionally, the amount of domain randomization to be used

is case-specific and a theory for the selection of domain randomization remains an open question. The quantification and description of *reality gaps* presents another opportunity for future research.

ACKNOWLEDGMENT

This research was partially supported by the President's Scholarship Programme funded by Imperial College London, and the Chilean National Agency for Research and Development (ANID) through the "BECAS DOCTORADO EN EXTRANJERO" program, Grant No. 72210279.

REFERENCES

- [1] J. Stewart. (2018) Humans just can't stop rear-ending self-driving cars—let's figure out why. [Online]. Available: <https://bit.ly/3M2JED8>
- [2] P. Palanisamy, "Multi-agent connected autonomous driving using deep reinforcement learning," in *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2020, pp. 1–7.
- [3] M. Zhou, J. Luo, J. Villella, Y. Yang, D. Rusu, J. Miao, W. Zhang, M. Alban, I. Fadakar, Z. Chen *et al.*, "Smarts: Scalable multi-agent reinforcement learning training school for autonomous driving," *arXiv preprint arXiv:2010.09776*, 2020.
- [4] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," *CoRR*, vol. abs/1703.06907, 2017. [Online]. Available: <http://arxiv.org/abs/1703.06907>
- [5] Y.-H. Suh, S.-P. Woo, H. Kim, and D.-H. Park, "A sim2real framework enabling decentralized agents to execute maddpg tasks," in *Proceedings of the Workshop on Distributed Infrastructures for Deep Learning*, 2019, pp. 1–6.
- [6] K. Zhang, T. Sun, Y. Tao, S. Genc, S. Mallya, and T. Basar, "Robust multi-agent reinforcement learning with model uncertainty," *Advances in Neural Information Processing Systems*, vol. 33, pp. 10 571–10 583, 2020.
- [7] L. Paull, J. Tani, H. Ahn, J. Alonso-Mora, L. Carlone, M. Cap, Y. F. Chen, C. Choi, J. Dusek, Y. Fang *et al.*, "Duckietown: an open, inexpensive and flexible platform for autonomy education and research," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 1497–1504.
- [8] E. Koutsoupias and C. Papadimitriou, "Worst-case equilibria," in *Annual symposium on theoretical aspects of computer science*. Springer, 1999, pp. 404–413.
- [9] T. Roughgarden, *Selfish routing and the price of anarchy*. MIT press, 2005.
- [10] R. E. Wang, M. Everett, and J. P. How, "R-maddpg for partially observable environments and limited communication," *arXiv preprint arXiv:2002.06684*, 2020.
- [11] H. Zhang, W. Chen, Z. Huang, M. Li, Y. Yang, W. Zhang, and J. Wang, "Bi-level actor-critic for multi-agent coordination," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 05, 2020, pp. 7325–7332.
- [12] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," in *Conference on robot learning*. PMLR, 2017, pp. 1–16.
- [13] S. Chen, J. Dong, P. Ha, Y. Li, and S. Labi, "Graph neural network and reinforcement learning for multi-agent cooperative control of connected autonomous vehicles," *Computer-Aided Civil and Infrastructure Engineering*, vol. 36, no. 7, pp. 838–857, 2021.
- [14] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: a survey," in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2020, pp. 737–744.
- [15] B. Eysenbach, S. Gu, J. Ibarz, and S. Levine, "Leave no trace: Learning to reset for safe and autonomous reinforcement learning," *arXiv preprint arXiv:1711.06782*, 2017.
- [16] H. Zhu, J. Yu, A. Gupta, D. Shah, K. Hartikainen, A. Singh, V. Kumar, and S. Levine, "The ingredients of real-world robotic reinforcement learning," *arXiv preprint arXiv:2004.12570*, 2020.

- [17] A. Gupta, J. Yu, T. Z. Zhao, V. Kumar, A. Rovinsky, K. Xu, T. Devlin, and S. Levine, “Reset-free reinforcement learning via multi-task learning: Learning dexterous manipulation behaviors without human intervention,” in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 6664–6671.
- [18] G. Wang, H. Niu, D. Zhu, J. Hu, X. Zhan, and G. Zhou, “Model: A modularized end-to-end reinforcement learning framework for autonomous driving,” *arXiv preprint arXiv:2110.11573*, 2021.
- [19] B. Balaji, S. Mallya, S. Genc, S. Gupta, L. Dirac, V. Khare, G. Roy, T. Sun, Y. Tao, B. Townsend *et al.*, “Deepracer: Educational autonomous racing platform for experimentation with sim2real reinforcement learning,” *arXiv preprint arXiv:1911.01562*, 2019.
- [20] C. Yu, A. Velu, E. Vinitsky, Y. Wang, A. Bayen, and Y. Wu, “The surprising effectiveness of ppo in cooperative, multi-agent games,” *arXiv preprint arXiv:2103.01955*, 2021.
- [21] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch, “Multi-agent actor-critic for mixed cooperative-competitive environments,” *Advances in neural information processing systems*, vol. 30, 2017.
- [22] T. Rashid, M. Samvelyan, C. Schroeder, G. Farquhar, J. Foerster, and S. Whiteson, “Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning,” in *International Conference on Machine Learning*. PMLR, 2018, pp. 4295–4304.
- [23] Duckietown, “DuckieBot MOOC Founder’s edition datasheet,” DB21-M datasheet, Dec 2020.
- [24] M. Chevalier-Boisvert, F. Golemo, Y. Cao, B. Mehta, and L. Paull, “Duckietown environments for openai gym,” <https://github.com/duckietown/gym-duckietown>, 2018.
- [25] S. Ercan, “Modelling strategies for a mobile robot,” Master’s thesis, Swiss Federal Institute of Technology Zurich, 2019.
- [26] P. G. Gipps, “A model for the structure of lane-changing decisions,” *Transportation Research Part B: Methodological*, vol. 20, no. 5, pp. 403–414, 1986.
- [27] S. Shalev-Shwartz, S. Shammah, and A. Shashua, “On a formal model of safe and scalable self-driving cars,” *arXiv preprint arXiv:1708.06374*, 2017.