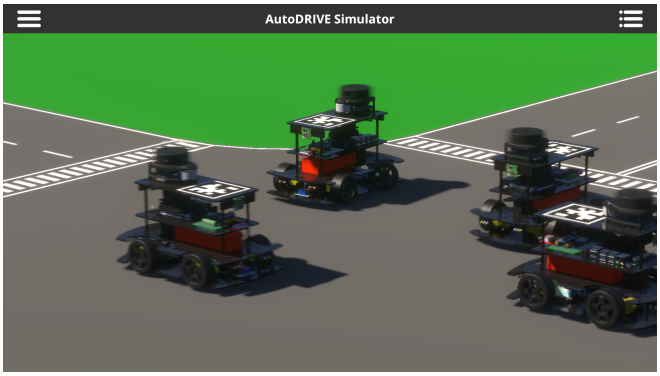
A Scalable and Parallelizable Multi-Agent Reinforcement Learning Framework for Cooperative and Competitive Autonomous Vehicles

Tanmay Samak*† and Chinmay Samak*†

***Abstract*— This work presents a modular and parallelizable multi-agent deep reinforcement learning framework capable of scaling the parallelized workloads on-demand. We first introduce AutoDRIVE Ecosystem as an enabler to develop physically accurate and graphically realistic digital twins of Nigel and F1TENTH, two scaled autonomous vehicle platforms with unique qualities and capabilities, and leverage this ecosystem to train and deploy cooperative as well as competitive multi-agent reinforcement learning policies. We first investigate an intersection traversal problem using a set of 4 cooperative vehicles (Nigel) that share limited state information with each other in single as well as multi-agent learning settings using a common policy approach. We then investigate an adversarial head-to-head autonomous racing problem using a set of 2 vehicles (F1TENTH) in a multi-agent learning setting using an individual policy approach. In either set of experiments, a decentralized learning architecture was adopted, which allowed robust training and testing of the approaches in stochastic environments, since the agents were mutually independent and exhibited asynchronous motion behavior. The problems were further aggravated by providing the agents with sparse observation spaces and requiring them to sample control commands that implicitly satisfied the imposed kinodynamic as well as safety constraints. The experimental results for both problem statements are reported in terms of quantitative metrics and qualitative remarks for training as well as deployment phases. Additionally, we discuss agent/environment parallelization techniques adopted to efficiently accelerate the MARL training in either case-studies.**

***Index Terms*— Multi-Agent Systems, Autonomous Vehicles, Deep Reinforcement Learning, Game Theory, Digital Twins, Parallel Simulations**



(a) Cooperative MARL using Nigel.

(b) Competitive MARL using F1TENTH.

Fig. 1: Multi-agent deep reinforcement learning framework using AutoDRIVE Ecosystem.

I. Introduction

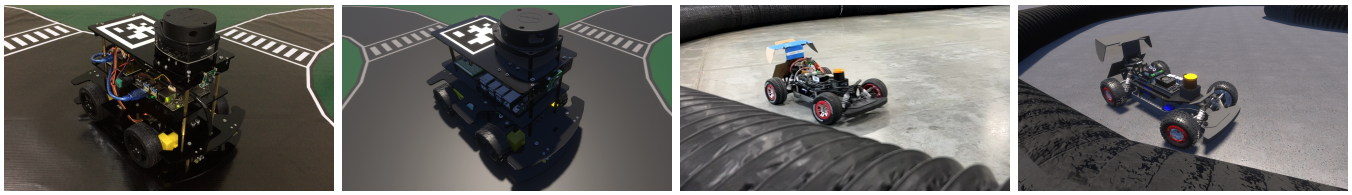
In the rapidly evolving landscape of connected and autonomous vehicles (CAVs), the pursuit of intelligent and adaptive driving systems has emerged as a formidable challenge. Multi-Agent Reinforcement Learning (MARL) stands out as a promising avenue in the quest to develop autonomous vehicles capable of navigating complex and dynamic environments, while taking into account the cooperative and/or competitive nature of interactions with their peers. Particularly, cooperative and competitive MARL represent two pivotal approaches to addressing the intricate challenges posed by multi-agent interactions in autonomous driving scenarios. While cooperative MARL encourages agents to collaborate and share information to achieve common objectives, competitive MARL introduces elements of rivalry and adversary among agents, where individual success may come at the expense of others. These paradigms offer crucial insights into the development of autonomous vehicles, and have the potential to reshape the future of transportation.

Cooperative MARL [1]–[6] fosters an environment in which autonomous vehicles cooperate to accomplish collective objectives such as optimizing traffic flow, enhancing safety, and efficiently navigating road networks. It mirrors real-world situations where vehicles must work together, such as traffic merging, intersection management, or platooning scenarios. Challenges in cooperative MARL include coordinating vehicle actions to minimize congestion, maintaining safety margins, and ensuring smooth interactions between self-interested agents.

On the other hand, competitive MARL [7]–[10] introduces a competitive edge to autonomous driving, simulating sce-

*These authors contributed equally.

†Department of Automotive Engineering, Clemson University International Center for Automotive Research (CU-ICAR), Greenville, SC 29607, USA. {tsamak, csamak}@clemson.edu



(a) Physical Nigel. (b) Virtual Nigel. (c) Physical F1TENTH. (d) Virtual F1TENTH.

Fig. 2: Creating the digital twins of Nigel and F1TENTH in AutoDRIVE Simulator.

narios such as overtaking, merging in congested traffic, or competitive racing. In this paradigm, autonomous vehicles strive to outperform their counterparts, vying for advantages while navigating complex and dynamic environments. Challenges in competitive MARL encompass strategic decision-making, opponent modeling, and adapting to aggressive driving behaviors while preserving safety standards.

As the field of MARL gains momentum within the realm of autonomous vehicles, it is crucial to comprehensively examine the implications of both cooperative and competitive approaches. In this paper, we present AutoDRIVE Ecosystem [11], [12] as an enabler to develop physically accurate and graphically realistic digital twins of scaled autonomous vehicles viz. Nigel [13] and F1TENTH [14] in Section II. We then present the problem formulation, solution approach as well as training and deployment results for a cooperative non-zero-sum use-case of intersection traversal (refer Fig. 1(a)) in Section III and a competitive zero-sum use-case of head-to-head autonomous racing (refer Fig. 1(b)) in Section IV. These sections also detail about the agent/environment parallelization techniques adopted to accelerate the MARL training. Finally we present an overall summary of our work with some concluding remarks on either case-studies.

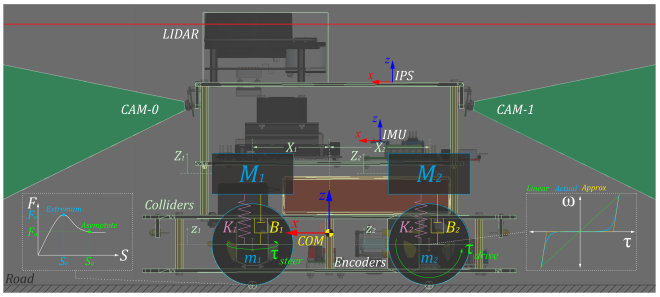
II. Digital Twin Creation

We leveraged AutoDRIVE Simulator [15], [16] to develop digital twin models of Nigel as well as F1TENTH (ref Fig. 2 and 3). It is to be noted that this work utilizes the said models in the capacity of virtual prototyping, but we seek to further investigate emerging possibilities of utilizing digital thread for harnessing the true potential of digital twins.

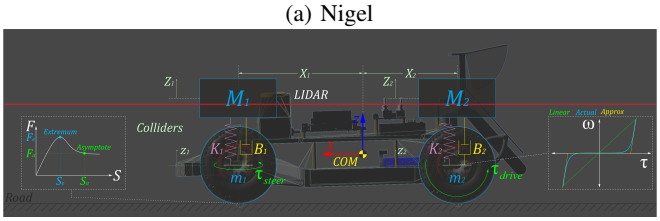
From MARL perspective, the said simulation framework was developed modularly using object-oriented programming (OOP) constructs. This allowed selectively scaling up/down the parallel agent/environment instances on demand. Additionally, the simulator took advantage of CPU multi-threading as well as GPU instancing (if available) to efficiently parallelize various simulation objects and processes, with cross-platform support.

A. Vehicle Dynamics Models

The vehicle model is a combination of a rigid body and a collection of sprung masses ${}^{i}M$, where the total mass of the rigid body is defined as $M = \sum {}^{i}M$. The rigid body's center of mass, $X_{COM} = \frac{\sum {}^{i}M * {}^{i}X}{\sum {}^{i}M}$, connects these representations, with ${}^{i}X$ representing the coordinates of the sprung masses.



(a) Nigel



(b) F1TENTH

Fig. 3: Simulation of vehicle dynamics, sensors and actuators for Nigel and F1TENTH digital twins.

The suspension force acting on each sprung mass is computed as ${}^{i}M * {}^{i}\ddot{Z} + {}^{i}B * ({}^{i}\dot{Z} - {}^{i}\dot{z}) + {}^{i}K * ({}^{i}Z - {}^{i}z)$, where ${}^{i}Z$ and ${}^{i}z$ are the displacements of sprung and unsprung masses, and ${}^{i}B$ and ${}^{i}K$ are the damping and spring coefficients of the i-th suspension, respectively.

The vehicle's wheels are also treated as rigid bodies with mass m, subject to gravitational and suspension forces: ${}^{i}m * {}^{i}\ddot{z} + {}^{i}B * ({}^{i}\dot{z} - {}^{i}\dot{Z}) + {}^{i}K * ({}^{i}z - {}^{i}Z)$.

Tire forces are computed based on the friction curve for each tire, represented as $\begin{cases} {}^{i}F_{t_x} = F({}^{i}S_x) \\ {}^{i}F_{t_y} = F({}^{i}S_y) \end{cases}$, where ${}^{i}S_x$ and ${}^{i}S_y$ are the longitudinal and lateral slips of the i-th tire, respectively. The friction curve is approximated using a two-piece cubic spline, defined as $F(S) = \begin{cases} f_0(S); & S_0 \leq S < S_e \\ f_1(S); & S_e \leq S < S_a \end{cases}$, where $f_k(S) = a_k * S^3 + b_k * S^2 + c_k * S + d_k$ is a cubic polynomial function. The first segment of the spline ranges from zero (S_0, F_0) to an extremum point (S_e, F_e), while the second segment ranges from the extremum point (S_e, F_e) to an asymptote point (S_a, F_a).

The tire slip is influenced by factors including tire stiffness ${}^{i}C_{\alpha}$, steering angle δ, wheel speeds ${}^{i}\omega$, suspension forces ${}^{i}F_s$, and rigid-body momentum ${}^{i}P$. These factors impact the longitudinal and lateral components of the vehicle's linear velocity. The longitudinal slip ${}^{i}S_x$ of the i-th tire is calculated

by comparing the longitudinal components of the surface velocity of the i-th wheel (i.e., longitudinal linear velocity of the vehicle) v_x with the angular velocity ${}^i\omega$ of the i-th wheel: ${}^iS_x = \frac{{}^ir*{}^i\omega - v_x}{v_x}$. The lateral slip ${}^iS_y$ depends on the tire's slip angle α and is determined by comparing the longitudinal v_x (forward velocity) and lateral v_y (side-slip velocity) components of the vehicle's linear velocity: ${}^iS_y = \tan(\alpha) = \frac{v_y}{|v_x|}$.

B. Sensor Models

The simulated vehicles can be equipped with the physically accurate interoceptive as well as exteroceptive sensing modalities. Specifically, the throttle (τ) and steering (δ) sensors are simulated using a straightforward feedback loop.

Incremental encoders are simulated by measuring the rotation of the rear wheels (i.e., the output shaft of driving actuators): ${}^iN_{ticks} = {}^iPPR * {}^iGR * {}^iN_{rev}$, where ${}^iN_{ticks}$ represents the ticks measured by the i-th encoder, ${}^iPPR$ is the base resolution (pulses per revolution) of the i-th encoder, ${}^iGR$ is the gear ratio of the i-th motor, and ${}^iN_{rev}$ represents the number of revolutions of the output shaft of the i-th motor.

The Inertial Positioning System (IPS) and Inertial Measurement Unit (IMU) are simulated based on temporally-coherent rigid-body transform updates of the vehicle $\{v\}$ with respect to the world $\{w\}$: ${}^w\mathbf{T}_v = \left[\begin{array}{c|c} \mathbf{R}_{3\times3} & \mathbf{t}_{3\times1} \\ \hline \mathbf{0}_{1\times3} & 1 \end{array}\right] \in SE(3)$. The IPS provides 3-DOF positional coordinates $\{x, y, z\}$ of the vehicle, while the IMU supplies linear accelerations $\{a_x, a_y, a_z\}$, angular velocities $\{\omega_x, \omega_y, \omega_z\}$, and 3-DOF orientation data for the vehicle, either as Euler angles $\{\phi_x, \theta_y, \psi_z\}$ or as a quaternion $\{q_0, q_1, q_2, q_3\}$.

The LIDAR simulation employs iterative ray-casting $\texttt{raycast}\{{}^w\mathbf{T}_l,\ \vec{\mathbf{R}},\ r_{max}\}$ for each angle $\theta \in [\theta_{min} : \theta_{res} : \theta_{max}]$ at an approximate update rate of 7 Hz. Here, ${}^w\mathbf{T}_l = {}^w\mathbf{T}_v * {}^v\mathbf{T}_l \in SE(3)$ represents the relative transformation of the LIDAR $\{l\}$ with respect to the vehicle $\{v\}$ and the world $\{w\}$, $\vec{\mathbf{R}} = [r_{max} * sin(\theta)\ \ r_{min} * cos(\theta)\ \ 0]^T$ defines the direction vector of each ray-cast R, where $r_{min} = 0.15$ m and $r_{max} = 12$ m denote the minimum and maximum linear ranges of the LIDAR, $\theta_{min} = 0°$ and $\theta_{max} = 360°$ set the minimum and maximum angular ranges of the LIDAR, and $\theta_{res} = 1°$ represents the angular resolution of the LIDAR. The laser scan ranges are determined by checking ray-cast hits and then applying a threshold to the minimum linear range of the LIDAR, calculated as $\texttt{ranges[i]} = \begin{cases} \texttt{hit.dist} & \text{if } \texttt{ray[i].hit} \text{ and } \texttt{hit.dist} \geq r_{min} \\ \infty & \text{otherwise} \end{cases}$, where `ray.hit` is a Boolean flag indicating whether a ray-cast hits any colliders in the scene, and $\texttt{hit.dist} = \sqrt{(x_{hit} - x_{ray})^2 + (y_{hit} - y_{ray})^2 + (z_{hit} - z_{ray})^2}$ calculates the Euclidean distance from the ray-cast source $\{x_{ray}, y_{ray}, z_{ray}\}$ to the hit point $\{x_{hit}, y_{hit}, z_{hit}\}$.

The simulated physical cameras are parameterized by their focal length ($f = 3.04$ mm), sensor size ($\{s_x, s_y\} = \{3.68, 2.76\}$ mm), target resolution (default = 720p), as well as the distances to the near and far clipping planes ($N = 0.01$ m and $F = 1000$ m). The viewport rendering pipeline for the simulated cameras operates in three stages. First, the camera view matrix $\mathbf{V} \in SE(3)$ is computed by obtaining the relative homogeneous transform of the camera $\{c\}$ with respect to the world $\{w\}$: $\mathbf{V} = \begin{bmatrix} r_{00} & r_{01} & r_{02} & t_0 \\ r_{10} & r_{11} & r_{12} & t_1 \\ r_{20} & r_{21} & r_{22} & t_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$, where r_{ij} and t_i denote the rotational and translational components, respectively. Next, the camera projection matrix $\mathbf{P} \in \mathbb{R}^{4\times4}$ is calculated to project world coordinates into image space coordinates: $\mathbf{P} = \begin{bmatrix} \frac{2*N}{R-L} & 0 & \frac{R+L}{R-L} & 0 \\ 0 & \frac{2*N}{T-B} & \frac{T+B}{T-B} & 0 \\ 0 & 0 & -\frac{F+N}{F-N} & -\frac{2*F*N}{F-N} \\ 0 & 0 & -1 & 0 \end{bmatrix}$, where N and F represent the distances to the near and far clipping planes of the camera, and L, R, T, and B denote the left, right, top, and bottom offsets of the sensor. The camera parameters $\{f, s_x, s_y\}$ are related to the terms of the projection matrix as follows: $f = \frac{2*N}{R-L}$, $a = \frac{s_y}{s_x}$, and $\frac{f}{a} = \frac{2*N}{T-B}$. The perspective projection from the simulated camera's viewport is given as $\mathbf{C} = \mathbf{P} * \mathbf{V} * \mathbf{W}$, where $\mathbf{C} = [x_c\ \ y_c\ \ z_c\ \ w_c]^T$ represents image space coordinates, and $\mathbf{W} = [x_w\ \ y_w\ \ z_w\ \ w_w]^T$ represents world coordinates. Finally, this camera projection is transformed into normalized device coordinates (NDC) by performing perspective division (i.e., dividing throughout by w_c), leading to a viewport projection achieved by scaling and shifting the result and then utilizing the rasterization process of the graphics API (e.g., DirectX for Windows, Metal for macOS, and Vulkan for Linux). Additionally, a post-processing step simulates lens and film effects, such as lens distortion, depth of field, exposure, ambient occlusion, contact shadows, bloom, motion blur, film grain, chromatic aberration, etc.

C. Actuator Models

The vehicle's motion is controlled by driving and steering actuators, with response delays and saturation limits matched to their real-world counterparts by tuning their torque profiles and actuation limits.

The driving actuators propel the rear/front/all wheels by applying a torque, calculated as ${}^i\tau_{drive} = {}^iI_w * {}^i\dot{\omega}_w$, where ${}^iI_w = \frac{1}{2} * {}^im_w * {}^ir_w{}^2$ represents the moment of inertia, ${}^i\dot{\omega}_w$ is the angular acceleration, ${}^im_w$ is the mass, and ${}^ir_w$ is the radius of the i-th wheel. Additionally, the driving actuators simulate holding torque by applying an idle motor torque equivalent to the braking torque, i.e., ${}^i\tau_{idle} = {}^i\tau_{brake}$.

The front wheels are steered using a steering actuator that generates a torque proportional to the required angular acceleration, given by $\tau_{steer} = I_{steer} * \dot{\omega}_{steer}$. The individual turning angles, δ_l and δ_r, for the left and right wheels, respectively, are computed based on the commanded steering angle δ, utilizing the Ackermann steering geometry defined by the wheelbase l and track width w, as follows:

$$\begin{cases} \delta_l = \tan^{-1}\left(\frac{2*l*\tan(\delta)}{2*l+w*\tan(\delta)}\right) \\ \delta_r = \tan^{-1}\left(\frac{2*l*\tan(\delta)}{2*l-w*\tan(\delta)}\right) \end{cases}$$

D. Environment Models

At each time step, the simulator conducts mesh-mesh interference detection and computes contact forces, frictional forces, momentum transfer, as well as linear and angular drag acting on all rigid bodies. Simulated environments can be established using one of the following approaches:

- *AutoDRIVE IDK:* Custom scenarios and maps can be crafted by utilizing the modular and adaptable Infrastructure Development Kit (IDK). This kit provides the flexibility to configure terrain modules, road networks, obstruction modules, and traffic elements. Specifically, the intersection traversal scenario was developed using AutoDRIVE IDK.
- *Plug-In Scenarios:* AutoDRIVE Simulator supports third-party tools, such as RoadRunner [17], and open standards like OpenSCENARIO [18] and OpenDRIVE [19]). This allows users to incorporate a diverse range of plugins, packages, and assets in several standard formats for creating or customizing driving scenarios. Particularly, the autonomous racing scenario was created based on the binary occupancy grid map of a real-world F1TENTH racetrack called "Proto" using a third-party 3D modelling software, which was then imported into AutoDRIVE Simulator and post-processed with physical as well as graphical enhancements to make it "sim-ready".
- *Unity Terrain Integration:* Since the AutoDRIVE Simulator is built atop the Unity [20] game engine, it seamlessly supports scenario design and development through Unity Terrain [21]. Users have the option to define terrain meshes, textures, heightmaps, vegetation, skyboxes, wind effects, and more, allowing design of both on-road and off-road scenarios. This option is well-suited for modelling full-scale environments.

III. Cooperative Multi-Agent Scenario

Inspired by [6], this use-case encompassed both single-agent and multi-agent learning scenarios, where each agent's objective was autonomous traversal of a 4-lane, 4-way intersection without collisions or lane boundary violations. Each vehicle possessed intrinsic state information and received limited state information about its peers; no external sensing modalities were employed. A deep neural network policy was independently trained for each scenario, guiding the agents through the intersection safely. The entire case-study was developed using an integrated ML framework [22] within AutoDRIVE Simulator.

A. Problem Formulation

In *single-agent learning scenario*, only the ego vehicle learned to traverse the intersection, while peer vehicles were controlled at different velocities using a randomized heuristic. Peer vehicles shared their states with the ego vehicle via V2V communication. All vehicles were reset together, making this scenario quite deterministic.

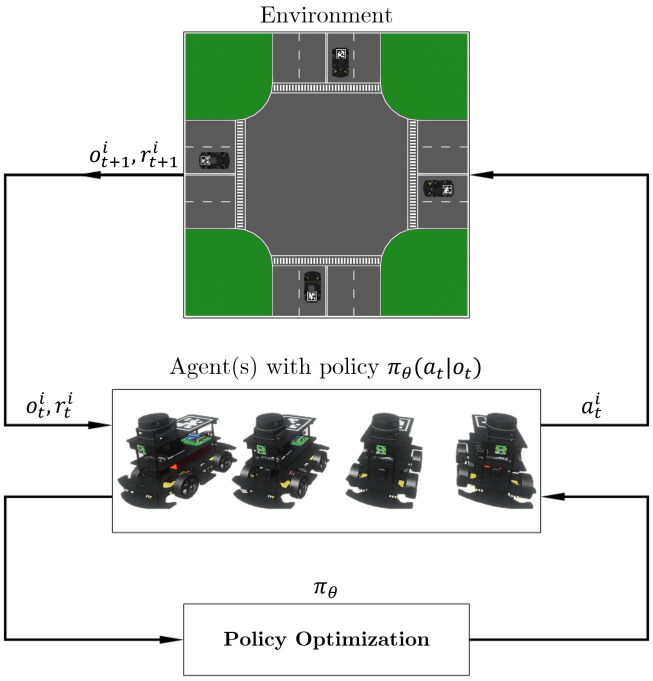


Fig. 4: Deep reinforcement learning architecture for collaborative intersection traversal scenario.

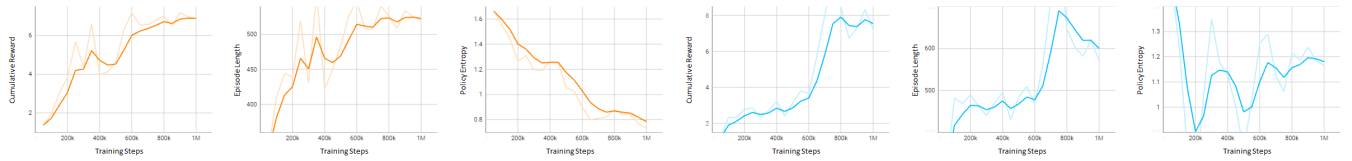
In *multi-agent learning scenario*, all vehicles learned to traverse the intersection simultaneously in a decentralized manner. Vehicles shared their states with each other via V2V communication and were reset independently, resulting in a highly stochastic scenario.

In both the scenarios, the challenge revolved around autonomous navigation in an unknown environment. The exact structure/map of the environment was not known to any agent. Consequently, this decision-making problem was framed as a Partially Observable Markov Decision Process (POMDP), which captured hidden state information through environmental observations.

B. State Space

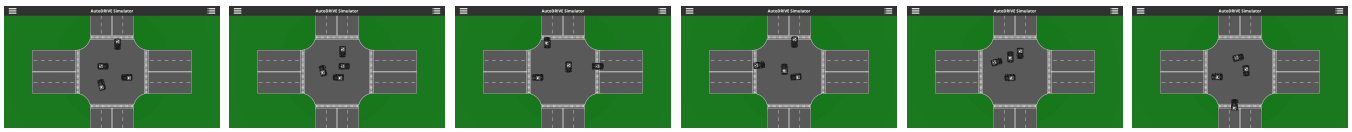
As previously discussed, the state space S for intersection traversal problem could be divided into observable $s^o \subset S$ and hidden $s^h \subset S$ components. The observable component included the vehicle's 2D pose and velocity, denoted as $s^o_t = [p_x, p_y, \psi, v]\, t \in \mathbb{R}^4$. The hidden component encompassed the agent's goal coordinates, represented as $st^h = [g_x, g_y]_t \in \mathbb{R}^2$. Thus, each agent could observe the pose and velocity of its peers (via V2V communication) but kept its own goal location hidden from others. Consequently, the complete state space of an agent participating in this problem was a vector containing all observable and hidden states:

$$s_t = \left[s^o_t, s^h_t\right] \tag{1}$$



(a) (b) (c) (d) (e) (f)

Fig. 5: Training results for (a)-(c) single-agent and (d)-(f) multi-agent intersection traversal scenarios: (a) and (d) denote cumulative reward, (b) and (e) denote episode length, while (c) and (f) denote policy entropy w.r.t. training steps.



(a) (b) (c) (d) (e) (f)

Fig. 6: Deployment results for (a)-(c) single-agent and (d)-(f) multi-agent intersection traversal scenarios: (a) and (d) denote first frozen snapshot, (b) and (e) denote second frozen snapshot, while (c) and (f) denote third frozen snapshot.

C. Observation Space

Based on the state space defined in Equation 1, each agent employed an appropriate subset of its sensor suite to collect observations (as per Equation 2). This included Inertial Positioning System (IPS) for positional coordinates $[p_x, p_y]\, t \in \mathbb{R}^2$, Inertial Measurement Unit (IMU) for yaw $\psi t \in \mathbb{R}^1$, and incremental encoders for estimating vehicle velocity $v_t \in \mathbb{R}^1$. Each agent i (where $0 < i < N$) was provided with an observation vector of the form:

$$o_t^i = \left[g^i, \tilde{p}^i, \tilde{\psi}^i, \tilde{v}^i\right]_t \in \mathbb{R}^{2+4(N-1)} \tag{2}$$

This formulation allowed $g_t^i = \left[g_x^i - p_x^i, g_y^i - p_y^i\right] t \in \mathbb{R}^2$ to represent the ego agent's goal location relative to itself, $\tilde{p}t^i = \left[p_x^j - p_x^i, p_y^j - p_y^i\right] t \in \mathbb{R}^{2(N-1)}$ to denote the position of every peer agent relative to the ego agent, $\tilde{\psi}t^i = \psi_t^j - \psi_t^i \in \mathbb{R}^{N-1}$ to express the yaw of every peer agent relative to the ego agent, and $\tilde{v}t^i = vt^j \in \mathbb{R}^{N-1}$ to indicate the velocity of every peer agent. Here, i represented the ego agent, and $j \in [0, N-1]$ represented every other (peer) agent in the scene, with a total of N agents.

D. Action Space

The vehicles were designed as non-holonomic rear-wheel-drive models featuring an Ackermann steering mechanism. As a result, the complete action space of an agent comprised longitudinal (throttle/brake) and lateral (steering) motion control commands. For longitudinal control, the throttle command τ_t was discretized as $\tau_t \in \{0.5, 1.0\}$, which allowed the agents to slow down if necessary. The steering command δ_t was discretized as $\delta_t \in \{-1, 0, 1\}$, which allowed the agents to steer left, straight or right to safely traverse the intersection, as expressed in Equation 3:

$$a_t^i = \left[\tau_t^i, \delta_t^i\right] \in \mathbb{R}^2 \tag{3}$$

E. Reward Function

The extrinsic reward function (as shown in Equation 4) was designed to reward each agent with $r_{goal} = +1$ for successfully traversing the intersection. Alternatively, it penalized agents proportionally to their distance from the goal, represented as $k_p * \left\|g_t^i\right\|_2$, for collisions or lane boundary violations. Finally, each agent was rewarded inversely proportional to its distance from the goal, so as to negotiate the sparse-reward problem. The reward (k_r) and penalty (k_p) constants were set to 0.01 and 0.425 respectively, resulting in a maximum reward/penalty of 1 per time-step.

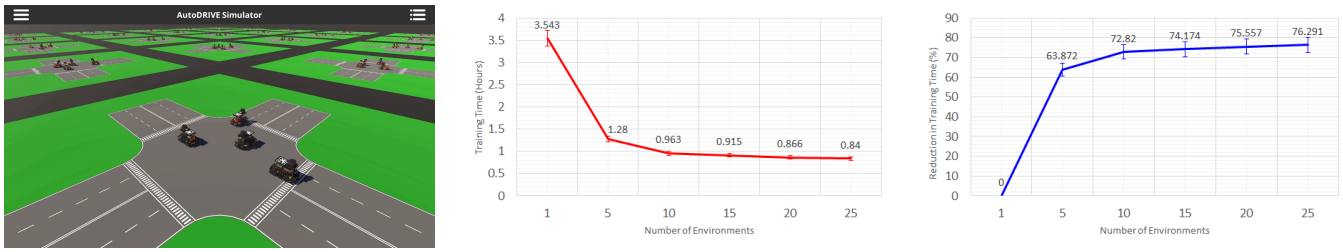
$$r_t^i = \begin{cases} r_{goal} & \text{if safe traversal} \\ -k_p * \left\|g_t^i\right\|_2 & \text{if traffic violation} \\ k_r * (0.001 + \left\|g_t^i\right\|_2)^{-1} & \text{otherwise} \end{cases} \tag{4}$$

This encouraged agents to get closer to their respective goals, reducing penalties and ultimately leading to a positive reward, r_{goal}. This approach not only expedited convergence but also restricted reward hacking.

F. Optimization Problem

The task of intersection traversal, with collision avoidance and lane-keeping constraints, was addressed through the extrinsic reward function described in Equation 4. This function motivated each individual agent to maximize the expected future discounted reward (as per Equation 5) by learning a policy $\pi_\theta\left(a_t|o_t\right)$. Over time, this policy transitioned into the optimal policy π^*.

$$\underset{\pi_\theta(a_t|o_t)}{\operatorname{argmax}} \quad \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right] \tag{5}$$



(a) (b) (c)

Fig. 7: Computational results for training intersection traversal scenario through environment parallelization: (a) depicts a snapshot of 25 environments training in parallel, (b) denotes the training time for different levels of environment parallelization, and (c) denotes the percentage reduction in training time for different levels of environment parallelization.

TABLE I: Training Configuration for Cooperative MARL

PARAMETER	VALUE
Hyperparameters	
Neural network architecture (FCNN)	3 × {128, Swish [24]}
Batch size	64
Buffer size	1024
Learning rate (α)	3e-4
Learning rate schedule	Linear
Entropy regularization strength (β)	0.001
Policy update hyperparameter (ϵ)	0.2
Regularization parameter (λ)	0.98
Epochs	3
Maximum steps	1e6
Extrinsic Reward	
Discount factor (${}^{e}\gamma$)	0.99
Strength	1.0

G. Training

At each time step t, each parallelized agent i collected an observation vector o_t^i and an extrinsic reward r_t^i. Based on these inputs, it took an action a_t^i determined by the policy π_θ, which was recursively updated to maximize the expected future discounted reward (refer Fig. 4).

This use-case employed a fully connected neural network (FCNN) as a function approximator for $\pi_\theta\left(a_t|o_t\right)$. The network had $\mathbb{R}^{14}$ inputs, $\mathbb{R}^1$ outputs, and three hidden layers with 128 neural units each. The policy parameters $\theta \in \mathbb{R}^d$ were defined in terms of the network's parameters. The policy was trained to predict steering commands directly based on collected observations, utilizing the proximal policy optimization (PPO) algorithm [23].

Table I hosts the detailed training configuration adopted for the cooperative MARL scenario, after careful hyperparameter tuning. The parameter values mentioned were arrived at by analyzing the agent(s) behaviors to satisfy the intended objective (collaborative safe intersection traversal) qualitatively, while also ensuring a stable learning process through the analysis of several training metrics. Fig. 5 depicts key training metrics used to analyze the learning process. A general indication of "good" training is that the cumulative reward is maximized and then saturated, the episode length is adequate (longer duration implies agents wandering off in the environment, while very short duration may be indicative of agents colliding/overstepping lane bounds), and the policy entropy (i.e., randomness) has decreased steadily as the training progressed. It is to be noted that the predominant cause for the difference in trends of training metrics for single and multi-agent scenarios is the higher stochasticity of the multi-agent scenario, which is especially evident from the policy entropy.

H. Simulation Parallelization

This case-study adopted environment parallelization for accelerating the RL training. Particularly, we analyzed the effect of parallelizing an intersection-traversal environment of 4 agents from a single instance up to 25 such environments (i.e., 100 agents) training in parallel. All the training experiments were carried out on a laptop PC with 12th Gen Intel Core i9-12900H 2.50 GHz CPU, NVIDIA GeForce RTX 3080 Ti GPU and 32.0 GB (31.7 GB usable) RAM. The software framework employed for training the policies included PyTorch 1.7.1 installed over CUDA 11.0. It is to be noted that the simulator (parallelized or otherwise) and the RL training took place on the same machine.

Fig. 7(a) depicts a snapshot of 25 intersection-traversal environments training in parallel. This clearly differentiates the architecture of environment parallelization from agent/actor parallelization (refer Fig. 11(a)). As observed in Fig. 7(b)-(c) the reduction in training time was quite non-linear since the simulation workload increased with increasing parallelization. As a result, we can notice the curves nearly saturate after a point, which is subject to change with a different hardware/software configuration. Additionally, it should be noted that parallelization beyond a certain point can hurt, wherein the increased simulation workload may slow down the training so much that parallel policy optimization can no longer accelerate it.

I. Deployment

The trained policies were deployed onto the simulated vehicles, separately for both single-agent and multi-agent scenarios. As previously mentioned, the single-agent scenario was less stochastic, and the ego vehicle could safely traverse the intersection in most cases. In contrast, the multi-agent scenario was highly stochastic, resulting in a significantly lower success rate, especially with all vehicles navigating the intersection simultaneously.

Fig. 6(a)-(c) present three key stages of the single-agent intersection traversal scenario. The first stage depicts the ego vehicle approaching the conflict zone, where it could potentially collide with peer vehicles. The second stage shows the vehicle executing a left turn to avoid collisions. Finally, the third stage illustrates the vehicle performing a subtle right turn to reach its goal. Fig. 6(d)-(f) display three critical stages of the multi-agent intersection traversal scenario. In the first frame, vehicles 1 and 4 successfully avoid collision. The second frame showcases vehicle 1 finding a gap between vehicles 2 and 3 to reach its goal. In the third frame, vehicles 2 and 3 evade collision, while vehicle 4 approaches its goal, and vehicle 1 is re-spawned.

IV. Competitive Multi-Agent Scenario

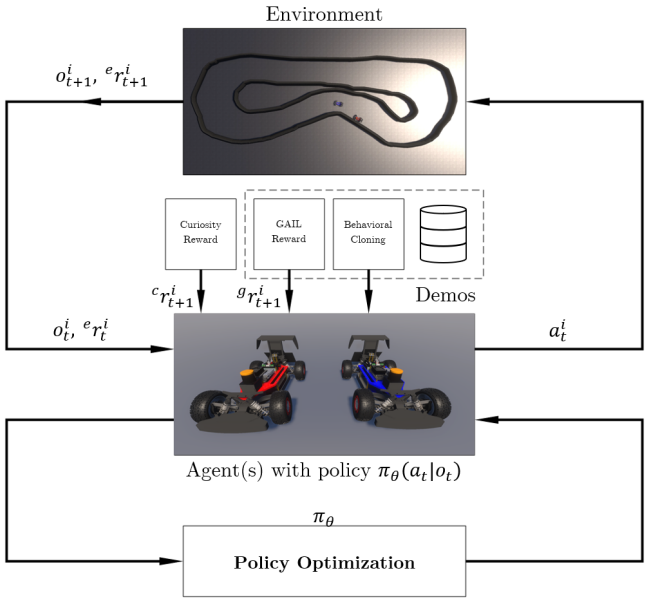


Fig. 8: Demonstration-guided deep reinforcement learning architecture for competitive head-to-head racing scenario.

Inspired by [9], this use-case encompassed a multi-agent learning scenario, where each agent's objective was minimizing its lap time without colliding with the track or its opponent. Each vehicle possessed intrinsic state information and sparse LIDAR measurements; no state information was shared among the competing vehicles. The entire use-case was developed using the same integrated ML framework mentioned in Section III.

A. Problem Formulation

This case-study addressed the problem of autonomous racing in an unknown environment. The exact structure/map of the environment was not known to any agent. Consequently, this decision-making problem was also framed as a POMDP, which captured hidden state information through environmental observations.

We adopted an equally-weighted hybrid imitation-reinforcement learning architecture to progressively inculcate autonomous driving and racing behaviors into the agents. Consequently, we recorded 5 laps worth of independent demonstration datasets for each agent by manually driving the vehicles in sub-optimal trajectories within a single-agent setting. We hypothesised that such a hybrid learning architecture would guide the agents' exploration, thereby reducing training time significantly.

B. Observation Space

At each time step t, the agent collected a vectorized observation, as shown in Equation 6. These observations were obtained using velocity estimation and exteroceptive ranging modalities mounted on the virtual vehicle(s):

$$o^i_t = \left[v^i_t, m^i_t\right] \in \mathbb{R}^{28} \tag{6}$$

Here, $v^i_t \in \mathbb{R}^1$ represents the forward velocity of i-th agent, and $m^i_t = \left[{}^1m^i_t, {}^2m^i_t, \cdots, {}^{27}m^i_t\right] \in \mathbb{R}^{27}$ is the measurement vector providing 27 range readings up to 10 meters. These readings are uniformly distributed over 270° around each side of the heading vector, spaced 10° apart. These observations were then input into a deep neural network policy denoted as π_θ, where $\theta \in \mathbb{R}^d$ denotes the policy parameters.

C. Action Space

The policy mapped the observations o_t directly to an appropriate action a_t, as expressed in Equation 7:

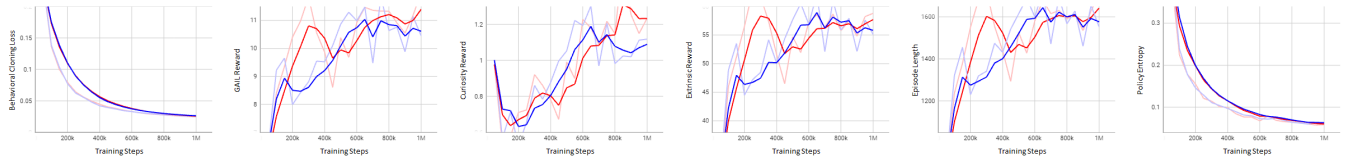
$$a^i_t = \left[\tau^i_t, \delta^i_t\right] \in \mathbb{R}^2 \tag{7}$$

Here, $\tau^i_t \in \{0.1, 0.5, 1.0\}$ represent the discrete throttle commands at 10%, 50% and 100% PWM duty cycles for torque limited (85.6 N-m) drive actuators, and $\delta^i_t \in \{-1, 0, 1\}$ represent the discrete steering commands for left, straight, and right turns, respectively.

D. Reward Function

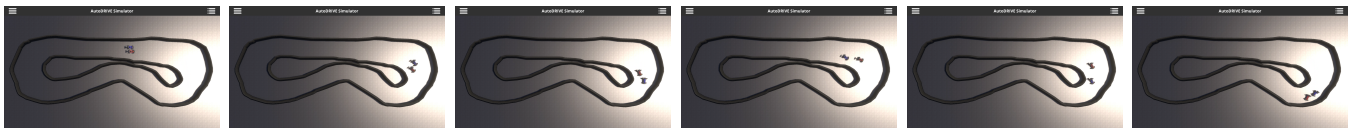
The policy π_θ was optimized based on the following signals:

- **Behavioral Cloning:** This was the core imitation learning algorithm, which updated the policy in a supervised fashion with respect to the recorded demonstrations. The behavioral cloning [25] update was carried out every once in a while, mutually exclusive of the reinforcement learning update.
- **GAIL Reward:** The generative adversarial imitation learning (GAIL) reward [26] $^g r_t$ ensured that the agent optimized its actions safely and ethically by rewarding proportional to the closeness of new observation-action pairs to those from the recorded demonstrations. This allowed the agent to retain its autonomous driving ability throughout the training process.
- **Curiosity Reward:** The curiosity reward [27] $^c r_t$ promoted exploration by rewarding proportional to the difference between predicted and actual encoded observations. This ensured that the agent effectively explored its environment, even though the extrinsic reward was sparse.



(a) (b) (c) (d) (e) (f)

Fig. 9: Training results for multi-agent autonomous racing: (a) denotes BC loss, (b) denotes GAIL reward, (c) denotes curiosity reward, (d) denotes extrinsic reward, (e) denotes episode length, and (f) denotes policy entropy w.r.t. training steps.



(a) (b) (c) (d) (e) (f)

Fig. 10: Deployment results for multi-agent autonomous racing: (a)-(c) denote three frozen snapshots of a block-block-overtake sequence, and (d)-(f) denote three frozen snapshots of a let-pass-and-overtake sequence.

- **Extrinsic Reward:** In the context of reinforcement learning, the objectives of lap time reduction and motion constraints were handled using a novel extrinsic reward function ${}^{e}r_t$ (as detailed in Equation 8), which guided the agent towards optimal behavior using the proximal policy optimization (PPO) algorithm [23]. The agent received a reward of $r_{checkpoint} = +0.01$ for passing each of the 19 checkpoints c_i, where $i \in [\mathrm{A}, \mathrm{B}, \cdots, \mathrm{S}]$ on the racetrack, $r_{lap} = +0.1$ upon completing a lap, $r_{best\,lap} = +0.7$ upon achieving a new best lap time, and a penalty of $r_{collision} = -1$ for colliding with the track bounds or peer agent (in which case both agents were penalized equally). Additionally, the agent received continuous rewards proportional to its velocity v_t, encouraging it to optimize its trajectory spatio-temporally.

$$ {}^{e}r_t^i = \begin{cases} r_{collision} & \text{if collision occurs} \\ r_{checkpoint} & \text{if checkpoint is passed} \\ r_{lap} & \text{if completed lap} \\ r_{best\,lap} & \text{if new best lap time is achieved} \\ 0.01 * v_t^i & \text{otherwise} \end{cases} \tag{8} $$

E. Optimization Problem

The task of head-to-head autonomous racing using demonstration guided reinforcement learning converged to maximizing the expected future discounted reward (as per Equation 9) by learning an optimal policy $\pi_\theta^*(a_t|o_t)$. The reward in this case was a sum of GAIL, curiosity and extrinsic rewards (i.e., $r_t^i = {}^{g}r_t^i + {}^{c}r_t^i + {}^{e}r_t^i$).

$$ \underset{\pi_\theta^i(a_t|o_t)}{\operatorname{argmax}} \quad \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t^i\right] \tag{9} $$

F. Training

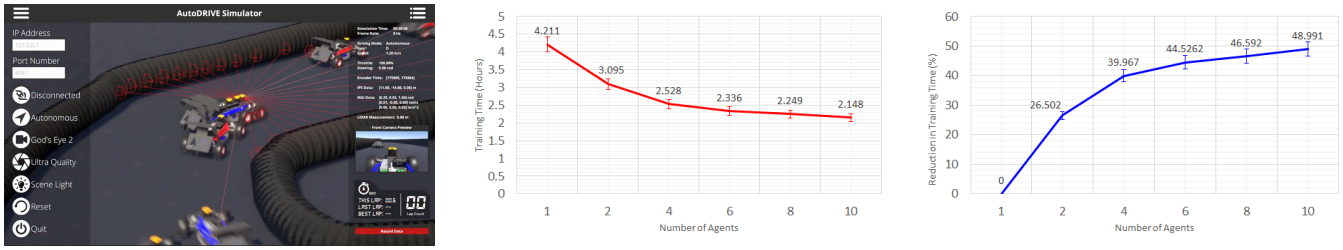
The policy π_θ was optimized to maximize the expected future discounted reward (GAIL, curiosity and extrinsic rewards), while also minimizing the BC loss (refer Fig. 8).

This use-case also employed a fully connected neural network (FCNN) as a function approximator for $\pi_\theta(a_t|o_t)$. The network had $\mathbb{R}^{28}$ inputs, $\mathbb{R}^2$ outputs, and three hidden layers with 128 neural units each. The policy parameters $\theta \in \mathbb{R}^d$ were defined in terms of the network's parameters. The policy was trained to predict throttle and steering commands directly based on collected observations, utilizing the proximal policy optimization (PPO) algorithm [23].

Table II hosts the detailed training configuration adopted for the competitive MARL scenario, after careful hyper-parameter tuning. The parameter values mentioned were arrived at by analyzing the agent(s) behaviors to satisfy the intended objective (competitive head-to-head autonomous racing) qualitatively, while also ensuring a stable learning process through the analysis of several training metrics. Fig. 9 depicts key training metrics used to analyze the learning process. A general indication of "good" training is that the behavioral cloning loss has decayed smoothly, the GAIL, curiosity and extrinsic rewards are maximized and then saturated, the episode length is adequate (longer duration implies agents driving slowly, while very short duration may be indicative of agents colliding without lap completion), and the policy entropy (i.e., randomness) has decreased steadily as the training progressed. It is to be noted that the non-zero offset in behavioral cloning loss indicates that the agents have not over-fit to the demonstrations; rather, they have explored the state space quite well to maximize the extrinsic reward by adopting aggressive "racing" behaviors.

G. Simulation Parallelization

This case-study adopted agent parallelization for accelerating the RL training. Particularly, we analyzed the effect of parallelizing the 2-agent adversarial racing family from a single instance up to 10 such families training in parallel within the same environment. All the training experiments were carried out on a laptop PC with the same hardware and software configuration as mentioned in Section III-H.



(a) (b) (c)

Fig. 11: Computational results for training intersection traversal scenario through agent parallelization: (a) depicts a snapshot of 10×2 agents training in parallel, (b) denotes the training time for different levels of agent parallelization, and (c) denotes the percentage reduction in training time for different levels of agent parallelization.

TABLE II: Training Configuration for Competitive MARL

PARAMETER	VALUE
Hyperparameters	
Neural network architecture (FCNN)	3 × {128, Swish [24]}
Batch size	64
Buffer size	1024
Learning rate (α)	3e-4
Learning rate schedule	Linear
Entropy regularization strength (β)	0.001
Policy update hyperparameter (ϵ)	0.2
Regularization parameter (λ)	0.98
Epochs	3
Maximum steps	1e6
Behavioral Cloning	
Strength	0.5
GAIL Reward	
Discount factor (${}^{g}\gamma$)	0.99
Strength	0.01
Encoding size	128
Learning rate (${}^{g}\alpha$)	3e-4
Curiosity Reward	
Discount factor (${}^{c}\gamma$)	0.99
Strength	0.02
Encoding size	256
Learning rate (${}^{c}\alpha$)	3e-4
Extrinsic Reward	
Discount factor (${}^{e}\gamma$)	0.99
Strength	1.0

The core agent parallelization architecture involved the creation and isolation of simulation objects in different virtual *"layers"*. For instance, the environment was simulated on the *"default"* layer, whereas each family of the multi-agent system was simulated on a different layer, with the agents of the same family being simulated on the same layer. This allowed selective interactions and collision-checks between specific layers of the simulation. Parallelization of perception modalities involved appropriately instantiating and isolating the interoceptive sensors (e.g., encoders, IPS, IMU, etc.) on respective layers. Exteroceptive sensors such as cameras were effectively parallelized by enabling culling mask bits only for specific layers, whereas LIDARs were parallelized by returning raycast hits only for specific layers.

Fig. 11(a) depicts a snapshot of 10×2 agents training in parallel within the same environment. Notice that the agents can collide, perceive or interact only with their *"true"* opponents, and none of the *"parallel"* agents. This clearly differentiates the architecture of agent/actor parallelization from environment parallelization (refer Fig. 7(a)). As observed in Fig. 11(b)-(c) the reduction in training time was quite non-linear in this case as well. This could be primarily attributed to the increase in simulation workload with increasing parallelization. As a result, training time started saturating after a point, which is subject to change with a different hardware/software configuration. Additionally, as mentioned earlier, parallelization beyond a certain point can hurt, wherein the increased simulation workload may slow down the training so much that parallel policy optimization can no longer accelerate it.

H. Deployment

The trained policies were deployed onto the respective simulated vehicles, which were made to race head-to-head on the same track with a phase-shifted initialization (as in real F1TENTH competitions).

Fig. 10(a)-(c) present three snapshots of a block-block-overtake sequence, wherein the red agent kept blocking the blue agent throughout the straight, but the blue agent took a wider turn with higher velocity and took advantage of its under-steer characteristic to cut in front of the red agent and overtake it. Fig. 10(d)-(f) display three snapshots of a let-pass-and-overtake sequence, wherein the blue agent found a gap between the red agent and inside edge of the track and opportunistically overtook it. However, due to its under-steering characteristic, it went wider in the corner, thereby allowing the red agent to overtake it and re-claim the leading position.

V. Conclusion

This work presented a scalable and parallelizable multi-agent reinforcement learning framework for imbibing cooperative and competitive behaviors within autonomous vehicles. We discussed representative case-studies for each behavior type in detail and analyzed the training and deployment results. Here, we deliberately formulated the two problems with distinct observation spaces and reward functions, but more importantly, we also varied the learning architecture from vanilla RL to demonstration-guided RL. Finally, we also analyzed the effect of agent/environment parallelization on the training time and noted its non-linear nature.

Since this work leveraged the real2sim approach to develop physically and graphically realistic digital twins for training sim2real-worthy policies, a natural extension would be to analyze the sim2real [28] transfer of these trained policies. Furthermore, analyzing MARL training across different types of computing platforms and configurations could provide better insight from a computing perspective. Additionally, innovation from the MARL front could be in the form of applying physics-informed RL to multi-agent settings for modeling agent(s) state-transitions, formulating physics-guided reward functions, applying safety guarantees, etc. Finally, extending/generalizing this approach to mid and full-scale autonomous vehicles with different perception and actuation configurations could unlock various operational design domains for MARL.

REFERENCES

[1] S. H. Semnani, H. Liu, M. Everett, A. de Ruiter, and J. P. How, "Multi-agent Motion Planning for Dense and Dynamic Environments via Deep Reinforcement Learning," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3221–3226, 2020.

[2] P. Long, T. Fan, X. Liao, W. Liu, H. Zhang, and J. Pan, "Towards Optimally Decentralized Multi-Robot Collision Avoidance via Deep Reinforcement Learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 6252–6259.

[3] S. Aradi, "Survey of Deep Reinforcement Learning for Motion Planning of Autonomous Vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 2, pp. 740–759, 2022.

[4] D. Wang, H. Deng, and Z. Pan, "MRCDRL: Multi-Robot Coordination with Deep Reinforcement Learning," *Neurocomputing*, vol. 406, pp. 68–76, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231220305932

[5] X. Zhou, P. Wu, H. Zhang, W. Guo, and Y. Liu, "Learn to Navigate: Cooperative Path Planning for Unmanned Surface Vehicles Using Deep Reinforcement Learning," *IEEE Access*, vol. 7, pp. 165 262–165 278, 2019.

[6] K. Sivanathan, B. K. Vinayagam, T. Samak, and C. Samak, "Decentralized Motion Planning for Multi-Robot Navigation using Deep Reinforcement Learning," in *2020 3rd International Conference on Intelligent Sustainable Systems (ICISS)*, 2020, pp. 709–716. [Online]. Available: https://doi.org/10.1109/ICISS49785.2020.9316033

[7] F. Fuchs, Y. Song, E. Kaufmann, D. Scaramuzza, and P. Dürr, "Super-Human Performance in Gran Turismo Sport Using Deep Reinforcement Learning," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4257–4264, 2021.

[8] Y. Song, H. Lin, E. Kaufmann, P. Dürr, and D. Scaramuzza, "Autonomous Overtaking in Gran Turismo Sport Using Curriculum Reinforcement Learning," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 9403–9409.

[9] C. V. Samak, T. V. Samak, and S. Kandhasamy, "Autonomous Racing using a Hybrid Imitation-Reinforcement Learning Architecture," 2021. [Online]. Available: https://arxiv.org/abs/2110.05437

[10] J. Betz, H. Zheng, A. Liniger, U. Rosolia, P. Karle, M. Behl, V. Krovi, and R. Mangharam, "Autonomous Vehicles on the Edge: A Survey on Autonomous Vehicle Racing," *IEEE Open Journal of Intelligent Transportation Systems*, vol. 3, pp. 458–488, 2022.

[11] T. Samak, C. Samak, S. Kandhasamy, V. Krovi, and M. Xie, "AutoDRIVE: A Comprehensive, Flexible and Integrated Digital Twin Ecosystem for Autonomous Driving Research & Education," *Robotics*, vol. 12, no. 3, p. 77, May 2023. [Online]. Available: http://dx.doi.org/10.3390/robotics12030077

[12] T. V. Samak and C. V. Samak, "AutoDRIVE - Technical Report," 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2211.08475

[13] C. Samak, T. Samak, and V. Krovi, "Towards Mechatronics Approach of System Design, Verification and Validation for Autonomous Vehicles," in *2023 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, 2023, pp. 1208–1213. [Online]. Available: https://doi.org/10.1109/AIM46323.2023.10196233

[14] M. O'Kelly, V. Sukhil, H. Abbas, J. Harkins, C. Kao, Y. V. Pant, R. Mangharam, D. Agarwal, M. Behl, P. Burgio, and M. Bertogna. (2019) F1/10: An Open-Source Autonomous Cyber-Physical Platform. [Online]. Available: https://arxiv.org/abs/1901.08567

[15] T. V. Samak, C. V. Samak, and M. Xie, "AutoDRIVE Simulator: A Simulator for Scaled Autonomous Vehicle Research and Education," in *2021 2nd International Conference on Control, Robotics and Intelligent System*, ser. CCRIS'21. New York, NY, USA: Association for Computing Machinery, 2021, p. 1–5. [Online]. Available: https://doi.org/10.1145/3483845.3483846

[16] T. V. Samak and C. V. Samak, "AutoDRIVE Simulator - Technical Report," 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2211.07022

[17] Mathworks Inc., "RoadRunner," 2021. [Online]. Available: https://www.mathworks.com/products/roadrunner.html

[18] Association for Standardization of Automation and Measuring Systems (ASAM), "OpenSCENARIO," 2021. [Online]. Available: https://www.asam.net/standards/detail/openscenario

[19] ——, "OpenDRIVE," 2021. [Online]. Available: https://www.asam.net/standards/detail/opendrive

[20] Unity Technologies, "Unity," 2021. [Online]. Available: https://unity.com

[21] ——, "Unity Terrain," 2021. [Online]. Available: https://docs.unity3d.com/Manual/script-Terrain.html

[22] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar, and D. Lange, "Unity: A General Platform for Intelligent Agents," 2018. [Online]. Available: https://arxiv.org/abs/1809.02627

[23] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017. [Online]. Available: https://arxiv.org/abs/1707.06347

[24] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for Activation Functions," 2017.

[25] M. Bain and C. Sammut, "A Framework for Behavioural Cloning," in *Machine Intelligence 15*, 1995.

[26] J. Ho and S. Ermon, "Generative Adversarial Imitation Learning," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, ser. NIPS'16. Red Hook, NY, USA: Curran Associates Inc., 2016, p. 4572–4580.

[27] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-Driven Exploration by Self-Supervised Prediction," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17. JMLR.org, 2017, p. 2778–2787.

[28] C. V. Samak, T. V. Samak, and V. Krovi, "Towards Sim2Real Transfer of Autonomy Algorithms using AutoDRIVE Ecosystem," 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2307.13272