IEEE Open Journal of
Intelligent Transportation
Systems

Received 7 August 2024; revised 8 October 2024 and 3 December 2024; accepted 7 December 2024. Date of publication 11 December 2024; date of current version 31 December 2024.

Digital Object Identifier 10.1109/OJITS.2024.3515270

DeepGame-TP: Integrating Dynamic Game Theory and Deep Learning for Trajectory Planning

GIOVANNI LUCENTE[1,2], MIKKEL SKOV MAARSSOE[1], SANATH HIMASEKHAR KONTHALA[1], ANAS ABULEHIA[1], REZA DARIANI[1], AND JULIAN SCHINDLER[1]

[1]Institute of Transportation Systems, German Aerospace Center (DLR), 38108 Braunschweig, Germany
[2]Fakultät Verkehrs- und Maschinensysteme, Technische Universität Berlin, TU Berlin, 10623 Berlin, Germany

CORRESPONDING AUTHOR: G. LUCENTE (e-mail: giovanni.lucente@dlr.de)

This work was supported by the German Aerospace Center (DLR).

ABSTRACT Trajectory planning for automated vehicles in traffic has been a challenging task and a hot topic in recent research. The need for flexibility, transparency, interpretability and predictability poses challenges in deploying data-driven approaches in this safety-critical application. This paper proposes DeepGame-TP, a game-theoretical trajectory planner that uses deep learning to model each agent's cost function and adjust it based on observed behavior. In particular, a LSTM network predicts each agent's desired speed, forming a penalizing term that reflects aggressiveness in the cost function. Experiments demonstrated significant advantages of this innovative framework, highlighting the adaptability of DeepGame-TP in intersection, overtaking, car following and merging scenarios. It effectively avoids dangerous situations that could arise from incorrect cost function estimates. The approach is suitable for real-time applications, solving the Generalized Nash Equilibrium Problem (GNEP) in scenarios with up to four vehicles in under 100 milliseconds on average.

INDEX TERMS Dynamic game, deep learning, generalized Nash equilibrium, LSTM, trajectory planning.

I. INTRODUCTION

IN RECENT years, many successful data-driven approaches have been applied to the field of automated driving, particularly in perception, prediction and planning tasks. Recent advances in deep learning techniques, such as transformers, large language models, and generative AI, have shown great potential in image and speech recognition and generation. The research community is now rushing to apply these techniques to automated driving, hoping to achieve the same success seen in other fields. While the application of deep learning techniques in perception is considered state-of-the-art, the discussion about which approach should be considered standard for planning is still ongoing and a hot topic in the literature.

It is possible to identify three main challenges in the planning task: the mutual influence between prediction and planning, the computational time constraint for online applications and the requirement for interpretability and transparency, which are necessary for a safety application like driving automation.

Data-driven approaches have shown great potential in the trajectory prediction task, particularly in interactive multi-agent scenarios where the reciprocal influence between traffic participants is crucial. Nevertheless, prediction alone is insufficient for effective planning. Separating prediction from planning can lead to suboptimal behavior, where planning merely reacts passively to predictions, or to inaccurate estimates, as the actions of the ego vehicle also affect other traffic participants. For this reason, many works in the literature prefer to consider prediction and planning as a single task.

Another crucial challenge for trajectory planning algorithms is the stringent computational time constraints required for real-time online applications. These algorithms must process data and generate accurate plans quickly (generally within 100 ms) to ensure the responsiveness and safety of automated driving systems.

The review of this article was arranged by Associate Editor Vesna Šešum-Čavić.

Deploying an Advanced Driver Assistance System (ADAS) in the real world requires a thorough understanding of its functionality. In this context, every algorithm used in safety applications must be interpretable and transparent. Good performance on a test set does not necessarily mean the solution will work well in real-world conditions, as testing sets often fail to capture the full range of deployment scenarios. Furthermore, the non-transparent nature of some systems prevents users from addressing issues, even when they are aware of them. This is particularly problematic for data-driven approaches, which are often designed in this manner. Therefore, transparent AI has become a major research focus, with many solutions being proposed to improve understanding and trust in these systems [1].

To tackle this challenge, this paper presents DeepGame-TP, a game-theoretic trajectory planner for multi-agent traffic scenarios that incorporates deep learning while preserving transparency in its approach. DeepGame-TP utilizes an Augmented Lagrangian Trust-Region solver to find Generalized Nash Equilibria (GNE) in dynamic games. The agents' behavior is modeled in the cost function by adding a penalty for the distance between the future speed and the desired speed profile, predicted through a Long-Short Term Memory (LSTM) neural network, trained on the NGSIM dataset [2]. The contributions of this paper are:

1) Definition of a framework that integrates deep learning into a game-theoretic approach for trajectory planning. An LSTM neural network predicts the desired longitudinal behavior of each actor, which is then incorporated into the cost function to compute the GNE. The approach combines the transparency of a dynamic game-based trajectory planning framework with the adaptability and behavior recognition capabilities of deep learning.
2) Definition of a straightforward approach to characterize the behavior of agents in a traffic environment. The predicted desired speed indicates the driver's level of aggressiveness.
3) Definition of a Trust Region solver based on an Augmented Lagrangian formulation, enabling real-time computation of the GNE and facilitating online applications.

II. RELATED WORKS

Trajectory planning in traffic has been tackled using various approaches to address the complexity of this task [3], [4], [5], [6], [7]. The challenges to face include optimal control in a multi-agent environment [3], [5], real-time applications and modeling the behavior of different agents [6], [7]. It is possible to categorize the approaches into model-based and learning-based methods. Game Theory is the traditional framework for modeling multi-agent environments and identifying optimal policies within them. The availability of large datasets and simulation environments allows us to bypass explicit modeling of agent interactions, instead delegating this task to learned models. Common learning-based methods are Reinforcement Learning, Imitation Learning and, more recently, Generative Models.

Reinforcement Learning (RL) algorithms enable an agent to act optimally in an environment by continuously interacting with it and collecting feedback on its behavior. The primary limitations of applying RL to trajectory planning for autonomous driving arise from the complexity of accurately representing real-world driving environments in simulation, from the high-dimensional state and action spaces inherent in trajectory planning tasks and from the challenge of designing an appropriate reward function. An agent that has learned an effective policy in a simulated environment does not guarantee optimal performance in reality. The simulated environment and the curse of dimensionality, resulting in sample inefficiency, pose challenges in generalizing the policy to all possible scenarios. Consequently, the agent's behavior can become unpredictable in scenarios not encountered during training, with an additional risk of overfitting. Another challenge is defining a reward function that consistently leads to an optimal policy in all situations, without causing suboptimal or unsafe behavior. For these reasons, RL still suffer from long training times and poor performance [8]. Recent works that apply RL in trajectory planning for automated driving and propose solutions for these problems are [9], [10], [11], [12]. In [9], the authors propose a simulation framework to generate driving scenarios from a real word dataset for training. The trained RL algorithm is then tested in a non-signalized T-junction and a non-signalized lane merge intersection. In [10], the authors propose a hierarchical framework for trajectory planning. Instead of directly mapping sensor information to low-level control signals, RL is applied only to the subtask of choosing a desired state. Subsequently, a low-level planner is used to generate and track the trajectory according to the chosen state. This is a common tendency that can be found in literature: since RL does not always provide effective end-to-end planning performance, often it is used to solve a smaller part of the planning problem [8]. In [11], the authors face the problem of sparse rewards, that deteriorate the performance of RL. They define a new reward function by leveraging field approximations, which is demonstrated to yield dense rewards. As these examples illustrate, defining an appropriate reward function remains a challenge for the performance of RL algorithms.

Imitation Learning (IL) provides a framework where defining a reward function is no longer necessary. Instead, the agent learns directly from examples provided by an expert, mapping observations to actions [13]. The literature presents different examples of IL application in the field of automated vehicles, particularly employing end-to-end approaches [14]. It is possible to divide IL techniques into Behavioral Cloning (BC), Inverse Reinforcement Learning (IRL) and Adversarial Imitation Learning (AIL). In Behavioral Cloning (BC), a model is trained through supervised learning to map the state of the environment to the corresponding expert action. The main advantage of BC is its simplicity; it does not require

knowledge of the environment's dynamics, as it relies solely on expert demonstrations. The most well-known limitation of this approach is the covariate shift problem [13], [15], which occurs when the state distribution during training differs from that during testing. This problem arises because the agent is trained on states generated by the expert policy but is tested on states influenced by its own policy. As a result, the agent may encounter traffic situations it never encountered during training, potentially leading to safety issues. Inverse Reinforcement Learning (IRL) is an alternative approach to IL, where the agent infers the underlying reward function or policy from expert demonstrations. Once the reward function is inferred, it is used to learn an optimal decision-making model through RL. IRL is less sensitive to the covariate shift problem because the state distribution during both training and testing is generated by the agent, ensuring consistency. However, major limitations of IRL include the difficulty of inferring the reward function and the high computational cost during training. These challenges are particularly pronounced for complex tasks and rich state-action spaces, leading to the curse of dimensionality. A policy indeed can be optimal for an infinite number of reward functions [13]. Recently some works have deployed IRL for trajectory planning in traffic [16], [17], [18]. In these works, the application of IRL is limited to a score module to evaluate already planned trajectories, as in [16], [17], or to a Personalized Adaptive Cruise Control (P-ACC) to learn the driver's car-following preferences from historical data, like in [18]. Adversarial Imitation Learning (AIL) is an imitation learning strategy that involves a competitive game between an agent and an adversary (discriminator). The agent generates trajectories aimed at emulating those of the expert, while the adversary endeavors to distinguish the agent's generated trajectories from the original ones provided by the expert. Recent works have shown the potentiality of AIL [19], [20], [21], however, a common issue with these approaches is that the driving policy may not perform well in situations that are different from those encountered during training [22].

Recent advancements in generative models have sparked increased interest within the research community in applying these models to automated driving, particularly in trajectory planning and prediction. However, most approaches primarily focus on predicting trajectories and generating traffic scenarios based on specific inputs, making examples of applications in trajectory planning rare. In trajectory planning, the generation process must be conditioned on inputs that specify the long-term objective of the planned trajectory, making the overall design and training process more complex compared to trajectory prediction. The most used approaches for trajectory prediction are Generative Adversarial Networks (GAN) [23], [24], Variational Autoencoders (VAE) [25] and Diffusion Models [26], [27], [28]. Diffusion models have proven to be highly effective in prediction tasks. However, they involve numerous computationally expensive denoising steps and sampling operations, making them less suitable for real-time, safety-critical applications [26]. A significant advancement was the introduction of transformers, which enhanced the performance of natural language processing algorithms through their innovative architecture. Their application in trajectory prediction also shows promising results [4], [29]. Nevertheless, the most critical issue of the application of data-driven approaches in safety tasks is the lack of transparency of neural network based architectures. The output is not predictable and there are limitations in detecting, understanding and fixing issues. This constitutes a big topic in research recently [1].

Game theory is the traditional framework used to model multi-agent environments and define optimal policies within them. The traffic problem is often framed as a dynamic game [30], [31], [32], [33], where the objective is to solve the Generalized Nash Equilibrium Problem (GNEP). A Generalized Nash Equilibrium (GNE) is a type of Nash Equilibrium (NE) where players are interconnected through shared state constraints [34], such as collision avoidance. In [30], the authors introduce an augmented Lagrangian algorithm to solve GNEPs for trajectory optimization problems. The proposed solver is based on a quasi-Newton root-finding algorithm to satisfy the first order optimality conditions, with constraints enforced using an augmented Lagrangian formulation. The algorithm is tested in highly interactive scenarios, like intersection and merging. Nonetheless, there is no distinction in the agents' objective functions. In [31], [33], two different methods for estimating and differentiating the agents' objective functions are presented. In [33], the authors propose an inverse optimal control algorithm that is able to estimate the other agents' objective functions in real time. From the normal distribution of the objective function parameters, sigma points are sampled. For each of these points, the GNEP is solved, resulting in a set of predicted trajectories. Upon receiving a new system measurement, an Unscented Kalman Filter updates the parameter distribution, making the sigma points with better predictive performance more likely. This approach requires solving a GNEP for each sigma point, where the number of sigma points is linearly proportional to the number of agents. In [31], Social Value Orientation (SVO) is employed to characterize agents' behavior in the dynamic game, enhancing prediction accuracy when computing the NE. SVO quantifies the extent of an agent's selfishness or altruism. The utility function for each agent is defined as a combination of its own rewards and those of other agents, weighted by the SVO angular preference. The reward functions are learned through IRL from the NGSIM driving data. The likelihood of candidate SVOs is computed evaluating the Gaussian kernel on the distance between predicted and actual trajectories. The features used to compute the reward function, however, are not always observable in realistic scenarios, which poses challenges for applying this approach to a real trajectory planner.

Generally, methods that estimate the cost functions of other agents online require solving prediction sub-problems to evaluate candidate parameters, as seen in the two

examples cited above. This process can degrade computational performance and affect online applicability. The implementation of learned models for this task, which avoids the need to solve sub-problems, has received limited exploration in the literature. DeepGame-TP addresses the limitations previously discussed:

- The lack of transparency and interpretability in learning-based methods, which is essential for safety-critical applications like trajectory planning.
- The issues with flexibility and reliability in learning-based methods, which are often constrained by the scope and generality of the dataset used.
- The absence of cost function estimation in traditional game-theoretical approaches.
- The computational complexity of solving GNE, along with the added complexity of solving prediction sub-problems for cost function estimation, both of which limit real-time applicability.

DeepGame-TP maintains the transparency of traditional approaches through its dynamic game formulation while benefiting from deep learning to estimate each agent's cost function. This enables the system to act optimally by recognizing and adapting to the behaviors of different agents. Experiments show its flexibility and reliability across various scenarios and topologies, with computational efficiency that supports real-time applications for up to four vehicles.

III. PROBLEM STATEMENT

The traffic environment is modeled as a multi-player dynamic game, the proposed algorithm solves the GNEP without distinguishing between the automated vehicle and the other vehicles that are present in the traffic scenario. The time horizon is discretized with N steps, the state and input size are denoted by n_X and n_U respectively. The state and the control input of the agent i at time step k are denoted with $\boldsymbol{x}_k^i$ and $\boldsymbol{u}_k^i$. The state of each agent at time step k is composed by $\boldsymbol{x}_k^i = [x,\ y,\ \psi,\ v]_k^T$, i.e., the cartesian coordinates, heading and speed, while the input is composed by $\boldsymbol{u}_k^i = [\delta,\ F]_k^T$, i.e., the steering angle and the longitudinal force.

Let's consider the GNEP with M players, each player i decides over its control input sequence, denoted with $U^i = [(u_1^i)^T, \dots\ (u_{N-1}^i)^T]^T \in \mathbb{R}^{n_U(N-1)}$, that generates a trajectory denoted with $X^i = [(x_2^i)^T, \dots\ (x_N^i)^T]^T \in \mathbb{R}^{n_X(N-1)}$, according to the dynamic model:

$$\boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{u}_k) \tag{1}$$

The cost function of agent i is denoted by $J^i(X^i, U^i) : \mathbb{R}^{n_U(N-1)} \to \mathbb{R}$, it depends on the trajectory and on the control input sequence. The goal of each agent is to minimize the cost function without violating the constraints. The trajectory of the system, that aggregates the agents' trajectories, is indicated with $X = [X^1, \dots\ X^M]^T \in \mathbb{R}^{n_X(N-1)\ M}$, the aggregate input sequence of the system is denoted by $U = [U^1, \dots\ U^M]^T \in \mathbb{R}^{n_U(N-1)\ M}$. The vector of all the players' strategies except the one of player i is denoted with U^{-i}. The problem is formalized as a set of optimization problems:

$$\begin{aligned} \min_{X^i, U^i} \quad & J^i\left(X^i, U^i\right) \\ \text{s.t.} \quad & D^i\left(X^i, U^i\right) = 0 \\ & C^i(X, U) \ < 0 \end{aligned} \tag{2}$$

This set of M problems constitutes a GNEP, since they are coupled through the inequality constraints $C^i(X, U) \ < 0$, that depend on the control input sequence of every vehicle. This coupling comes from the collision avoidance constraints. The equality constraints $D^i(X^i, U^i) = 0$ represent the dynamic model that links the control input sequence U^i with the trajectory X^i. However, in the practical implementation of the optimization problem, the equality constrained are ignored, since the dynamic model is imposed through the integration function $X^i = F(U^i, \boldsymbol{x}_0^i)$, where $\boldsymbol{x}_0^i$ is the i vehicle state at time 0. The optimization problem is then simplified:

$$\begin{aligned} \min_{U^i} \quad & J^i\left(U^i\right) \\ \text{s.t.} \quad & C^i(U) < 0 \end{aligned} \tag{3}$$

The solution to this set of optimization problems, U, is an open-loop generalized Nash equilibrium. The control signal U^i of agent i is the best response to the other agents' strategies U^{-i} given the initial state of the system $\boldsymbol{x}_0 = [\boldsymbol{x}_0^1, \dots, \boldsymbol{x}_0^M]^T$. It is open-loop since the control input sequences U is a function of time, and not function of the state of the system at time step k, $\boldsymbol{x}_k$. Nonetheless, if the open-loop game is repeatedly resolved online, as new information is obtained, the solution constitutes a policy that is closed-loop in the model-predictive control sense [30]. In the following subsections we will present the augmented Lagrangian formulation of the problem, the Trust Region solver and how are practically implemented the cost function and the inequality constraints.

A. AUGMENTED LAGRANGIAN FORMULATION

The constrained optimization problems are transformed into unconstrained optimization problems through the augmented Lagrangian formulation. The augmented Lagrangian for each agent i is then defined as:

$$L^i(U) = J^i\left(U^i\right) + {\lambda^i}^T C^i(U) + \frac{1}{2} C^i(U)^T I_\mu \ C^i(U) \tag{4}$$

where J^i is the cost function, λ^i is the vector of Lagrangian multipliers associated to the inequality constraints C^i and I_μ is the penalty weight matrix, whose elements are:

$$I_{\mu,jq} = \begin{cases} \mu \text{ if } j = q \text{ and } C_j^i \geq 0 \\ 0 \text{ if } j \neq q \text{ or } \quad C_j^i < 0 \end{cases} \tag{5}$$

where μ is the penalty weight for the quadratic term, which, along with the Lagrange multiplier, is increased if

the corresponding constraint is violated. The updating rules are:

$$\lambda_j^{i\ (k+1)} \leftarrow \max\Big(0,\ \lambda_j^{i\ (k)} + \mu^{(k)} C_j^{i\ (k)}\Big)$$
$$\mu^{(k+1)} \leftarrow \gamma \mu^{(k)} \qquad \gamma > 1 \tag{6}$$

The strategy U^i is considered optimal for agent i, as it minimizes J^i under the inequality constraints C^i, if:

$$\nabla_{U^i} L^i(U) = 0 \tag{7}$$

If equation (7) is valid $\forall i$, that is, for each agent in the scenario, then the solution U is a GNE. The unconstrained minimization problem is solved using a Trust Region algorithm, which is explained in the following subsection.

B. TRUST REGION SOLVER

In the unconstrained optimization, Trust Region methods define a region around the current iterate where the quadratic model is trusted to be a good approximation of the objective function. Within this region, the algorithm seeks the approximate minimizer of the model. Trust Region methods determine both the direction and the step size simultaneously. If a step is not acceptable, the region's size is reduced, and a new minimizer is sought. The size of the trust region is crucial for the efficiency of each step: a small region results in small steps, while a large region may lead to a minimizer of the model that is far from the function's minimizer. The size of the trust region is adjusted based on the algorithm's performance in the previous iteration [35].

Algorithm 1 presents the details of the Trust Region solver. Table 1 shows the parameters of the algorithm. Some key points should be highlighted:

- The Trust Region method is typically used to solve a single unconstrained optimization problem. In this work, however, it is employed to solve a generalized Nash equilibrium, involving multiple optimization problems that are coupled through inequality constraints.
- In the algorithm, there is an initial loop over the agents to solve the sub-problems, followed by a second loop to check if the actual reduction in the cost function is close to the predicted one. Having two separate loops instead of a single combined loop ensures that all agents are treated equally, preventing any strategic advantage for the agents that appear earlier in the list.
- The calculation of the Hessian matrix for the sub-problem is a critical point for computational complexity and online feasibility. To mitigate this complexity, in this implementation, the Hessian matrix is estimated using the Symmetric Rank-One (SR1) method (Algorithm 2).

Algorithm 1: Trust Region Algorithm

Input: Initial system state $\boldsymbol{x}_0 = [\boldsymbol{x}_0^1, \ldots, \boldsymbol{x}_0^M]^T$, Lagrange function $L^i(U)$, initial guess U_0, initial trust region radius Δ_0, tolerance ϵ, acceptance treshold η, maximum number of iterations k_{max}

Output: Agents' optimal control sequence $U^* = [U^{*1}, \ldots\ U^{*M}]^T$

Initialize: $k \leftarrow 0$, $U_k \leftarrow U_0$, $\Delta_k^i \leftarrow \Delta_0$, $H_k^i \leftarrow I$;
while *not converged and* $k < k_{max}$ **do**
  $\nabla L(U_k) \leftarrow [\nabla_{U_k^1} L^1, \ldots\ \nabla_{U_k^M} L^M]^T$;
  $L(U_k) \leftarrow [L^1(U_k^1), \ldots\ L^M(U_k^M)]^T$;
  for *each agent* i **do**
    $s^i \leftarrow \arg\min_{s^i} s^{iT} \nabla L^i(U_k) + \frac{1}{2} s^{iT} H_k^i s^i$ s. t. $\|s^i\| \le \Delta_k^i$;
  end
  $s = [s^1, \ldots s^M]^T$;
  for *each agent* i **do**
    $\delta L^i \leftarrow L^i(U_k) - L^i(U_k + s)$;
    $\delta \hat{L}^i \leftarrow -(s^{iT} \nabla L^i(U_k) + \frac{1}{2} s^{iT} H_k^i s^i)$;
    $\rho_k^i \leftarrow \delta L^i / \delta \hat{L}^i$;
    if $\rho_k^i > \eta$ **then**
      $U_{k+1}^i \leftarrow U_k^i + s^i$;
    else
      $U_{k+1}^i \leftarrow U_k^i$;
    end
    if $\rho_k^i > 0.75$ **then**
      if $\|s^i\| > 0.8\ \Delta_k^i$ **then**
        $\Delta_{k+1}^i \leftarrow 2.0\ \Delta_k^i$;
      end
    end
    if $\rho_k^i < 0.1$ **then**
      $\Delta_{k+1}^i \leftarrow 0.5\ \Delta_k^i$;
    end
    $\delta \nabla L^i = \nabla_{U_k^i} L^i(U_k^i + s^i) - \nabla_{U_k^i} L^i(U_k^i)$;
    $H_{k+1}^i \leftarrow SR1(H_k^i,\ \delta \nabla L^i,\ s^i)$;
  end
  if $\|\nabla L(U_{k+1})\| \le \epsilon$ **then**
    converged;
  end
  $C(U_{k+1}) \leftarrow [C^1(U_{k+1}), \ldots C^M(U_{k+1})]^T$;
  $\lambda^{(k+1)}, \mu^{(k+1)} \leftarrow update(\lambda^{(k)},\ \mu^{(k)},\ C(U_{k+1}))$;
  $k \leftarrow k + 1$;
end
return U_{k+1}

C. COST FUNCTION

The GNE simultaneously represents the planned trajectory for the controlled vehicle and the predicted trajectories for the other vehicles. To ensure a reliable model of the agents, and consequently a reliable prediction and optimal strategy, the agents' behavior must be understood and accurately represented in the cost function they aim to minimize. In this work, the cost function component that gives a representation of the agent behavior is the desired speed, predicted through a Long-Short Term Memory (LSTM) neural network. A higher desired speed will lead to more aggressive behaviors such

Algorithm 2: Symmetric Rank 1 (SR1) Hessian update

Input: Initial Hessian H_k, gradient difference $y = \nabla f(x + \Delta x) - \nabla f(x)$, step Δx, parameter $r \in (0, 1)$

Output: Hessian update H_{k+1}

```
if |Δx^T (y − HΔx)| ≥ r ‖Δx‖ ‖∇y − HΔx‖ then
    H_{k+1} = H_k + (y − HΔx)(y − HΔx)^T / (y − HΔx)^T Δx;
else
    H_{k+1} = H_k;
end
return H_{k+1}
```

TABLE 1. Parameters of the trust region algorithm.

Δ_0^i	ϵ	η	k_{max}
1.0	$1e^{-2}M$	$1e^{-4}$	25

as overtaking, taking priority when entering unsignalized intersections or maintaining a lower time headway.

The cost function for agent i is the following:

$$J^i\left(U^i\right) = \frac{q_f}{2}\left(\sum_{k=0}^{k<N}\left[\alpha_1\left\|\tilde{\boldsymbol{x}}_k^i - \tilde{\boldsymbol{x}}_k^C\right\|^2 + \alpha_2\left(\left\|cos\left(\psi_k^i\right) - cos\left(\psi_k^C\right)\right\|^2 + \left\|sin\left(\psi_k^i\right) - sin\left(\psi_k^C\right)\right\|^2\right) + \alpha_3\left\|v_k^i - \hat{v}_k^i\right\|^2 + \alpha_4\left\|F_k^i\right\|^2\right]\right)^2 \tag{8}$$

where $\tilde{\boldsymbol{x}}_k^i$ and ψ_k^i are the cartesian coordinates and the heading of the trajectory of angent i at timestep k, $\tilde{\boldsymbol{x}}_k^C$ and ψ_k^C the cartesian coordinates and the heading of the closest point on the centerlane from the point $\tilde{\boldsymbol{x}}_k^i$. The term F_k^i denotes the longitudinal force input. The coefficients q_f, α_1, α_2, α_3, α_4 have been epirically tuned to have a natural behavior and to avoid numerical instability ($q_f = 1e^{-2}$, $\alpha_1 = 1e^{-1}$, $\alpha_2 = 1e^{2}$, $\alpha_3 = 1e^{-1}$, $\alpha_4 = 1.0$). The first two terms in the cost function penalize the deviation from center line, in terms of cartesian distance and heading, the third term penalizes the deviation from the predicted longitudinal behavior, the last term penalizes the longitudinal force input. The variable $\hat{v}_k^i$ indicates the desired speed of agent i at time step k. For the other vehicles, this speed is predicted by the LSTM network, while for the ego vehicle it is specified by the user. Predicting the future desired speed has two key benefits:

- It enables a realistic representation of agents in the traffic scenario by tailoring the cost function to the observed behavior.
- It facilitates the convergence of the algorithm to realistic GNE. For example, if a vehicle is slowing down to match the speed of the vehicle in front, it is likely intending to follow rather than overtake. The cost function is then adjusted online to reflect this behavior, ensuring convergence to the appropriate GNE.

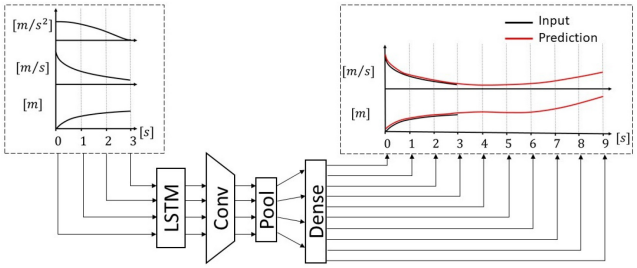


FIGURE 1. Architecture of the neural network for desired speed prediction.

By including the term $\|v_k^i - \hat{v}_k^i\|^2$ in the cost function of agent i, it becomes possible to better capture and understand the behavior of agent i. Alternatively, the desired speed could have been set as the maximum speed to encourage the vehicle to move: $\|v_k^i - v_{\max}\|^2$. However, this approach would fail to account for different driving behaviors. For example, some drivers may want to accelerate to the maximum speed, while others may prefer to maintain a slower speed or slow down to allow another vehicle to merge into the same lane.

Using $\|v_k^i - v_{\max}\|^2$ in the objective function would not capture these variations in behavior. In contrast, by using $\|v_k^i - \hat{v}_k^i\|^2$ and predicting the desired speed profile, the objective function of agent i reflects the influence of such diverse behaviors. In Section IV, the advantages of using the desired speed instead of the maximum speed in the cost function are discussed.

D. LSTM NEURAL NETWORK FOR BEHAVIOR PREDICTION

Figure 1 illustrates the architecture of the network used to predict the desired speed, along with its inputs and outputs. The network is composed by a first LSTM layer with 32 units, a 1D convolutional layer with 32 units, a Max pooling layer with a pool size of 2 and a final dense layer with 182 neurons. Further details can be found in [36].

The input consists of the vehicle's longitudinal behavior over the past 3-seconds, while the output describes the vehicle's behavior over a 9-second period, including the observed 3 seconds and the subsequent 6 seconds. The input features include progress, speed, and acceleration, and the output features consist of progress and speed. These features are sampled every 0.1 seconds, resulting in 30 time steps for the input and 90 time steps for the output.

The last 60 time steps of the predicted speed in output are inserted in the cost function expressed in equation (8), with the term $\hat{v}_k$.

1) DATASET

The model was trained on the NGSIM dataset [2], which includes trajectory data from two freeway segments and two arterial road segments. The dataset captures the movement of over 11,000 vehicles passing through these road sections.

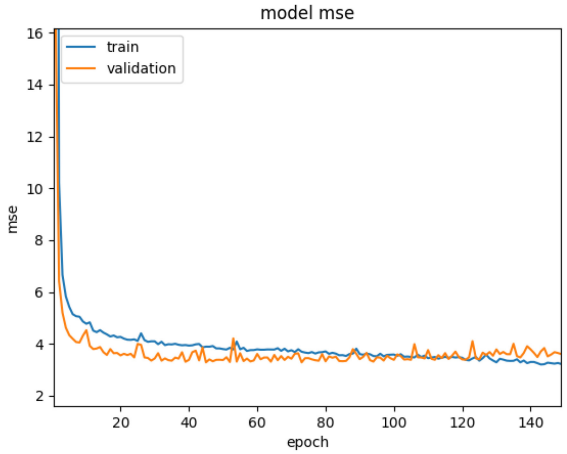


FIGURE 2. Learning curve of the LSTM network on the training and validation dataset [36].

TABLE 2. Hyperparameters of the training process for the LSTM network.

epochs	batch size	learning rate	optimizer
150	256	0.001	*ADAM*

The data includes vehicle position relative to a global reference system, position relative to the center lane, speed, acceleration, time headway, and space headway. However, only a subset of these variables, specifically progress, speed, and acceleration, was used to train the network.

Each trajectory varies in duration based on traffic conditions and vehicle speed. On average, the trajectories last 73.5 seconds, with a standard deviation of 32.9 seconds. As the training samples need to be 9 seconds long, multiple samples were extracted from each trajectory, resulting in a total of 734,000 samples used to train the network.

2) TRAINING PROCESS

The hyperparameters selected for the training process are shown in Table 2. The dataset was split into 70% for training, 20% for validation, and 10% for testing.

The LSTM network has been trained to minimize the mean squared error (MSE) in the longitudinal progress prediction. In Figure 2, the learning curve of the model is shown. The fact that the performance on the training dataset is close to the performance on the validation dataset ensures the absence of overfitting. The training session typically lasts around 4 hours.

3) PERFORMANCE

In Table 3, the performance of the LSTM network has been compared with some other prediction models present in literature [29], [37], [38]. It was necessary to establish a common performance metric for algorithms that predict different types of outcomes, such as position or longitudinal motion. The chosen performance measure is the root mean squared error (RMSE) of the final displacement (equation (10)) for models predicting position, while it is the RMSE of the final progress (equation (9)) for models predicting longitudinal motion, such as the LSTM module used in DeepGame-TP:

$$\text{RMSE}^{(1)} = \sqrt{\frac{1}{N_{test}} \sum_{i=1}^{N_{test}} \left(\int_0^T v_t^i \, dt - \int_0^T \hat{v}_t^i \, dt \right)^2} \tag{9}$$

$$\text{RMSE}^{(2)} = \sqrt{\frac{1}{N_{test}} \sum_{i=1}^{N_{test}} \left\| \tilde{\boldsymbol{x}}_T^i - \hat{\boldsymbol{x}}_T^i \right\|^2} \tag{10}$$

where $\tilde{\boldsymbol{x}}_T^i$ is the predicted final position of vehicle i, while $\hat{\boldsymbol{x}}_T^i$ is the true final position of vehicle i. All the algorithms are tested on the NGSIM dataset. The topology and traffic conditions depicted in the NGSIM dataset render trajectory prediction and longitudinal motion prediction quite comparable. In freeway scenarios, the lateral motion and its associated prediction error are significantly lower than those of longitudinal motion. Essentially, in the context of trajectory prediction on freeways, the primary source of error (by an order of magnitude) originates from predicting longitudinal motion [36]. This ensures a fair comparison between models, even if one model predicts only longitudinal motion while another predicts overall position.

Table 3 compares the performance of the LSTM network module with several state-of-the-art models. Below are some key considerations for each aspect of the table:

- Prediction: the field shows what is actually predicting the model, if the position (X, Y), the longitudinal coordinate Y or the progress $\int V$.
- History: it refers to the historical time window that the model receives as input.
- Input features: the features that each model receives as input in the historical temporal window. They can be features related to the target vehicle, like coordinates (X, Y), progress $\int V$, speed V, acceleration A, or interactions information with the traffic environment, like relative positions to the nearest vehicles or time headway.

Table 3 shows that the LSTM module of DeepGame-TP reaches the state-of-the-art performance in the prediction horizon 1-s, 2-s, 3-s, despite its simplicity and lack of information about vehicles surrounding the target one. However, its performance deteriorates in the 4-s and 5-s horizons (although still comparable with [37] in the 4 s horizon). This decline is attributed to the increased importance of interactions on longer horizons, where relying solely on the target vehicle's history becomes insufficient.

Concluding, the LSTM module achieves state-of-the-art performance, particularly within a 4-s prediction horizon, while maintaining good performance for longer-term predictions [36]. Moreover, the simple input structure enables application in scenarios where only the target vehicle is observable, without requiring data from surrounding vehicles. This is common in situations without V2I or V2V communication. In contrast, other models depend on the

TABLE 3. Comparison of RMSE values between the DeepGame-TP LSTM network and several state-of-the-art prediction algorithms, tested across different forecasting horizons on the NGSIM dataset, further details can be found in [36].

Paper	RMSE (m)					Input features						
	1 s	2 s	3 s	4 s	5 s	Pred.	Hist.	X, Y	$\int V$	V	A	Interactions
Attention LSTMs Lin et al. (2022, [21])	0.56					(X, Y)	3 s	X				X
Hierarchical LSTM Min et al. (2024, [22])	1.39	1.70	3.06	3.52	4.53	Y	5 s	X		X	X	X
PiP Song et al. (2020, [24])	0.55	1.18	1.94	2.88	4.04	(X, Y)	3 s	X				X
DeepGame-TP's LSTM network module	0.49	1.33	2.48	3.95	5.69	$\int V$	3 s		X	X	X	

history of all surrounding vehicles, which may not always be accessible.

E. CONSTRAINTS

In the GNEP presented in equation (3), only inequality constraints are considered. This is because the only relevant equality constraints for the trajectory planning problem, the dynamic constraints, are enforced through the integration function $X^i = F(U^i, \boldsymbol{x}_0^i)$, where $\boldsymbol{x}_0^i$ represents the initial state of vehicle i. The constraints for the GNEP of agent i are:

- Constraints on the inputs:

$$\begin{aligned} \delta_{min} \leq \delta_k^i \leq \delta_{max} \qquad & \forall\ k \\ F_{min} \leq F_k^i \leq F_{max} \qquad & \forall\ k \end{aligned} \tag{11}$$

  where δ_k^i and F_k^i are the steering angle and the longitudinal force of agent i at time step k.
- Constraints to stay in the lane:

$$\left\| \tilde{\boldsymbol{x}}_k^i - \tilde{\boldsymbol{x}}_k^C \right\|^2 \leq r_{lim} \qquad \forall\ k \tag{12}$$

  where $\tilde{\boldsymbol{x}}_k^i$ are the cartesian coordinates of agent i at time step k and $\tilde{\boldsymbol{x}}_k^C$ are the cartesian coordinates of the closest point on the center line.
- Constraints for collision avoidance:

$$\tilde{\boldsymbol{x}}_k^i \notin \Omega\left(\tilde{\boldsymbol{x}}_k^j, \psi_k^j\right) \qquad \forall\ k,\ \forall\ j \neq i \tag{13}$$

  where $\Omega(\tilde{\boldsymbol{x}}_k^j, \psi_k^j)$ represents the area of an ellipse centered on the agent j's position $\tilde{\boldsymbol{x}}_k^j$ at time step k and rotated in the direction of its heading ψ_k^j. A key feature is the asymmetry in how the penalty is applied between the vehicles. In a car-following scenario, the leading vehicle incurs no penalty for a collision, while the following vehicle receives the full penalty. This design prevents unrealistic behavior, such as the leading vehicle accelerating to avoid a rear-end collision caused by the following vehicle.

IV. CASE STUDIES AND DISCUSSION

To evaluate the approach of DeepGame-TP, demonstrate its flexibility and verify the effectiveness of using a learned-based model in cost function estimation, the following scenarios are tested and analyzed:

- Intersection scenario
- Car following scenario
- Overtaking scenario
- Merging scenario
- Congested intersection scenario

In each scenario, DeepGame-TP is compared with a baseline approach where the future desired speed, estimated by the LSTM network, is replaced by the maximum speed, thereby omitting the learned-based module. Each scenario is repeated 10 times to gather the necessary statistics. For the comparison, the following KPIs of the controlled vehicle are considered:

- Minimum distance from the closest vehicle (measured between the centers of gravity)
- Collision risk
- Average jerk
- Average speed
- Minimum acceleration
- Maximum acceleration

The quantification of the collision risk has been inspired by the work of [39]. The collision risk is defined as:

$$\text{Risk} = 1 - \Phi\left(\frac{\mu_{d_{min}} - d_{safe}}{\sigma_{d_{min}}}\right) \tag{14}$$

where $\mu_{d_{min}}$ and $\sigma_{d_{min}}$ represent the mean and standard deviation of the minimum distance to the closest vehicle during the episode. These values are calculated from the 10 repetitions of each episode. d_{safe} is a safety threshold, that depends on the scenario. The safety threshold varies depending on whether the minimum distance is reached when the cars are side by side or aligned one behind the other. To calculate the safety threshold, it is assumed a car length of $l = 4.0\ [m]$ and a width of $w = 1.7\ [m]$. The expression $\Phi(\cdot)$ denotes the cumulative function of the normal distribution.

The simulation environment is Automated Driving Open Research (ADORe), an open source modular software library and toolkit for decision making, planning, control and simulation of automated vehicles, developed by the Institute of Transportation Systems of the German Aerospace Center (DLR). In the following subsections, each scenario is analyzed individually, with discussions and considerations on the measured KPIs.

A. INTERSECTION

In the tested scenario, illustrated in Figure 3, the ego vehicle is the white one, which needs to turn left, while the red vehicle continues straight, creating a potential collision risk. The red vehicle is controlled by a simple Intelligent Driver Model (IDM). Two cases are tested: in the first case, the

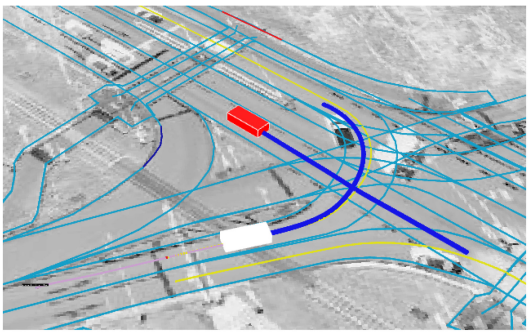


FIGURE 3. Intersection scenario.

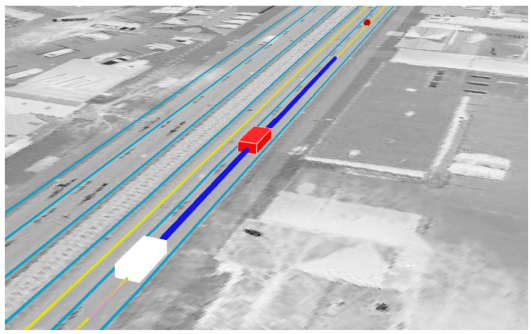


FIGURE 4. Car following scenario.

TABLE 4. Performance of DeepGame-TP (DeepG.) using the LSTM module and without it (baseline) in the yield case. The table shows the average value and the standard deviation of 10 runs, for each KPI.

	Min Dist $[m]$	Coll Risk	Avg Jerk $[m/s^3]$	Avg Vel $[m/s]$	Min Acc $[m/s^2]$	Max Acc $[m/s^2]$
DeepG.	5.3 ± 1.6	7.8%	1.2 ± 0.3	3.6 ± 0.2	−1.3± 0.5	2.0 ± 0.01
Baseline	3.9 ± 2.1	34.9%	1.2 ± 0.3	3.9 ± 0.4	−1.3± 0.5	2.0 ± 0.02

TABLE 5. Performance of DeepGame-TP (DeepG.) using the LSTM module and without it (baseline) in the proceed case. The table shows the average value and the standard deviation of 10 runs, for each KPI.

	Min Dist $[m]$	Coll Risk	Avg Jerk $[m/s^3]$	Avg Vel $[m/s]$	Min Acc $[m/s^2]$	Max Acc $[m/s^2]$
DeepG.	4.5 ± 0.8	3.8%	0.9 ± 0.3	3.9 ± 0.2	−0.7± 0.1	2.0 ± 0.01
Baseline	3.9 ± 0.6	6.1%	1.4 ± 0.4	3.3 ± 0.5	−1.1± 0.4	2.0 ± 0.01

red vehicle proceeds at a lower speed, allowing the ego vehicle to enter the intersection before it ("proceed case"). In the second case, the red vehicle proceeds at a higher speed, forcing the ego vehicle to complete its turn after the red vehicle has passed ("yield case"). Therefore, the aim is to test the difference in using the LSTM network to predict the desired speed when the ego vehicle can adopt a more aggressive approach versus when it needs to be more cautious.

In the two cases tested, if the LSTM network is not used, then the desired speed is replaced by the maximum speed in the cost function. A PID controller is used to follow the planned trajectories.

The threshold for the safety distance used in this scenario is $d_{safe} = 0.5\ (l + w)\ + 0.2 = 3.05\ [m]$, this is because, in this scenario, the minimum distance is reached when the cars are perpendicular to each other.

The results shown in Tables 4 and 5 lead to some considerations:

– In the yield scenario, using the LSTM module reduces the collision risk by a factor of five. Without the LSTM module, incorrect estimation of the desired speed causes the two vehicles to come dangerously close, increasing the collision risk.
– In the proceed scenario, using the LSTM network to predict the desired speed reduces the collision risk by half. In this case, the red vehicle does not accelerate to its maximum speed, allowing the ego vehicle to enter the intersection first. With the LSTM network, this intention is accurately interpreted, enabling the ego vehicle to respond appropriately. Without the LSTM network, the red vehicle is incorrectly predicted to accelerate, leading to confusion for the ego vehicle, which attempts to enter the intersection only after the red vehicle has passed. This discrepancy in speed prediction significantly increases the collision risk, as reflected by the corresponding KPI.

B. CAR FOLLOWING

The objective of this scenario is to demonstrate that DeepGame-TP can effectively handle standard car-following situations while improving safety compared to the baseline model, which does not include the LSTM module. The scenario, shown in Figure 4, is divided into two phases:

– The ego vehicle follows a leading vehicle, starting at a higher speed and then adjusting its speed to match the leading vehicle's speed.
– At the end of the scenario, the leading vehicle comes to a stop, and the ego vehicle must also stop while maintaining a safe distance behind it.

The safety distance threshold used in this scenario is defined as $d_{safe} = 0.5\ (l + l)\ + 0.2 = 4.20\ [m]$, since the vehicles are longitudinally aligned when the minimum distance is achieved.

Table 6 presents the KPIs for the car-following scenario. It is clear that DeepGame-TP provides a more accurate estimate of longitudinal motion, leading to a higher level of safety compared to the baseline, where the desired speed is simply set to the maximum. In the final phase, when the leading vehicle comes to a stop, DeepGame-TP successfully recognizes this behavior and maintains a safer following distance. In contrast, the baseline model predicts that the leading vehicle will accelerate, resulting in a higher

TABLE 6. Performance of DeepGame-TP (DeepG.) using the LSTM module and without it (baseline) in the car following scenario. The table shows the average value and the standard deviation of 10 runs, for each KPI.

	Min Dist $[m]$	Coll Risk	Avg Jerk $[m/s^3]$	Avg Vel $[m/s]$	Min Acc $[m/s^2]$	Max Acc $[m/s^2]$
DeepG.	7.9 ± 1.6	1.0%	1.1 ± 0.3	3.9 ± 0.1	-1.3 ± 0.3	2.0 ± 0.02
Baseline	5.1 ± 1.4	26.0%	1.3 ± 0.4	4.0 ± 0.1	-1.7 ± 0.5	2.0 ± 0.02

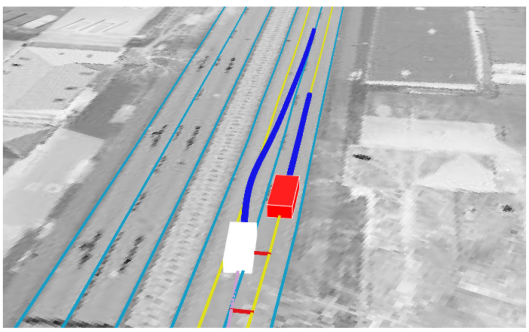
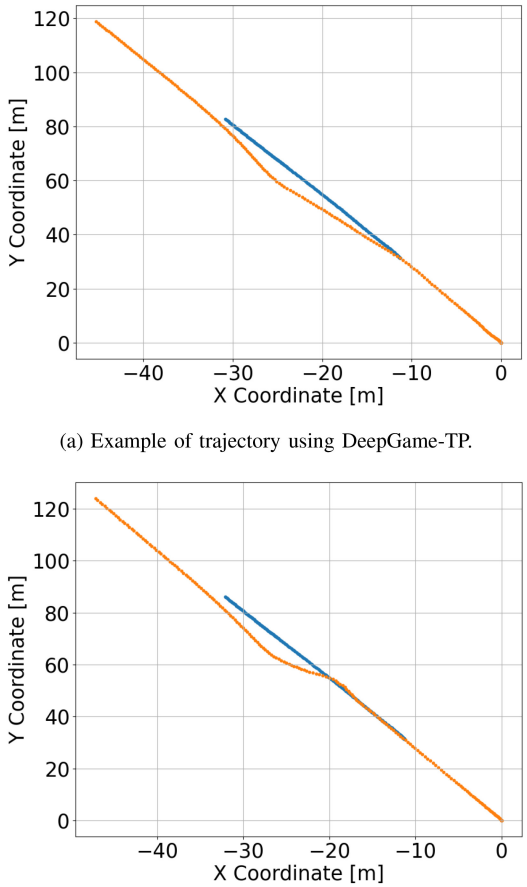


FIGURE 5. Overtaking scenario.

TABLE 7. Performance of DeepGame-TP (DeepG.) using the LSTM module and without it (Baseline) in the overtaking scenario. The table shows the average value and the standard deviation of 10 runs, for each KPI.

	Min Dist $[m]$	Coll Risk	Avg Jerk $[m/s^3]$	Avg Vel $[m/s]$	Min Acc $[m/s^2]$	Max Acc $[m/s^2]$
DeepG.	2.8 ± 0.1	0.0%	1.6 ± 0.4	6.7 ± 0.2	-1.5 ± 0.3	2.0 ± 0.01
Baseline	2.3 ± 0.3	98.9%	1.3 ± 0.4	6.7 ± 0.2	-2.0 ± 0.2	2.0 ± 0.01



(a) Example of trajectory using DeepGame-TP.

(b) Example of Trajectory without the LSTM module (baseline).

FIGURE 6. Example of trajectories if the LSTM module is used (a) and if it is not used (b) for the overtaking scenario. The ego vehicle trajectory is in orange. In (a), the inclusion of the LSTM module results in a smoother trajectory, allowing the overtaking maneuver to occur earlier. In contrast, (b), shows that the baseline model struggles to accurately estimate the longitudinal motion of the leading vehicle, causing a delayed emergency overtaking maneuver and increasing the risk of collision.

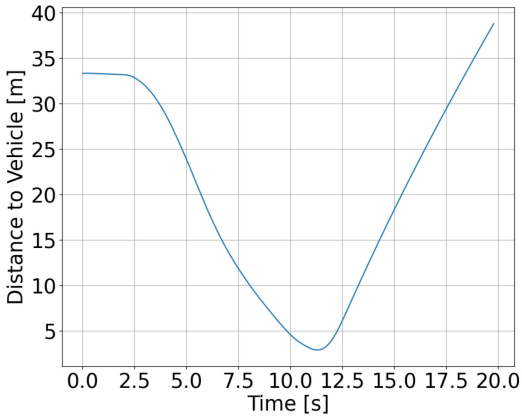
speed trajectory and requiring an emergency brake when the leading vehicle unexpectedly stops.

C. OVERTAKING

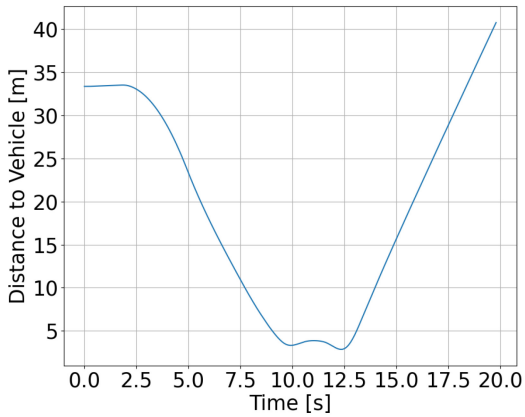
Figure 5 shows the simulated overtaking scenario. In this scenario, the ego vehicle has a desired speed equal to the maximum allowed, while the red vehicles proceeds with a speed that is way lower. For this reason, the GNE converges to an overtaking maneuver. The ego vehicles is controlled by DeepGame-TP, while the red vehicle is controlled by an IDM.

Table 7 presents the KPI results. It is evident that without the LSTM module, and thus without an accurate estimate of the vehicle's future trajectory, the risk of collision significantly increases. This is more clearly illustrated in Figure 6, which shows the trajectories in both scenarios, and in Figure 7, which displays the distance between the vehicles over time. From these two figures, the following observations can be made:

– With the LSTM module, the overtaking maneuver is significantly smoother. During the bumper-to-bumper phase, the distance is maintained, and the minimum distance is achieved in the final phase of the maneuver, when the vehicles are side by side. This results in a minimum distance of 2.8 meters between the centers of gravity, which is acceptable (approximately 1 meter of door-to-door distance). Since the minimum distance is reached in this phase of the maneuver, for the risk assessment the safety distance used is $d_{safe} = 0.5 (w + w) + 0.2 = 1.9$ $[m]$.
– Without the LSTM module, the overtaking maneuver is abrupt. The front vehicle is predicted to accelerate to the maximum allowed speed, which is an incorrect estimate. This brings the ego vehicle too close to the front vehicle, causing the overtaking maneuver to be initiated with significant delay and urgency. The minimum distance is reached not only during the side-by-side phase but also during the bumper-to-bumper phase, leading to a collision. A distance of approximately 2.3 meters between the centers of gravity is indeed insufficient for maintaining adequate bumper-to-bumper space. Since the minimum distance is reached in this phase of the



(a) Distance to vehicle using DeepGame-TP.



(b) Distance to vehicle without the LSTM module (baseline).

FIGURE 7. Example of distance between the ego vehicle and the other one over time during the overtaking scenario if the LSTM module is used (a) and if it is not used (b). As shown in (b), the minimum distance is reached much earlier in the maneuver compared to the other case, dropping to below 4 meters when the cars are aligned one behind the other, which increases the risk of collision. In contrast, in (a), the minimum distance, approximately 2.5 meters, is reached later, when the cars are side by side, making it acceptable in terms of safety.

maneuver, for the risk assessment the safety distance used is $d_{safe} = 0.5\ (l + l)\ + 0.2 = 4.2\ [m]$.

D. MERGING

Figure 8 illustrates the merging scenario used to test DeepGame-TP. In this scenario, the ego vehicle merges into the left lane where two other vehicles are present. The configuration of the scene allows the ego vehicle to choose between merging ahead of the two cars by accelerating or merging between them by decelerating. The two vehicles are controlled by an IDM with their target speed set to the maximum speed. Since the minimum distance is reached when the cars are side by side, for the risk assessment the safety distance used is $d_{safe} = 0.5\ (w + w)$ $+0.2 = 1.9\ [m]$.

In scenarios where vehicles are traveling near the maximum speed, selecting this speed as the desired speed in the cost function for the other agents is a reasonable assumption. This explains why there is no significant performance difference between DeepGame-TP and the version without the LSTM module, as shown in Table 8.

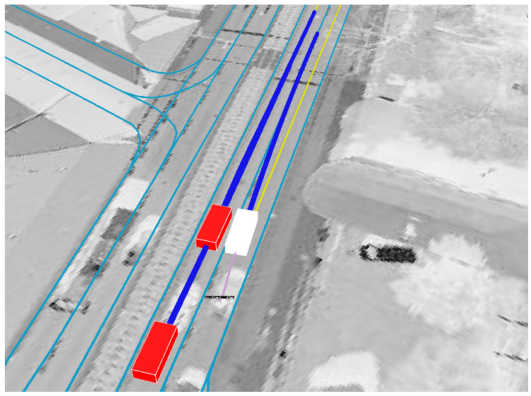


FIGURE 8. Merging scenario.

TABLE 8. Performance of DeepGame-TP (DeepG.) using the LSTM module and without it (baseline) in the merging scenario. The table shows the average value and the standard deviation of 10 runs, for each KPI.

	Min Dist $[m]$	Coll Risk	Avg Jerk $[m/s^3]$	Avg Vel $[m/s]$	Min Acc $[m/s^2]$	Max Acc $[m/s^2]$
DeepG.	2.8 ± 0.3	0.13%	1.9 ± 0.7	5.1 ± 0.3	-0.8 ± 0.3	2.0 ± 0.01
Baseline	2.7 ± 0.4	2.28%	1.7 ± 0.8	5.3 ± 0.2	-0.5 ± 0.2	2.0 ± 0.01

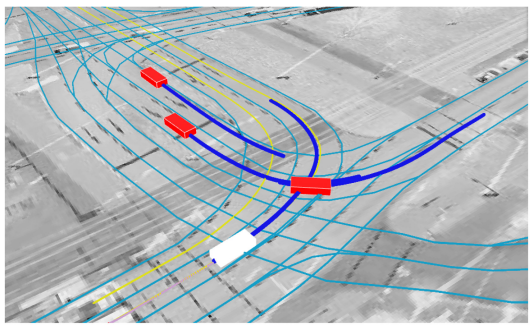


FIGURE 9. Congested intersection scenario.

E. CONGESTED INTERSECTION

Figure 9 illustrates the congested unsignalized intersection scenario, designed to push the algorithm to its limits by increasing the complexity of the GNEP with the addition of four vehicles. In this scenario, three vehicles make left turns at the intersection, two in one lane and one in the adjacent lane. These vehicles are controlled by an IDM. The ego vehicle must cross the paths of these vehicles, deciding whether to enter the intersection during the gaps in traffic flow. The safety distance used for the risk assessment is the same of the intersection scenario: $d_{safe} = 0.5\ (l+w)\ +0.2 = 3.05\ [m]$.

Table 9 shows the performance of DeepGame-TP compared to the baseline. In this case as well, the desired speed of

TABLE 9. Performance of DeepGame-TP (DeepG.) using the LSTM module and without it (baseline) in the congested intersection scenario. The table shows the average value and the standard deviation of 10 runs, for each KPI.

	Min Dist $[m]$	Coll Risk	Avg Jerk $[m/s^3]$	Avg Vel $[m/s]$	Min Acc $[m/s^2]$	Max Acc $[m/s^2]$
DeepG.	5.1 ± 1.2	4.32%	2.4 ± 0.3	4.0 ± 0.1	−1.1± 0.1	2.0 ± 0.02
Baseline	4.5 ± 0.8	4.02%	1.9 ± 0.4	3.9 ± 0.1	−1.0± 0.2	2.0 ± 0.01

the vehicles in the IDM model is set to the maximum speed. This explains the lack of significant performance differences between DeepGame-TP and the baseline. The baseline's error in predicting the desired speed is not particularly observable in this scenario.

The algorithm's ability to manage this four-vehicle scenario without a noticeable reduction in safety is remarkable. Another significant achievement is that the computational time to solve the GNEP, even with approximation, consistently stays under 100 milliseconds on average, as will be demonstrated in the next section.

V. REAL TIME COMPUTATIONAL PERFORMANCE

In this section, considerations on execution time and online applicability are presented. Figures 10, 11, 12, 13, 14 show histograms of the execution time for each episode. Various strategies have been implemented to reduce computational time:

- The initial guess U_0 in the Trust Region solver (Algorithm 1) is the solution from the previous time step. This accounts for the bimodal distribution observed particularly in the histograms in Figures 12, 13 and 14. The first peak at low execution times occurs when the traffic situation remains similar to the previous time step, meaning the initial guess is close to the actual GNEP solution. The second peak at higher execution times arises when the previous time step's solution is no longer valid for the current situation, requiring more time to solve the GNEP.
- The computation of the gradient ∇L has been parallelized across multiple cores.
- The number of integration nodes (N) has been set to 12, with the integration time step configured to 0.5 seconds.
- The maximum number of iterations has been set to 25.

Table 10 shows the solve time of DeepGame-TP compared to ALGAMES and LUCIDGames [30], [33], in comparable scenarios. Given the available computational power, DeepGame-TP is comparable to the state of the art in terms of real-time applicability. In the four-vehicle scenario, DeepGame-TP demonstrates superior computational efficiency compared to state-of-the-art game theory-based algorithms. This performance gain is partly attributed to limiting the solution to 25 iterations. Despite this approximation, the collision risk remains within acceptable bounds.

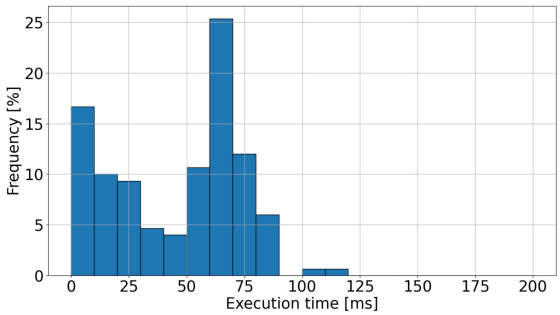


FIGURE 10. Execution time for the intersection scenario.

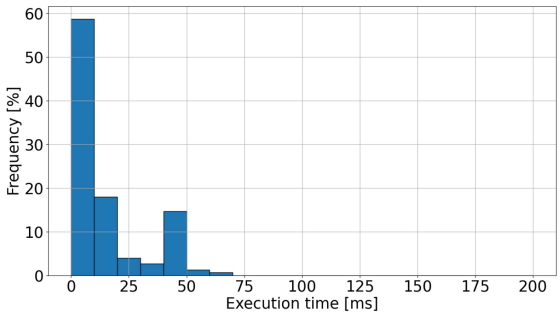


FIGURE 11. Execution time for the car following scenario.

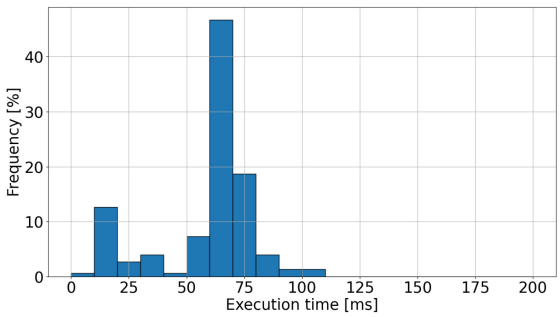


FIGURE 12. Execution time for the overtaking scenario.

Figures 10, 12 and 11 show that the computational time is always below 100 milliseconds for each scenario with two agents. In the merging scenario with three agents, the computational time increases to between 100 and 150 milliseconds for about 30% of the run time. In the congested intersection scenario with four agents, the computational time ranges between 100 and 150 milliseconds for approximately 50% of the run time, while for the remaining time, it stays below 100 milliseconds. Despite these fluctuations, the overall average remains below 100 milliseconds. (Table 10).

In this work, all the experiments have been executed on a 8-core processor (Intel® Core™ i7-11850H).

VI. IMPLEMENTATION AND SYSTEM INTEGRATION

This section presents the implementation of DeepGame-TP within the ADORe software environment, with a particular emphasis on its integration into the ROS 2 framework. ROS stands for Robot Operating System, it is a widely used

TABLE 10. Comparison of computational times between DeepGame-TP and other algorithms.

Algorithm	intersection	overtaking	merging	congested scenario	# cores
	2 players	2 players	3 players	4 players	
DeepGame-TP	46 ± 27 $[ms]$	58 ± 22 $[ms]$	52 ± 48 $[ms]$	$87 \pm 29[ms]$	8
ALGAMES [30]	50 ± 11 $[ms]$	-	89 ± 14 $[ms]$	$509 \pm 33[ms]$	-
LUCID [33]	-	-	26 ± 37 $[ms]$	$87 \pm 94[ms]$	16

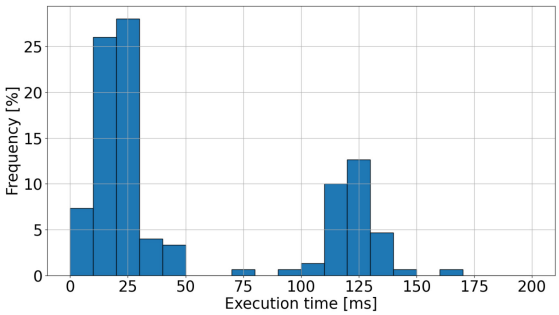


FIGURE 13. Execution time for the merging scenario.

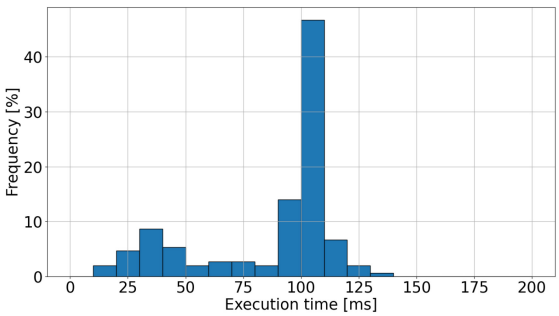


FIGURE 14. Execution time for the congested intersection scenario.

framework for robotics software [40]. The core components of ROS implementations are nodes, messages, topics. Nodes are processes that perform computation. A system is typically composed by many nodes, each node can be considered as a software module. Nodes communicate with each other by passing messages. A message is a strictly typed data structure, it can be composed of other messages, and arrays of other messages, nested arbitrarily deep. A node sends a message by publishing it to a given topic, which is simply a string, while it receives the information contained in a message by subscribing to its specific topic [40]. Topics can be viewed as named communication channels that function like 'pipes' for transmitting messages.

Figure 15 shows the architecture of DeepGame-TP within the ROS 2 framework. In the figure, topics are italicized and represented by arrows, while nodes are in bold and enclosed in rectangles. The topics are:

– *view:* the messages that are published and subscribed to this topic contain the information about the current traffic situation: the state of each vehicle and its associated center, left and right lanes. This topic is published by modules upstream of DeepGame-TP, such as the sensor data fusion module when the software is deployed on a real vehicle, or the simulation environment module in case of a simulation. The nodes that subscribe to this topic are the Recorder node and the Dynamic Game Solver node.
– *history:* the messages published and subscribed to this topic contain the trajectories of each vehicle present in the traffic scenario over the past three seconds. This topic is published by the Recorder node and is subscribed to by the LSTM Module node.
– *behavior_prediction:* the messages published and subscribed to this topic contain the predicted speed profile and progress for the next six seconds of each vehicle. This topic is published by the LSTM Module node and is subscribed to by the Dynamic Game Solver node.
– *planned_trajectory:* the messages published and subscribed to this topic contain the planned trajectory of the ego vehicle. This topic is published by the Dynamic Game Solver node and subscribed to by the Controller downstream in the chain.

The architecture of DeepGame-TP is composed by three nodes:

– *Recorder:* this node records the trajectories of each vehicle in the traffic scenario over the past three seconds and publishes them to a topic. It subscribes to the topic *view* and publishes to the topic *history*.
– *LSTM Module:* this node contains the LSTM neural network employed to predict the longitudinal behavior of each vehicle. It subscribes to the topic *history* and publishes to the topic *behavior_prediction*.
– *Dynamic Game Solver:* this node solves the Generalized Nash Equilibrium of the dynamic game. It receives input regarding the current traffic state (topic *view*) and the predicted longitudinal behavior of each vehicle (topic *behavior_prediction*), and it publishes the planned trajectory for the ego vehicle to the topic *planned_trajectory*.

The ROS 2 framework facilitates easy interfacing of DeepGame-TP with upstream and downstream modules through messages published and subscribed to specific topics. To generate a trajectory, upstream modules, either a simulation environment or a sensor data fusion module for real world applications, must publish to the topic *view*. The planned trajectory is published to the topic *planned_trajectory*, which is subscribed to by a downstream controller, either PID or model predictive, that executes the trajectory. For safety reasons, the downstream controller must

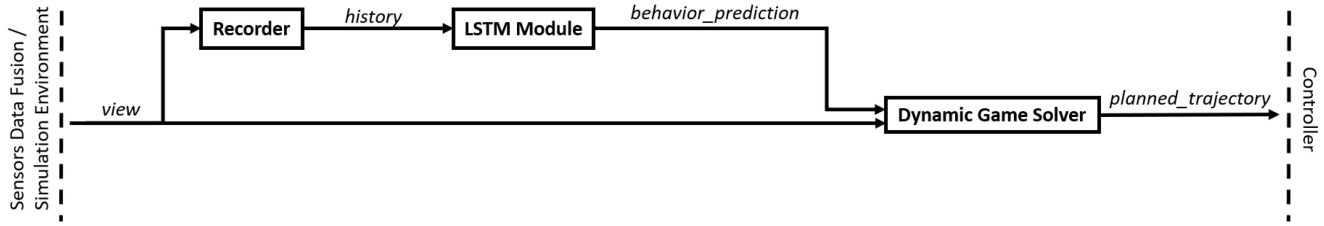


FIGURE 15. Architecture of DeepGame-TP within the ROS 2 framework.

verify that the planned trajectory is feasible, does not violate constraints, and that the vehicle is within its Operational Design Domain (ODD). If any of these conditions are not met, the controller must execute a minimum risk maneuver.

VII. ALGORITHM DISCUSSION

Finding a Nash equilibrium is generally a non-convex problem. In complex interaction spaces, such as those encountered in autonomous driving with collision avoidance constraints, the cost functions and constraints are often highly nonlinear and non-convex. Consequently, global convergence to a generalized Nash equilibrium cannot be guaranteed. On the other hand, the Trust Region algorithm ensures local convergence [41], as it dynamically adjusts the step size and approximates the problem using a quadratic model within a localized region. By iteratively solving these localized sub-problems and updating the trust region radius based on the agreement between predicted and actual reductions in the objective function, the algorithm converges to a stationary point. However, this convergence is local, meaning it depends on the quality of the initial guess and does not guarantee reaching a global Nash equilibrium in the presence of non-convexities.

The maximum number of iterations in Algorithm 1, denoted as k_{max}, enables real-time applications. If convergence is not achieved within k_{max} iterations, the approximated generalized Nash equilibrium is less accurate. Figures 10–14 show cases where convergence has not been reached, as indicated by the bimodal distribution of computation times. When data cluster around 100–150 ms, it suggests that the maximum number of iterations has been reached. Even in the most challenging scenario, the congested intersection, where convergence criteria are frequently unmet, the approximated solution remains valid, as safety performance does not deteriorate.

VIII. CONCLUSION

This work introduces DeepGame-TP, a trajectory planner for multi-agent traffic environments that solves the Generalized Nash Equilibrium Problem using the Augmented Lagrangian formulation and a Trust Region solver. The speed component of each agent's cost function is learned by an LSTM network, which predicts the desired speed profile for the next 6 seconds. Case studies demonstrate that the learning-based cost function approach of DeepGame-TP outperforms the traditional approach, where the desired speed is fixed at the maximum speed, especially in intersection and overtaking scenarios. DeepGame-TP enables the ego vehicle to adapt to observed behaviors, understanding the longitudinal aggressiveness of agents and adjusting their cost functions accordingly. As a result, DeepGame-TP offers a transparent approach to trajectory planning in highly interactive scenarios such as intersections, overtaking, car following and merging, without renouncing deep learning to model the agents' cost function. Simulation campaigns demonstrate the approach's flexibility, as it is not restricted to any specific topology, and its potential for real-time application, with the ability to handle scenarios involving up to four agents within 150 milliseconds. Further advancements can be made by exploring the use of learning-based models to improve understanding and modeling within game-theoretic frameworks, with parallel application to real vehicles.

REFERENCES

[1] T. Räuker, A. Ho, S. Casper, and D. Hadfield-Menell, "Toward transparent ai: A survey on interpreting the inner structures of deep neural networks," 2023, *arXiv:2207.13243*.

[2] 2016, "Next generation simulation (NGSIM) vehicle trajectories and supporting data," Dataset, U.S. Department of Transportation Federal Highway Administration. [Online]. Available: http://doi.org/10.21949/1504477

[3] J. Betz et al., "Autonomous vehicles on the edge: A survey on autonomous vehicle racing," *IEEE Open J. Intell. Transp. Syst.*, vol. 3, pp. 458–488, 2022. [Online]. Available: http://dx.doi.org/10.1109/ojits.2022.3181510

[4] Z. Wang, J. Guo, Z. Hu, H. Zhang, J. Zhang, and J. Pu, "Lane transformer: A high-efficiency trajectory prediction model," *IEEE Open J. Intell. Transp. Syst.*, vol. 4, pp. 2–13, 2023.

[5] R. Trauth, K. Moller, and J. Betz, "Toward safer autonomous vehicles: Occlusion-aware trajectory planning to minimize risky behavior," *IEEE Open J. Intell. Transp. Syst.*, vol. 4, pp. 929–942, 2023.

[6] D. I. Tselentis and E. Papadimitriou, "Driver profile and driving pattern recognition for road safety assessment: Main challenges and future directions," *IEEE Open J. Intell. Transp. Syst.*, vol. 4, pp. 83–100, 2023.

[7] L. Vasile, N. Dinkha, B. Seitz, C. Däsch, and D. Schramm, "Comfort and safety in conditional automated driving in dependence on personal driving behavior," *IEEE Open J. Intell. Transp. Syst.*, vol. 4, pp. 772–784, 2023.

[8] K. R. Williams et al., "Trajectory planning with deep reinforcement learning in high-level action spaces," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 59, no. 3, pp. 2513–2529, Jun. 2023.

[9] E. Zhang, R. Zhang, and N. Masoud, "Predictive trajectory planning for autonomous vehicles at intersections using reinforcement learning," *Transp. Res. Part C, Emerg. Technol.*, vol. 149, Apr. 2023, Art. no. 104063. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0968090X23000529

[10] Z. Wang, J. Tu, and C. Chen, "Reinforcement learning based trajectory planning for autonomous vehicles," in *Proc. China Autom. Congr. (CAC)*, 2021, pp. 7995–8000.

[11] Z. Li, K. You, J. Sun, and G. Wang, "Informative trajectory planning using reinforcement learning for minimum-time exploration of spatiotemporal fields," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 35, no. 12, pp. 17216–17226, Dec. 2024.

[12] Y.-C. Ni, V. L. Knoop, J. F. P. Kooij, and B. van Arem, "Adaptive cruise control utilizing noisy multi-leader measurements: A learning-based approach," *IEEE Open J. Intell. Transp. Syst.*, vol. 5, pp. 251–264, 2024.

[13] M. Zare, P. M. Kebria, A. Khosravi, and S. Nahavandi, "A survey of imitation learning: Algorithms, recent developments, and challenges," 2023, *arXiv:2309.02473*.

[14] L. Le Mero, D. Yi, M. Dianati, and A. Mouzakitis, "A survey on imitation learning techniques for end-to-end autonomous vehicles," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 9, pp. 14128–14147, Sep. 2022.

[15] F. Codevilla, E. Santana, A. M. Lopez, and A. Gaidon, "Exploring the limitations of behavior cloning for autonomous driving," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, 2019, pp. 9328–9337.

[16] T. Phan-Minh et al., "DriveIRL: Drive in real life with inverse reinforcement learning," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2023, pp. 1544–1550.

[17] Z. Huang, H. Liu, J. Wu, and C. Lv, "Conditional predictive behavior planning with inverse reinforcement learning for human-like autonomous driving," *IEEE Trans. Intell. Transp. Syst.*, vol. 24, no. 7, pp. 7244–7258, Jul. 2023.

[18] Z. Zhao et al., "Personalized car following for autonomous driving with inverse reinforcement learning," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, 2022, pp. 2891–2897.

[19] G. C. K. Couto and E. A. Antonelo, "Generative adversarial imitation learning for end-to-end autonomous driving on urban environments," in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI)*, 2021, pp. 1–7.

[20] R. Bhattacharyya et al., "Modeling human driving behavior through generative adversarial imitation learning," *IEEE Trans. Intell. Transp. Syst.*, vol. 24, no. 3, pp. 2874–2887, Mar. 2023.

[21] A. Jamgochian, E. Buehrle, J. Fischer, and M. J. Kochenderfer, "SHAIL: Safety-aware hierarchical adversarial imitation learning for autonomous driving in urban environments," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2023, pp. 1530–1536.

[22] A. Plebe, H. Svensson, S. Mahmoud, and M. Da Lio, "Human-inspired autonomous driving: A survey," *Cogn. Syst. Res.*, vol. 83, Jan. 2024, Art. no. 101169. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389041723001031

[23] C.-C. Hsu, L.-W. Kang, S.-Y. Chen, I.-S. Wang, C.-H. Hong, and C.-Y. Chang, "Deep learning-based vehicle trajectory prediction based on generative adversarial network for autonomous driving applications," *Multimedia Tools Appl.*, vol. 82, pp. 10763–10780, Mar. 2022.

[24] L. Zhao, Y. Liu, A. Y. Al-Dubai, A. Y. Zomaya, G. Min, and A. Hawbani, "A novel generation-adversarial-network-based vehicle trajectory prediction method for intelligent vehicular networks," *IEEE Internet Things J.*, vol. 8, no. 3, pp. 2066–2077, Feb. 2021.

[25] X. Chen, J. Xu, R. Zhou, W. Chen, J. Fang, and C. Liu, "Trajvae: A variational AutoEncoder model for trajectory generation," *Neurocomputing*, vol. 428, pp. 332–339, Mar. 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231220312017

[26] Y. Choi, R. C. Mercurius, S. M. A. Shabestary, and A. Rasouli, "DICE: Diverse diffusion model with scoring for trajectory prediction," 2023, *arXiv:2310.14570*.

[27] I. Bae, Y.-J. Park, and H.-G. Jeon, "Singulartrajectory: Universal trajectory predictor using diffusion model," 2024, *arXiv:2403.18452*.

[28] K. Chen, X. Chen, Z. Yu, M. Zhu, and H. Yang, "EquiDiff: A conditional equivariant diffusion model for trajectory prediction," 2023, *arXiv:2308.06564*.

[29] L. Lin, W. Li, H. Bi, and L. Qin, "Vehicle trajectory prediction using LSTMs with spatial–temporal attention mechanisms," *IEEE Intell. Transp. Syst. Mag.*, vol. 14, no. 2, pp. 197–208, Mar./Apr. 2022.

[30] S. Le Cleac'h, M. Schwager, and Z. Manchester, "ALGAMES: A fast augmented lagrangian solver for constrained dynamic games," *Auton. Robots*, vol. 46, no. 1, pp. 201–215, Jan. 2022. [Online]. Available: https://doi.org/10.1007/s10514-021-10024-7

[31] W. Schwarting, A. Pierson, J. Alonso-Mora, S. Karaman, and D. Rus, "Social behavior for autonomous vehicles," *Proc. Nat. Acad. Sci.*, vol. 116, no. 50, pp. 24972–24978, 2019. [Online]. Available: https://www.pnas.org/doi/abs/10.1073/pnas.1820676116

[32] M. Bhatt, Y. Jia, and N. Mehr, "Efficient constrained multi-agent trajectory optimization using dynamic potential games," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, 2023, pp. 7303–7310. [Online]. Available: https://doi.org/10.1109/IROS55552.2023.10342328

[33] S. Le Cleac'h, M. Schwager, and Z. Manchester, "LUCIDGames: Online unscented inverse dynamic games for adaptive trajectory prediction and planning," *IEEE Robot. Autom. Lett.*, vol. 6, no. 3, pp. 5485–5492, Jul. 2021.

[34] F. Facchinei and C. Kanzow, "Generalized Nash equilibrium problems," *Annals Oper. Res.*, vol. 175, no. 1, pp. 177–211, Mar. 2010. [Online]. Available: https://doi.org/10.1007/s10479-009-0653-x

[35] N. Andrei, "The trust-region method," in *Modern Numerical Nonlinear Optimization* (Springer Optimization and its Applications). Cham, Switzerland: Springer Int. Publ., Jun. 2022, ch. 8, pp. 331–353. [Online]. Available: https://doi.org/10.1007/978-3-031-08720-2_8

[36] G. Lucente, M. S. Maarssoe, I. Kahl, and J. Schindler, "Deep learning algorithms for longitudinal driving behavior prediction: A comparative analysis of convolutional neural network and long–short-term memory models," *SAE Int. J. Connect. Autom. Veh.*, vol. 7, pp. 1–16, Jun. 2024. [Online]. Available: https://doi.org/10.4271/12-07-04-0025

[37] H. Min, X. Xiong, P. Wang, and Z. Zhang, "A hierarchical LSTM-based vehicle trajectory prediction method considering interaction information," *Autom. Innov.*, vol. 7, no. 1, pp. 71–81, Feb. 2024. [Online]. Available: https://doi.org/10.1007/s42154-023-00261-0

[38] H. Song, W. Ding, Y. Chen, S. Shen, M. Y. Wang, and Q. Chen, "PiP: Planning-informed trajectory prediction for autonomous driving," in *Proc. 16th Eur. Conf. Comput. Vis. (ECCV)*, 2020, pp. 598–614.

[39] L. Gharavi, A. Dabiri, J. Verkuijlen, B. D. Schutter, and S. Baldi, "Proactive emergency collision avoidance for automated driving in highway scenarios," 2024, *arXiv:2310.17381*.

[40] M. Quigley et al., "ROS: An open-source robot operating system," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2009, pp. 1–6. [Online]. Available: https://api.semanticscholar.org/CorpusID:6324125

[41] J. Nocedal and S. J. Wright, *Numerical Optimization* (Springer Series in Operations Research and Financial Engineering). New York, NY, USA: Springer Nat., 2006, pp. 1–664.

GIOVANNI LUCENTE received the B.S. degree in mechanical engineering and the M.S. degree in mechanical engineering from the Politecnico di Milano, Italy, in 2017 and 2020, respectively. With previous working experience in automotive companies, he is currently pursuing the Ph.D. degree from Technische Universität Berlin, Berlin and working at the Institute of Transportation Systems, German Aerospace Center (DLR), Braunschweig. His research interests revolve around decision-making processes and cooperation between automated and connected vehicles in urban traffic environments.

MIKKEL SKOV MAARSSOE was born in Denmark, Odense, in 1997. He received the B.S. degree in robotics engineering and the M.S. degree in robotics engineering focused on advanced robotics technology from the University of Southern Denmark (SDU), Odense, Denmark, in 2021 and 2023, respectively. From 2019 to 2023, he worked as an ambassador for the Robotics Engineering Education with SDU, and has among other teaching roles been an instructor in the course embodied artificial intelligence with SDU in 2022. He is currently working as a Researcher of Autonomous Driving with the German Aerospace Center (DLR), with a focus on computer science, control engineering, and software development.

SANATH HIMASEKHAR KONTHALA received the B.S. degree in mechanical engineering and the M.S. degree in automotive engineering from FH Aachen in 2019 and 2023, respectively. He is currently a Research Assistant with the German Aerospace Center (DLR), focusing on autonomous driving. His research interests include trajectory planning and control, as well as cooperative automated driving.

ANAS ABULEHIA received the bachelor's degree in mechatronics engineering from Palestine Polytechnic University, Palestinian Territories, in 2016, and the Master of Engineering degree in embedded systems from Fachhochschule Dortmund in 2023. With extensive experience as a mechatronics engineer, he is currently pursuing the Ph.D. degree with the Institute of Transportation Systems, German Aerospace Center (DLR), Braunschweig. His research interests include autonomous systems, vehicles, and connected vehicles, and transportation.

REZA DARIANI received the B.S. degree in power electrical engineering from the University of Saveh, Iran, in 2008, the first M.S. degree in electronic-electrical and automatic from the University of Reims, France in 2010, the second M.S. degree in mechatronics from the University of Strasbourg, France, in 2011, and the Ph.D. degree in trajectory planning for autonomous vehicles from the Otto-von-Guericke Universität of Magdeburg, Germany. He is currently a Professor of Signals and Systems with Hochschule Merseburg. Since 2016, he has been working as a Researcher with the Institute of Transportation Systems, German Aerospace Center, Braunschweig, Germany.

JULIAN SCHINDLER has been working with the Institute of Transportation Systems, German Aerospace Center (DLR-ITS), Braunschweig, Germany, since 2006. Since 2017, he has been also leading the Group "System-Automation and Integration," DLR-ITS. As a computer scientist, he was first responsible for the software architecture of several driving simulators. Working on the ergonomic design of vehicle automation functions and the cooperation between driver and vehicle, he was participating in several national and international projects, such as Citymobil (FP6), interactIVe, HAVEit, and ISiPADAS (FP7). Since cooperation afterwards also included other agents, e.g., infrastructure or VRUs, he focussed on that topic, coordinating, e.g., H2020-MAVEN and H2020-TransAID.