2021 IEEE Intelligent Transportation Systems Conference (ITSC)
Indianapolis, USA. September 19-21, 2021

Multi-layer Control Architecture for Unsignalized Intersection Management via Nonlinear MPC and Deep Reinforcement Learning

Ahmed H. Hamouda [2,3], Dalia M. Mahfouz [2,3], Catherine M. Elias[2,3] and Omar M. Shehata[1,2,3]

***Abstract*— In this study, a multi-layer control architecture for unsignalized intersection management is proposed. The architecture is composed of two layers. The low-level layer deals with a single decoupled vehicle model controlled depending on a dynamical-based Nonlinear Model Predictive Control (NMPC) algorithm. The high-level layer holds the decision-making module which is controlled via a centralized-trained/ decentralized-executed Multi Agent Twin Delayed Deep Deterministic Policy Gradient (MATD3) algorithm. In the proposed architecture, the high-level decision making layer generates a reference velocity for each vehicle to pass the unsignalized intersection without collisions. The NMPC is utilized to ensure that each vehicle follows the provided trajectory, taking into account the vehicle's acceleration profile to attain passengers' comfort. The proposed approach is trained and executed in a custom-built multi-agent intersection environment simulated using CARLA simulator. Results indicate effective training as the MATD3 was able to encounter the suitable policy to avoid the intersection collisions with a success percentage of 100%.**

***Index Terms*— Architecture; Intersection Management; Reinforcement Learning; Nonlinear MPC; Vehicle Dynamics; CARLA.**

I. Introduction and State of Art

OVER the last seven decades, the world has witnessed a jump in the number of vehicles on roads. This can be justified by the rapid development of the social economy and the automobile industry [1]. This led to a surge in multiple problems such as fuel consumption, traffic congestion, road accidents, and others. In the United States (US), it is recorded that the average American driver loses about 99 hours yearly on the road. This of course has a huge financial impact costing more than $88 billion nationally due to traffic jams [2]. Moreover, this reflects the number of deaths caused by road accidents which reached about $1.35 million annually [3]. While investigating the different causes, it is found that about 27% of the traffic casualties recorded in the US occur at intersections [4]. Therefore, researchers started to invest their resources to find a solution for the massive aforementioned issues relying upon the development in the Intelligent Transportation System (ITS) field, making use of the upcoming autonomous revolution. One of the proposed solutions is replacing the traffic light system with an unsignalized controlled intersection.

*This work was not supported by any organization

[1]Mechatronics Department - Faculty of Engineering - Ain Shams University Cairo, Egypt
[2]Multi-Robot Systems (MRS) Research Group, Cairo, Egypt `mrs.research.lab@gmail.com`
[3]Mechatronics Department - Faculty of Engineering and Materials Science - German University in Cairo, Egypt
`ahmed.hatem3h@gmail.com, dalia.mahfouz@ieee.org, catherine.elias@ieee.org, omar.shehata@ieee.org`

This improves traffic mobility [5], by replacing the human driver with an Automated Driving System (ADS). This system is capable of performing complex communication with the surrounding vehicles approaching the intersection. This allows the vehicles to pass the intersection with higher efficiency and safer manner using the vehicle-to-vehicle (V2V) communication methodologies.

The implementation of this decision making level in intersections requires the existence of an agent that is capable of gathering information about the approaching vehicles and coordinate them. Investigating the different control architectures to perform the high-level decision layer, it is found that Reinforcement Learning (RL) possesses a remarkable place in the literature [6]–[8]. RL is characterized by its ability to carry out optimal control strategies applied in different scenarios while handling the uncertainties in nonlinear dynamical models [9]–[11].

Moreover in the intersection management field, it is heavily used especially in the decision making layer. In 2018, Tram et.al [7] proposed a two-layer control architecture between automated and human-driven vehicles, where Deep Recurrent Q Network (DRQN) is used in the higher layer while the low-level planning layer is based on a Sliding Mode Control (SMC) algorithm. This algorithm proved its effectiveness in avoiding collision by 98% in the intersection. A year after, the authors extended their work by investigating the performance of the Model Predictive Control (MPC) in the planning layer which succeeded in shortening the training time for the traffic scenarios compared to the SMC agent [8]. A limitation presented in this work was testing only the two-way intersections on a simple self-made environment where vehicles were assumed to be point-masses.

Lin et.al. in 2019 tackled the problem of Adaptive Cruise Control (ACC). They tested the Deep RL controller on a kinematic vehicle modeled as point-mass. The kinematics model could not be generalized to more realistic cases and it was recommended to design Deep RL controllers while considering the vehicle dynamics for more complex driving scenarios [12]. Additionally; to the best of our knowledge, very few studies investigated the RL usage in the multi AVs in unsignalized intersections. In 2017, a centralized fine-grained approach was proposed by Mirzaei et.al. controlling the acceleration of autonomous vehicles in a grid street plan [13]. Despite the fact that the intersection problems can be represented as a single centralized agent with a large state and action space utilizing the single-agent RL methods, they are immensely hard to scale. This results in an action space that increases exponentially with the number of agents.

2021 IEEE International Intelligent Transportation Systems Conference (ITSC) | 978-1-7281-9142-3/21/$31.00 ©2021 IEEE | DOI: 10.1109/ITSC48978.2021.9565126

Based on the conducted literature review, there exists a gap in the investigation of the multi-layer intersection management control architecture including the hybridization of both Deep RL algorithms and the conventional controllers. The use of a Multi-Agent RL with centralized training and decentralized execution approach in the autonomous intersection management problem is still unexplored. Furthermore, due to the high uncertainties existing in the system especially for multi-AVs, the vehicle dynamics can no longer be neglected.

Thus, the contribution of this paper is to investigate the aforementioned research gaps through addressing the four-way intersection management for multiple autonomous vehicle scenarios. The targeted scenario is solved by introducing a multi-layer control architecture utilizing a Multi-Agent Twin Delayed Deep Deterministic Policy Gradient (MATD3) algorithm and the Non-linear MPC (NMPC). In this architecture, a centralized-trained/ decentralized-executed MATD3 algorithm is implemented in the high-level decision making layer where it generates a reference velocity for each vehicle to pass the unsignalized intersection without crashing. Meanwhile, the NMPC is utilized in the low-level control layer to control the dynamical vehicle model. This layer is responsible for maintaining the generated reference velocity, considering the vehicle's acceleration to achieve the passengers' comfort. This architecture is simulated while training and execution using CARLA simulator with its realistic vehicle physics to bridge the sim-to-real gap.

II. Preliminary

A. Vehicle Dynamics

The designed multi-layer intersection management control architecture considers the dynamical model of the vehicle to minimize the error between the desired and actual performance due to the uncertainties in the system.

A three Degrees of Freedom (DoF) decoupled lateral and longitudinal nonlinear model is used for the vehicle dynamics. In this model, the state vector $\bar{q}$ and control input $\bar{u}$ for the vehicle are given by: $\bar{q} = [X, Y, V_x, V_y, \psi, \dot{\psi}]$, $\bar{u} = [\alpha_{th}, \alpha_{br}, \delta]$ where X and Y denote the vehicle's global position, V_x and V_y the vehicle's velocities in the longitudinal and lateral direction, respectively, ψ and $\dot{\psi}$ the vehicle's heading and its derivative, α_{th}, α_{br}, and δ are the vehicle's throttle, brakes, and steering respectively.

1) Lateral Dynamics: The lateral dynamics of the vehicle is simplified to a dynamic bicycle model based on [14] as illustrated in Figure 1. The dynamic model utilizes a linear tire model to calculate the lateral forces acting on the tires due to the interaction between the tire and the ground. The lateral dynamics can be defined by Equations (1a) and (1b).

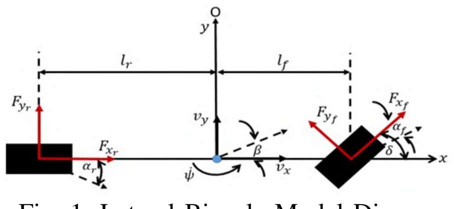


Fig. 1: Lateral Bicycle Model Diagram

The dynamics are computed in terms of ℓ_f and ℓ_r which are the distances from the vehicle's center of mass to the front and rear axles respectively, the cornering stiffness of the two front and two rear tires $C_{\alpha f}$ and $C_{\alpha r}$ respectively, vehicle mass m, and moment of inertia about the z-axis I_z.

$$\ddot{\psi} = -\frac{\ell_f C_{\alpha f} - \ell_r C_{\alpha r}}{I_z V_x} V_y - \frac{\ell_f^2 C_{\alpha f} + \ell_r^2 C_{\alpha r}}{I_z V_x} \dot{\psi} + \frac{\ell_f C_{\alpha f}}{I_z} \delta \tag{1a}$$

$$\dot{V}_y = -\frac{C_{\alpha f} + C_{\alpha v}}{m V_x} V_y + (-V_x - \frac{C_{\alpha f} \ell_f - C_{\alpha r} \ell_r}{m V_x}) \dot{\psi} + \frac{C_{\alpha f}}{m} \delta \tag{1b}$$

2) Longitudinal Dynamics: The longitudinal model is based on a one-wheel vehicle as in [15], where the force outputted by the engine, the frictional forces, and the aerodynamic forces acting on the vehicle in the longitudinal direction are considered as shown in Figure 2 and mathematically described by Equation (2).

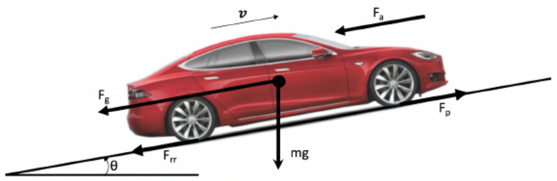


Fig. 2: Vehicle Longitudinal Forces on an Inclined Road

$$\dot{V}_x = \frac{R^2}{mR^2 + I_w} \left(R_g \alpha_{th} T_{e_{\max}}(\omega_e) - \alpha_{br} T_{b_{\max}} - (F_a + F_{rr}) R \right) \tag{2}$$

where m is the vehicle's mass, R is the wheel's radius, and I_w is its moment of inertia. R_g is the gearbox ratio, $T_{b_{\max}}$ is the maximum brake torque and ω_e is the engine speed while $T_{e_{\max}}$ is the maximum engine torque at a specific ω_e. The aerodynamic force is calculated as $F_a = \frac{1}{2} \rho C_d V_x^2$, with the air density ρ and the drag coefficient C_d. The rolling resistance force is computed as $F_{rr} = C_r mg \cos\theta$, with the rolling resistance coefficient C_r.

The vehicle dynamics is then discretized with a sampling time T_s using Euler discretization method to obtain $x(k+1) = x(k) + T_s f^c(x(k), u(k))$, where $f^c(x^c(t), u^c(t))$ is the set of the first-order differential equations shown in Equations (1a), (1b), and (2). This discrete model is used as the prediction model of the NMPC.

B. Nonlinear Model Predictive Control (NMPC)

MPC -known as Receding Horizon- is a control approach that uses the plant model in order to make future predictions about the output along a prediction horizon $\boldsymbol{N}$. Afterward, it solves an online constrained optimization problem to minimize a certain cost function by producing a sequence of optimized control inputs to the system.

1) Objective Function: It is a function that needs to be minimized by a solver. It can be formulated in terms of error terms, calculated along the horizon $\boldsymbol{N}$. Equation (3) describes the objective function used in the study, where x and x_{ref} indicate the actual and reference state variables, u and u_{ref} are the actual and reference control input.

In this objective function, the term $\left(x_{k+1} - x_{ref_{k+1}}\right)$ penalizes the difference between the predicted state and its reference, while $(u_k - u_{ref_k})$ penalizes the deviation of the future control inputs from the steady-state input, and $(u_k - u_{k-1})$ penalizes the change in the control input between two successive time steps. These terms are squared and multiplied by the weight matrices Q, R, and D, respectively.

$$\begin{aligned}\ell(x,u) = &\sum_{k=1}^{N}\left(x_{k+1} - x_{ref_{k+1}}\right)^T Q\left(x_{k+1} - x_{ref_{k+1}}\right)\\ &+ \left(u_k - u_{ref_k}\right)^T R\left(u_k - u_{ref_k}\right)\\ &+ \sum_{k=2}^{N}\left(u_k - u_{k-1}\right)^T D\left(u_k - u_{k-1}\right)\end{aligned} \tag{3}$$

2) Optimization Problem: The MPC problem can be stated as an Optimal Control Problem (OCP) formulated by:

$$\begin{aligned}&\underset{\mathbf{u}}{\text{minimize}} && \ell\left(\mathbf{x_u}(k), \mathbf{u}(k)\right) \\ &\text{subject to} && :\mathbf{x_u}(k+1) = \mathbf{f}\left(\mathbf{x_u}(k), \mathbf{u}(k)\right) \quad , \quad \mathbf{x_u}(0) = \mathbf{x}_0\\ &&& \mathbf{u}(k) \in U \quad , \quad \forall k \in [1, N]\\ &&& \mathbf{x_u}(k) \in X \quad , \quad \forall k \in [1, N+1]\end{aligned} \tag{4}$$

In order to solve the OCP, it needs to be converted to a Nonlinear Programming Problem (NLP). Nonlinear programming is the standard formulation in numerical optimization with a general form $\min_{\mathbf{w}} \Phi(\mathbf{w})$, subjected to:
$\mathbf{g}_1(\mathbf{w}) \leq 0 \quad , \quad \mathbf{g}_2(\mathbf{w}) = 0.$

where $\Phi(\mathbf{w})$ is the objective function, $\mathbf{w}$ are the manipulated variables, $\mathbf{g}_1(\mathbf{w})$ and $\mathbf{g}_2(\mathbf{w})$ are inequality constraints and equality constraints respectively. In order to express the OCP as an NLP, multiple shooting technique is used. In the multiple shooting technique, both the input and the state variables are considered the manipulated variables $w = [u_1, \ldots, u_n, x_1, \ldots, x_{N+1}]$. However, an additional path constraint is applied at each optimization step $x(k+1) - f(x(k), u(k)) = 0$. These equality constraints are to obtain the zero difference between the actual and predicted states. The NLP is then solved using the Interior Point OPTimizer (IPOPT) solver in CasADi/MATLAB [16].

C. Reinforcement Learning (RL)

RL is classified as a branch of machine learning in which an agent learns from experience gained while exploring an environment formulated as a Markov Decision Process (MDP). At each time step, an agent observes an environment state s and then applies action a according to a policy π.

The environment updates its state to s' and sends the new state back to the agent along with a reward r. The return is described as the total discounted reward $R_t = \sum_{i=t}^{T} \gamma^{i-t} r\left(s_i, a_i\right)$, where γ is a discount factor and T is the time horizon. This reward will be used by the agent in order to update its policy π and keep looping till reaching the optimal policy π^* that maximizes the expected return $J(\phi) = \mathbb{E}_{s_i \sim p_\pi, a_i \sim \pi}\left[R_0\right]$.

1) Twin Delayed Deep Deterministic Policy Gradient (TD3): It is introduced in [17] based on the classic Deep Deterministic Policy Gradient (DDPG) [18] network which is categorized as an actor-critic method. In actor-critic methods, the policy -known as the actor- is used to approximate the optimal policy deterministically and is updated through the deterministic policy gradient algorithm:

$$\nabla_\phi J(\phi) = \mathbb{E}_{s \sim p_\pi}\left[\nabla_a Q^\pi(s,a)\big|_{a=\pi(s)} \nabla_\phi \pi_\phi(s)\right] \tag{5}$$

where $Q^\pi(s,a) = \mathbb{E}_{s_i \sim p_\pi, a_i \sim \pi}\left[R_t \mid s, a\right]$, is known as the value function or the critic, which is the expected return when performing action a in state s and following the policy π. The critic is updated by using temporal difference learning with a second separate target network $Q_{\theta'}(s,a)$ to maintain a fixed objective y over multiple updates:

$$y = r + \gamma Q_{\theta'}\left(s', a'\right), \quad a' \sim \pi_{\phi'}\left(s'\right) \tag{6}$$

where the target actor network $\pi_{\phi'}$ selects the actions. The weights of the target network are updated by some proportion τ at each time step $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$.

One of the DDPG problems which are addressed in TD3 is that the learned Q-function begins to dramatically overestimate Q-values [17]. This leads to the policy breaking as it exploits the errors in the Q-function. The TD3 method solves this by introducing three tricks. Firstly, TD3 uses two Q-networks $Q_{\theta 1}$, $Q_{\theta 2}$, along with two target networks. The Q-functions are updated with the target $y = r_t + \gamma min_{1,2} Q_{\theta'_i}(s', a')$, while updating the policy with $Q_{\theta 1}$.

Secondly, delayed policy updates are introduced where the policy π and target network parameters θ'_π, θ'_Q are updated once every d critic updates. This results in value estimate with lower variance which consequently results in a better policy. The final trick is to smooth the target policy by adding a small amount of clipped noise to the next action of the critic target $a' = \mu_{\theta'_\pi}(s') + \epsilon$, where ϵ is a clipped Gaussian noise $\epsilon = clip(N(0, \alpha), -c, c)$, where c is a tunable parameter.

2) Multi-Agent TD3 (MATD3) [19]: It extends TD3 to the multi-agent domain as the extension done from DDPG to Multi-agent DDPG (MADDPG) [20]. MATD3 uses a centralized training with a decentralized execution framework. During execution, each agent has its own separate actor/policy. However, while training, a single criteria is formulated sharing two centralized critics $Q^\pi_{\theta_{1,2}}$. The combined criteria has access to the past actions, observations, and rewards of all agents. Moreover, it takes the action and state of agent i as input and outputs the Q-value for agent i. A shared replay buffer containing the transitions of all agents is used as well.

III. Multi-layer Unsignalized Intersection Architecture

In this work, a multi-layer architecture is built for the unsignalized intersection system. This architecture is composed of two layers. The first one is used for making decision for the vehicle in the intersection. This layer is built based on the Multi-Agent RL (MARL). The second layer is responsible for controlling the single vehicle motion based on the made decision by the higher layer. This architecture is simulated and tested in CARLA [21] simulator. Figure 3 shows the overall connection between the different layers, simulation environment, and the vehicles represented as agents in the intersection scenario.

A. Simulation Environment

In order to simulate the architecture, CARLA -autonomous driving simulator- is used as the simulation environment.

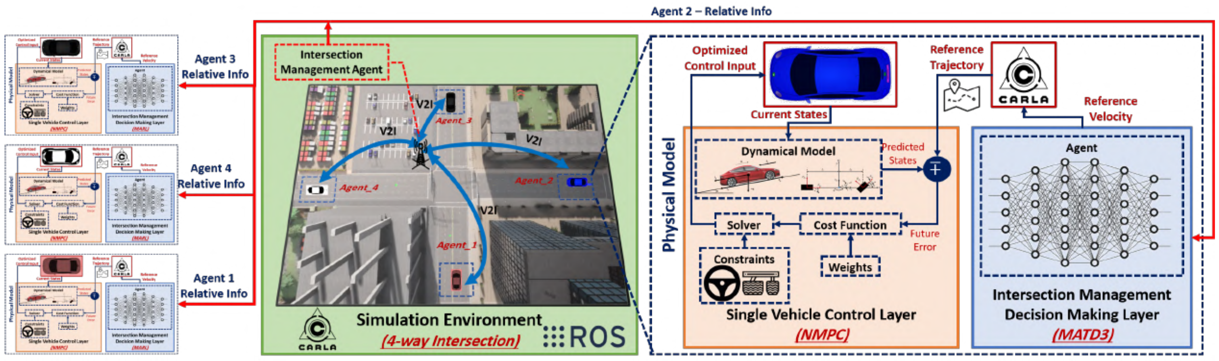


Fig. 3: Multi-layer 4-way Unsignalized Intersection Architecture Diagram

CARLA has been built to help with prototyping, training, and validation of autonomous driving models, including both perception and control. It has realistic vehicle physics. One can control the throttle, steering, and brakes of many vehicles at the same time using its Python API. One of the most important features of CARLA is its Waypoint API, which was used to provide the reference trajectory to the vehicles.

In the simulation, two different scenarios are tested in order to validate the architecture layers. The first environment is designed to test the ability of the NMPC to track waypoints generated by CARLA in four-left turns around a block with an area of $60 \times 50m^2$ having arcs resembling the turns of radius $15m$ as shown in Figure 4a.

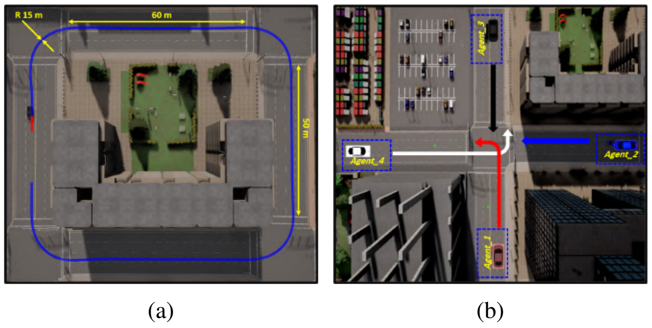


Fig. 4: Simulation Environment in Town05: (a) Four-left Turns around a Block, (b) Four-Way Intersection

Regarding the second environment, it is dedicated for testing the overall intersection architecture. In the tested environment, the number of agents (vehicles) is fixed to four. The vehicles are spawned in an intersection in Town05 in CARLA as shown in Figure 4b with a collision sensor attached to each vehicle. Each vehicle is assigned a specific destination as indicated using arrows where agent 1 (red) and agent 4 (white) take the left turn, while agent 2 (blue) and agent 3 (black) continue straight through the intersection.

In this scenario, a homogeneous multiple autonomous cars are used. The chosen car is the Tesla Model 3 with a mass (m) of $1500kg$ and a yaw moment of inertia (I_z) of $3175kgm^2$. The vehicle's longitudinal distances measured from its center of gravity (c.g.) to the front (l_f) and rear (l_r) axles are of length $1m$ and $2m$ respectively. The wheels' cornering stiffness of the front $(C_{\alpha f})$ and rear $(C_{\alpha r})$ tires are $50kN/rad$ and $33kN/rad$ respectively and the tires' radius (R) is set to $0.36m$. Most of the parameters are extracted from CARLA and used in the dynamical model to minimize the parametric uncertainties.

B. Single Vehicle Control Layer

In this layer, the NMPC is implemented to control the vehicle's motion. As illustrated in Figure 3, this layer takes the desired trajectory as input and generates the optimized control input to drive the vehicle's physical model in CARLA. The state variables and inputs in Equation (3) are set to be the previously mentioned vehicle's state vector $\bar{q}$ and control input $\bar{u}$. The discretization time constant for the model and the implemented NMPC (T_s) is adjusted to be $0.05s$. The prediction horizon (N) is chosen to be 20. The weighting matrices used for the NMPC incorporate the diagonal matrices penalizing the states' error Q, penalizing the control effort R, and penalizing the rate of change of the inputs D with diagonal elements equal to $[30, 30, 0, 5, 5, 5]$, $[5, 5, 5]$ and $[10, 5, 50]$, respectively. All constants are tuned manually and based on literature.

The NMPC optimization problem is subjected to inequality constraints considering the vehicle's control inputs limitation. This set of constraints includes the steering command (δ) which ranges satisfy $[-1.22, 1.22]rad$, throttle (α_{th}) and brake (α_{br}) commands each is between $[0, 1]$ indicating 0% to 100% throttling and braking. The constraints involve the system states as well taking into account the longitudinal velocity V_x constraints $[0.1, 30]m/s$ according to the vehicle's velocity bounds, in addition to the yaw rate $\dot{\psi}$ that varies between $[-0.8, 0.8]rad/s$.

C. Intersection Management Decision Making Layer

This layer is responsible for generating the desired speed profile that the vehicle should follow in order to avoid the collision with the other approaching vehicles in the intersection using the multi-agent RL; MATD3, defined by its state space, action space, and reward function. Moreover, this layer consists of an Intersection Management Agent (IMA) which coordinated information sharing during two phases; Centralized training and Decentralized execution. As illustrated in Figure 3, this layer takes information from the IMA as input and generates the reference velocity back to CARLA to map into waypoints to be followed by the NMPC.

1) State Space: In order for the vehicle to pass the intersection efficiently without colliding with other vehicles, the agent needs to know not only its own states, but also the states of the other vehicles approaching the intersection. The state-space $\boldsymbol{S}$ of agent i is given as:

$$\boldsymbol{S} = \left[x^i_{goal}, y^i_{goal}, x^{ij}_{rel}, y^{ij}_{rel}, v^{ij}_{x_{rel}}, v^{ij}_{y_{rel}}\right] \tag{7}$$

where x^i_{goal}, y^i_{goal} are the difference between the position of vehicle i and its target position, x^{ij}_{rel}, y^{ij}_{rel}, $v^{ij}_{x_{rel}}$, and $v^{ij}_{y_{rel}}$ are the relative position and velocity difference between vehicle i and j in the X and Y axes respectively. j ranges from 1 to the number of agents in the environment with $j \neq i$. The size of the state space is $4(n-1)+2$, where n is the number of agents (vehicles) inside the environment. All of the states are normalized between -1 and 1.

2) Action Space: The MATD3 algorithm provides each vehicle with the reference velocity required to follow in order to pass the intersection safely. Therefore, each actor will output a single continuous value that represents the reference velocity of the vehicle ranging between -1 to 1 and then mapped to a reference velocity from $5m/s$ to $10m/s$.

3) Reward Function: Two states terminate an episode; success or failure. Success is when an agent passes the intersection and reaches its target destination safely, while failure is when a collision occurs indicated by any of the collision sensors attached to the vehicles. Accordingly, the reward function r_t is defined as a constant value based on state, +1 in case of success, -5 in case of collision, and -0.1 otherwise to motivate agents (vehicles) to speed up.

4) Networks Architecture and Parameters: For both actor and critic, a two-hidden layer feed-forward neural network composed of 512 and 256 hidden nodes respectively is used. Regarding the activation functions, Rectified Linear Units (ReLU) is used for each hidden layer in both actor and critic networks, while, linear and $tanh$ activation functions are used for the output layer of the critic and the actor respectively. For the input layer, The actor takes only the state, however, the critic takes both the state and action. Both network parameters are updated using $Adam$ optimizer with a learning rate of 10^{-3} for the critic and 10^{-4} for the actor. After each time step, the networks are trained with a mini-batch of 256 transitions, sampled uniformly from a replay buffer containing the entire history of all agents.

D. Intersection Management Agent (IMA)

The Intersection Management Agent (IMA) represents the communication point that collects information from all the vehicles in the intersection and sends back information to the vehicle. This agent represents the Vehicle-to-Infrastructure (V2I) communication in the intersection system. During the training and execution phases of the MATD3, this agent follows centralized and decentralized architectures.

1) Phase 1: Centralized Training: During the training, the IMA follows a centralized manner, where it collects information about all agents at the same time. Then, it performs the MATD3 algorithm, sending the desired velocity to each vehicle's NMPC layer. The training scenario is formulated as: whenever a vehicle reaches its target destination or collides with another vehicle, it will be re-spawned to its original spawn point starting again with the lower velocity bound. Only the collided vehicle is re-spawned while the others continue their journey normally. Since each vehicle speed differs from the others, the episodes get terminated at different simulation times. Therefore, the relative distance between the vehicles is continuously changing during training which provide generalizability for the algorithm. Each vehicle is trained to the different crossing scenarios of the vehicles, this is due to the use of a single replay buffer among all agents, leading to training using the experience of all vehicles.

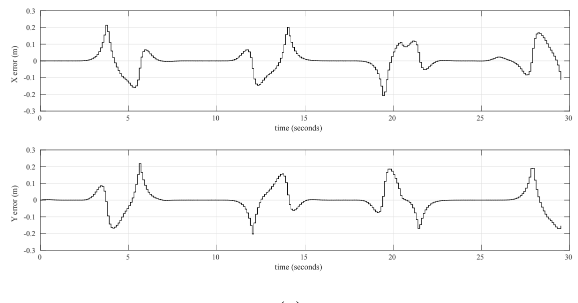
2) Phase 2: Decentralized Execution: During the execution, the IMA follows a decentralized manner. Whenever a vehicle approaches the intersection, it sends information about its position, velocity, and target destination. The IMA collects the same information from all vehicles approaching the intersection. Afterward, the IMA gives each vehicle its corresponding relevant information where a deployed RL agent shall use this information as input to the trained neural network to generate the reference velocity back to CARLA to generate the waypoints to be followed by the NMPC. The vehicle keeps sending its updated information to the IMA till it passes the intersection. It is also worth mentioning that the reference velocity will remain the same for one second, despite the NMPC sampling time.

IV. Results and Discussion

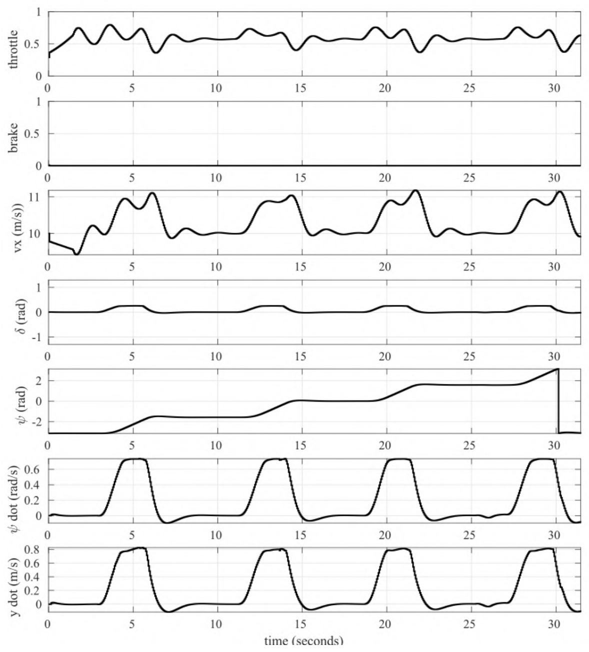
The validation of the architecture is done layer by layer, where first the response of the single vehicle control layer is validated on the four-turns block environment. Afterwards, the intersection management decision making layer two phases are verified to prove the concept of the IMA. The visualization of the results can be represented through the experiment link https://youtu.be/CjD8EcMpZPQ.

A. Single Vehicle Control Layer Validation

The NMPC controller's performance is tested on a single vehicle given a reference velocity of $10m/s$. The waypoints to be followed are generated by CARLA's Waypoint API. It is found that the average computation time is about $0.036s$ per time step which is considered sufficient for real-life system. Analyzing some graphical results to test the ability of the vehicle to ensure passenger comfort, Figure 5b illustrates the control input to drive the vehicle, longitudinal and lateral velocities, yaw angle, and yaw rate. It is observed that the performance is smooth and stable without fluctuations.



(a)



(b)

Fig. 5: Single Vehicle Control Layer Performance: (a) X and Y Position Errors, (b) Control Inputs and States Variables

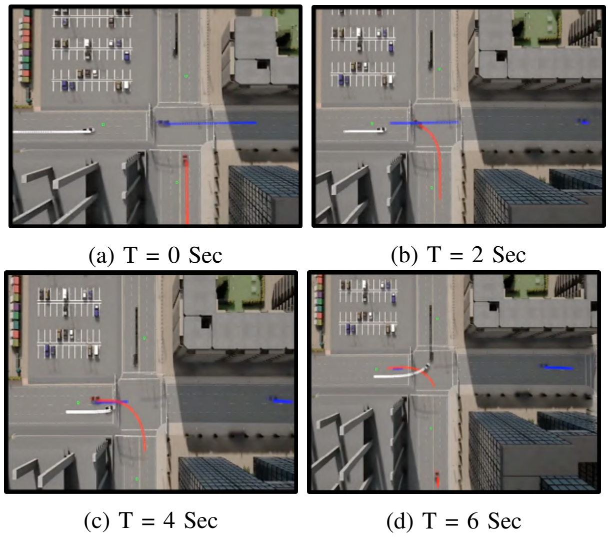
Moreover, it is noticed that the velocity profile faces an increase in the four turns, however, the performance is within acceptable bounds. Regarding the tracking capabilities, errors in the X and Y axes are plotted in 5a revealing that the maximum error in both directions is about $0.2m$ which proves the results' reliability.

B. Intersection Management Layer Performance

1) Phase 1: Centralized Training Results: In phase 1, the MATD3 algorithm is trained in the intersection environment for $5k$ episodes for all agents, equally distributed among all agents. Illustrating the training performance, Figure 6 shows some snippets taken for the four agents in the intersections while training. It can be noticed from Figure 6d that agents 3 and 4 collided, while agents 1 and 2 passed safely.

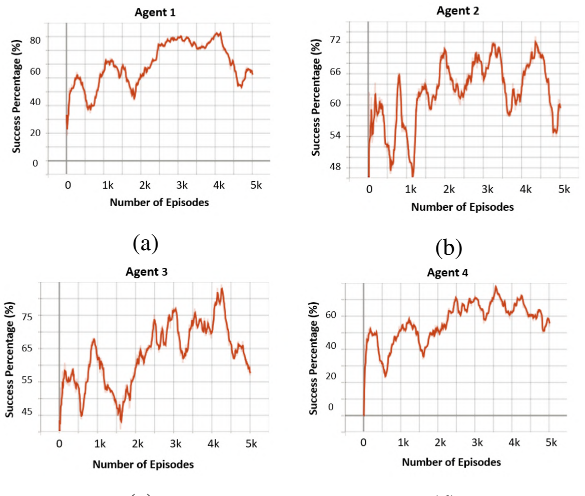
Regarding the numerical analysis, Figure 7 shows the success percentages for all agents over the training period over the last 100 episodes reflecting the number of times the agent is able to pass the intersection and reach its target destination without colliding. It can be observed that all agents learned to increase the success percentage as the training continues. Agent 1 has the maximum success percentage compared to all other agents with 93% and it is achieved in episode 4141 while agents 2, 3, and 4 had a maximum success percentage of 73%, 84%, and 78% respectively.

Figure 8a shows the average reward of all agents over the last 100 episodes. From the figure, it is observed that on average all agents learned to increase the total reward obtained in an episode as the training continues. The maximum average reward reaches 0.716 collected in episode 4041. Figure 8b shows the average success $\%$ of the agents to evaluate the effectiveness of the algorithm. The agents' model with the highest average success percentage was saved. The maximum average success percentage reached 78.5% in episode 4146


(a) T = 0 Sec (b) T = 2 Sec

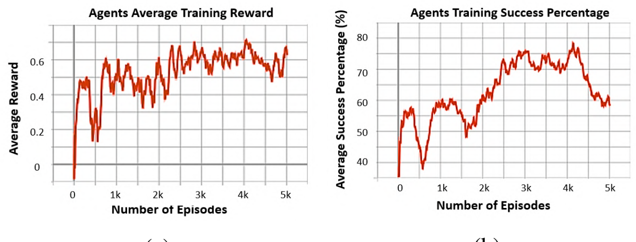
(c) T = 4 Sec (d) T = 6 Sec

Fig. 6: Training Snippets in the Intersection



(a) (b) (c) (d)

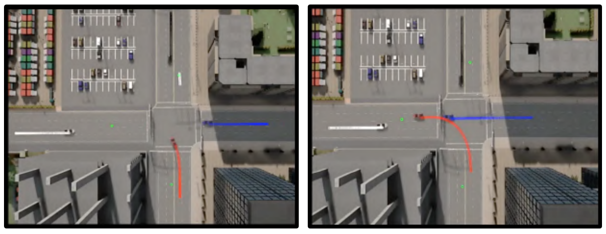
Fig. 7: Success Percentage: (a) Agent 1, (b) Agent 2, (c) Agent 3, (d) Agent 4



(a) (b)

Fig. 8: Training Average (a) Reward, (b) Success %

2) Phase 2: Decentralized Execution Results: After saving the superior trained model, the testing method is conducted for 400 episodes under the same conditions without the noise that has been added during training deployed in all vehicles. Figure 9 shows some snippets taken for the four agents in the intersections while testing. It can be easily noticed that all agents passed safely.


(a) T = 0 Sec (b) T = 0.5 Sec

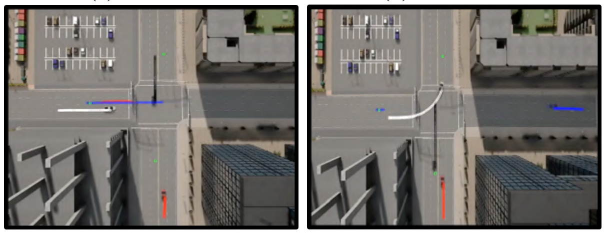
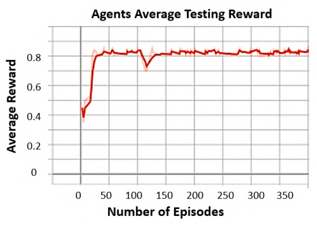
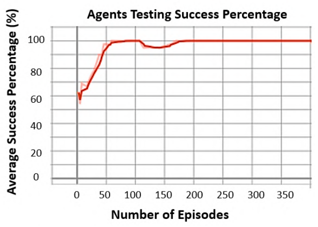

(c) T = 1 Sec (d) T = 1.5 Sec

Fig. 9: Execution Snippets in the Intersection



(a)



(b)

Fig. 10: Execution Average (a) Reward, (b) Success %

The testing results proved the capability of the MARL to find the correct policy to avoid collision in a short number of episodes. Figure 10a shows that the average reward gained by the 4 agents maxed-out at around 0.8, while Fig. 10b shows some failures at the beginning of testing while maintaining an average success percentage of all agents of a value around 100% during the remaining episodes.

V. Conclusion and Future Recommendations

In this paper, the problem of unsignalized intersections is addressed. A multi-layer architecture is proposed where it is composed of two layers. The first is responsible for the decision making in the intersection and is controller via MATD3 algorithm while the second deals with the single vehicle control using dynamical-based NMPC. An intersection management agent is introduced to coordinate the information shared in both training and execution phases. The results are tested on an intersection environment built in CARLA. The proposed MATD3 was able to find the correct policy to avoid collisions with a success percentage of a 100%. As for future recommendations, it is recommended to investigate more dynamic scenarios that consider as an example, the increase in the number of agents, and mixed traffic including autonomous and human-driven vehicles and other road agents as pedestrians to improve the robustness of the decision making layer and mimic the real-road scenarios.

References

[1] J.-P. Rodrigue, *The geography of transport systems*. Routledge, 2020.
[2] T. Reed, “Inrix global traffic scorecard,” 2019.
[3] W. H. Organization *et al.*, “Global status report on road safety 2018: Summary,” World Health Organization, Tech. Rep., 2018.
[4] “Fatality analysis and reporting system (fars) data publication 2018,” https://www-fars.nhtsa.dot.gov/Main/index.aspx, accessed: 2021.
[5] M. M. Abdelhameed, M. Abdelaziz, S. Hammad, and O. M. Shehata, “Development and evaluation of a multi-agent autonomous vehicles intersection control system,” in *2014 International Conference on Engineering and Technology (ICET)*. IEEE, 2014, pp. 1–6.
[6] B. Mirchevska, C. Pek, M. Werling, M. Althoff, and J. Boedecker, “High-level decision making for safe and reasonable autonomous lane changing using reinforcement learning,” in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2018, pp. 2156–2162.
[7] T. Tram, A. Jansson, R. Grönberg, M. Ali, and J. Sjöberg, “Learning negotiating behavior between cars in intersections using deep q-learning,” in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2018, pp. 3169–3174.
[8] T. Tram, I. Batkovic, M. Ali, and J. Sjöberg, “Learning when to drive in intersections by combining reinforcement learning and model predictive control,” in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2019, pp. 3263–3268.
[9] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. Al Sallab, S. Yogamani, and P. Pérez, “Deep reinforcement learning for autonomous driving: A survey,” *IEEE Transactions on Intelligent Transportation Systems*, 2021.
[10] B. Kiumarsi, K. G. Vamvoudakis, H. Modares, and F. L. Lewis, “Optimal and autonomous control using reinforcement learning: A survey,” *IEEE transactions on neural networks and learning systems*, vol. 29, no. 6, pp. 2042–2062, 2017.
[11] L. Buşoniu, T. de Bruin, D. Tolić, J. Kober, and I. Palunko, “Reinforcement learning for control: Performance, stability, and deep approximators,” *Annual Reviews in Control*, vol. 46, pp. 8–28, 2018.
[12] Y. Lin, J. McPhee, and N. L. Azad, “Longitudinal dynamic versus kinematic models for car-following control using deep reinforcement learning,” in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2019, pp. 1504–1510.
[13] H. Mirzaei and T. Givargis, “Fine-grained acceleration control for autonomous intersection management using deep reinforcement learning,” in *2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*. IEEE, 2017, pp. 1–8.
[14] R. Rajamani, “Lateral vehicle dynamics,” in *Vehicle Dynamics and control*. Springer, 2012, pp. 15–46.
[15] R. Attia, R. Orjuela, and M. Basset, “Combined longitudinal and lateral control for automated vehicle guidance,” *Vehicle System Dynamics*, vol. 52, no. 2, pp. 261–279, 2014.
[16] J. A. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, “Casadi: a software framework for nonlinear optimization and optimal control,” *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.
[17] S. Fujimoto, H. Hoof, and D. Meger, “Addressing function approximation error in actor-critic methods,” in *International Conference on Machine Learning*. PMLR, 2018, pp. 1587–1596.
[18] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” *arXiv preprint arXiv:1509.02971*, 2015.
[19] J. Ackermann, V. Gabler, T. Osa, and M. Sugiyama, “Reducing overestimation bias in multi-agent domains using double centralized critics,” *arXiv preprint arXiv:1910.01465*, 2019.
[20] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, “Multi-agent actor-critic for mixed cooperative-competitive environments,” *arXiv preprint arXiv:1706.02275*, 2017.
[21] 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.