# Event-Triggered Model Predictive Control With Deep Reinforcement Learning for Autonomous Driving

Fengying Dang, Dong Chen, *Member, IEEE*, Jun Chen, *Senior Member, IEEE*, and Zhaojian Li, *Senior Member, IEEE*

***Abstract*—Event-triggered model predictive control (eMPC) is a popular optimal control method with an aim to alleviate the computation and/or communication burden of MPC. However, it generally requires *a priori* knowledge of the closed-loop system behavior along with the communication characteristics for designing the event-trigger policy. This paper attempts to solve this challenge by proposing an efficient eMPC framework and demonstrates successful implementation of this framework on the autonomous vehicle path following. First of all, a model-free reinforcement learning (RL) agent is used to learn the optimal event-trigger policy without the need for a complete dynamical system and communication knowledge in this framework. Furthermore, techniques including prioritized experience replay (PER) buffer and long short-term memory (LSTM) are employed to foster exploration and improve training efficiency. In this paper, we use the proposed framework with three deep RL algorithms, i.e., Double Q-learning (DDQN), Proximal Policy Optimization (PPO), and Soft Actor-Critic (SAC), to solve this problem. Results show that all three deep RL-based eMPC (deep-RL-eMPC) can achieve better evaluation performance than the conventional threshold-based and previous linear Q-based approach in the autonomous path following. In particular, PPO-eMPC with LSTM and DDQN-eMPC with PER and LSTM obtain a superior balance between the closed-loop control performance and event-trigger frequency.**

***Index Terms*—Autonomous vehicles, double Q-learning (DDQN), event-triggered model predictive control (eMPC), proximal policy optimization (PPO), reinforcement learning (RL), soft actor-critic (SAC).**

## I. Introduction

AUTONOMOUS vehicles have attracted researchers' attention dramatically in recent years due to the advanced technology in automation, high-speed communication network and new energy [1], [2]. Path planning and path following are two major tasks for the behaviour control of autonomous vehicles [3], [4], [5], [6], [7]. Path planning is executed to plan the path considering safety constraints, and a controller is then used to follow this path accurately by considering the current states and providing suitable control. Path planning has been well explored by many researchers [8], [9], [10], [11]. However, path following still remains a problem due to the high dynamic, limited computation and communication of autonomous vehicles. The path following controller are expected to provide accurate control inputs in real-time with constrained computation and communication. Path following control can be implemented using different controllers, e.g., proportional-integral-derivative (PID) control, state feedback controllers, model predictive control (MPC), and so on [12], [13], [14], [15], [16], [17], [18].

MPC is capable of handling multi-input multi-output (MIMO) systems with various constraints, making it specially suitable for real-world autonomous vehicle path following problem. Despite the advances of MPC over the years [19], [20], [21], [22], [23], [24], solving the constrained optimal control problem requires high computational power, which is further increased as the system dimension and prediction horizon increase. This has hindered its application to autonomous vehicles' path following that require a short sampling time but have limited computation power.

To reduce computational burden, event-triggered MPC (eMPC) has emerged as a promising paradigm where MPC algorithm is solved – instead of at each time instant as in the traditional MPC implementation – only when triggered by a predefined trigger condition [25], [26], [27], [28], [29], [30], [31], [32], [33], [34]. In such framework, a triggering event can be defined based on either the deviation of the system states [25], [26], [27] or the cost function value [28], [29]. By solving the optimization problem only when necessary, eMPC can significantly reduce online computations. However, the trigger mechanism design, concerning when to trigger the optimization so as to preserve system performance while keeping the number of triggers low, still remains a challenge [35].

The most common event-trigger policy is the threshold-based event-trigger policy, where an event is triggered if the predicted state trajectory and real-time feedback diverge beyond a certain threshold [25], [26], [27]. Besides the threshold based on policy, other researchers also investigate the triggering policy considering specific requirement of the system [36], [37], [38], [39], [40]. In [36], event-triggered real-time scheduling of stabilizing

Manuscript received 9 October 2023; accepted 31 October 2023. Date of publication 3 November 2023; date of current version 23 February 2024. This work was supported in part by the National Science Foundation under Grants 2045436 and 2237317. *(Corresponding authors: Jun Chen; Zhaojian Li.)*

Fengying Dang is with the University of Michigan Transportation Research Institute, Ann Arbor, MI 48109 USA (e-mail: fdang2@gmu.edu).

Dong Chen is with the Department of Electrical and Computer Engineering, Michigan State University, East Lansing, MI 48824 USA (e-mail: chendon9@msu.edu).

Jun Chen is with the Department of Electrical and Computer Engineering, Oakland University, Rochester, MI 48309 USA (e-mail: junchen@oakland.edu).

Zhaojian Li is with the Department of Mechanical Engineering, Michigan State University, East Lansing, MI 48824 USA (e-mail: lizhaoj1@egr.msu.edu).

The associated code is open-sourced and available at: https://github.com/DangFengying/RL-based-event-triggered-MPC.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TIV.2023.3329785.

Digital Object Identifier 10.1109/TIV.2023.3329785

control is proposed. This approach aims to reduce the computational load and communication bandwidth requirements of the control system while maintaining stability and performance. In [37], researchers investigate event-triggered based-network fault detection problems for nonlinear networked control system. However, the performance index design of all these algorithms is usually based on the knowledge of the closed-loop system behavior which is not always available, especially for complex systems. To address this limitation, researchers begin to explore other methods to trigger the event with different application background. In [38], a recurrent neural network is used to build the building dynamics and a cost estimation is used for event trigger. Machine-learning-based event-triggered model predictive control (ETMPC) system is developed to optimize both building energy efficiency and thermal comfort. The system is evaluated through simulations by applying it to air-conditioning control for performance assessment. In [41], a dynamic threshold parameters is used to determine when to trigger the information exchange and control update. It aims to improve cooperative cruise control for multiple high-speed trains with random switching topologies. However, triggering condition design is still a hard problem, especially when the system or environment is very complex which often affects the performance of the algorithm.

In recent years, the deep learning technique has been applied to finding the better event-triggered strategy. Our prior work [42] investigates the use of model free RL techniques, a simple linear Q-learning approach, to synthesize a triggering policy with the aim of achieving the optimal balance between control performance and computational efficiency. However, this linear Q-learning has a hard time capturing the nonlinear event-trigger policy, leading to unnecessarily high event frequency. Therefore, in this paper, we propose to use deep RL to learn the event-trigger policy which makes the proposed framework achieve better trade-offs between system performance and computation cost. At the same time, a new multiple-input model is used in this paper to make simulation environment can be applied to more scenarios.

Note that the use of deep reinforcement learning in event-triggered control has been reported in the literature. See for example [43]. The problem considered in this paper is substantially different from that of [43]. Specifically, in [43], when an event is not triggered, the zero-order-hold is applied to control input, i.e., the control input is invariant in between two events, while in the proposed RL-based event-triggered MPC framework, the control input can be varying in between two events, making is a harder learning problem for RL. This paper addresses the autonomous driving path following problem using a novel control framework. First, it extends the previous work [42] with an improved vehicle model, thereby removing the limitation of using only the front steering angle as driving control. Second, we develop a *model-free* deep-RL-eMPC framework that uses deep RL to learn the event-trigger condition in the path tracking problem of autonomous driving, so that no prior knowledge of the closed-loop system is needed, which is essential for a dynamic and complex system. Both off-policy and on-policy RL methods are tested. Meanwhile, techniques including prioritized experience replay (PER) buffer and long-short term memory (LSTM) are exploited to significantly improve the training efficiency and control performance. Third, comparative validation of various DRL methods for event-triggered control in path following is conducted using a nonlinear autonomous vehicle model. Simulation results show that our approach clearly outperforms the previous linear Q-learning based approach in [42].

The remainder of the paper is organized as follows. Section II formulates the autonomous vehicle path following problem. Section III presents the framework of eMPC with triggering policy obtained from RL. The experiment setup and results of the proposed deep-RL-eMPC method in the autonomous vehicle path following problem are presented in Section IV. Finally, conclusion remarks are provided in Section V.

## II. Problem Formulation

This paper aims to improve autonomous vehicles path following control by proposing a systematic, algorithmic framework where eMPC can be used without having the prior knowledge of the closed-loop system behavior. Our goal is to use an RL agent to learn the optimal event-trigger policy automatically.

### A. Task Description: Autonomous Vehicle Dynamics and Path Following Problem

In order to demonstrate the proposed deep RL-eMPC and its improving techniques, a path following task is chosen. For a single track vehicle model, the equations for vehicle center of gravity (CG) and wheel dynamics are given by

$$\dot{l}_x = v_x \cos\psi - v_y \sin\psi, \tag{1a}$$

$$\dot{v}_x = v_y r + \frac{2}{m}\sum_{i=f,r} F_{x,i} - g\sin\sigma_g - \frac{1}{m}F_a, \tag{1b}$$

$$\dot{l}_y = v_x \sin\psi + v_y \cos\psi, \tag{1c}$$

$$\dot{v}_y = -v_x r + \frac{2}{m}\sum_{i=f,r} F_{y,i}, \tag{1d}$$

$$\dot{\psi} = r, \tag{1e}$$

$$\dot{r} = \frac{1}{I}\left(2L_{x,f}F_{y,f} - 2L_{x,r}F_{y,r}\right), \tag{1f}$$

where $l_x$ and $l_y$ are the longitudinal and lateral position of the center of gravity of vehicle, respectively; $\psi$ is the vehicle rotational angle along the longitudinal axis in the *global* inertial frame; and $v_x$, $v_y$, and $r$ are, respectively, the vehicle longitudinal velocity, lateral velocity, and yaw rate in the *vehicle* frame. $F_a$ is the aerodynamic drag force [44] and $F_x$ and $F_y$ are tire forces. $m$ is the vehicle mass, $I$ is the vehicle rotational inertia on yaw dimension, $L_{xf}$ and $L_{xr}$ are the distance from CG to the middle of front and rear axle, respectively.

The tire force $F_{x,i}$ and $F_{y,i}$ in (1b), (1d) in vehicle frame can be modeled by

$$F_{x,i} = \bar{F}_{x,i}\cos\beta_i - \bar{F}_{y,i}\sin\beta_i \tag{2a}$$

$$F_{y,i} = \bar{F}_{x,i}\sin\beta_i + \bar{F}_{y,i}\cos\beta_i, \tag{2b}$$

where $\beta_i$ is the wheel-road-angle for the wheel $i$, $i = \{f, r\}$ represents the front or rear wheel, $\bar{F}_{x,i}$ and $\bar{F}_{y,i}$ are the tire force in wheel frame which can be obtained as

$$\bar{F}_{x,i} = \frac{T_i}{2R}, \tag{3a}$$

$$\bar{F}_{y,i} = C_i \mu_i F_{z,i} \alpha_i, \tag{3b}$$

where $T_i$ is the propulsion/braking torque along the axle, $R$ is the effective tire radius, $C_i$ is the tire corner stiffness and $\mu_i$ characterize the road surface, $\alpha_i$ is the slip angle. We refer readers to [27] for a detailed computation of the slip angle $\alpha_i$.

The normal force $F_{z,i}$ in (1f) can be modeled by static load transfer,

$$F_{z,i} = \frac{L_{x,i} mg}{2(L_{x,f} + L_{x,r})}. \tag{4}$$

In this paper, we consider a problem of autonomous vehicle following a sinusoidal trajectory using the proposed deep-RL-eMPC method [27], [45], the following path is given by

$$l_y = g(l_x) = 4 \sin\left(\frac{2\pi}{100} l_x\right). \tag{5}$$

### B. Optimal Control Problem and its Goal

Consider a discrete-time system with the following dynamics

$$x_{t+1} = f(x_t, u_t), \tag{6}$$

where $x_t \in \mathbb{R}^n$ is the system state at discrete time $t$ and $u_t \in \mathbb{R}^m$ is the control input. Given a prediction horizon $p$, MPC aims to find the optimal control sequence $U_t$ and optimal state sequence $X_t$ by solving the following optimal control problem:

$$\min_{X_t, U_t} \quad J_{\text{mpc}} = \sum_{k=0}^{p} \ell(x_{t+k}, u_{t+k}) \tag{7a}$$

$$\text{s.t.} \quad x_t = \hat{x}_t \tag{7b}$$

$$x_{t+k} = f(x_{t+k-1}, u_{t+k-1}), \quad 1 \le k \le p \tag{7c}$$

$$x_{\min} \le x_{t+k} \le x_{\max}, \quad 1 \le k \le p \tag{7d}$$

$$u_{\min} \le u_{t+k} \le u_{\max}, \quad 0 \le k \le p-1 \tag{7e}$$

$$\Delta_{\min} \le u_{t+k} - u_{t+k-1} \le \Delta_{\max}, \quad 0 \le k \le p-1, \tag{7f}$$

Where $U_t$ and $X_t$ are defined as $U_t = \{u_t, u_{t+1}, \ldots, u_{t+p-1}\}$ and $X_t = \{x_{t+1}, x_{t+2}, \ldots, x_{t+p}\}$, $\ell(x_{t+k}, u_{t+k})$ is the stage cost function, $\hat{x}_t$ denotes the real state or current state estimation, and $u_{t+k}$ denotes the control action at time step $t+k$. For conventional time-triggered MPC, the above optimal control problem is solved for every sampling time $t$, and only the first element $u_t$ of $U_t$ is applied to the system as the control command, while all the remaining elements $u_{t+1}, \ldots, u_{t+p-1}$ are abandoned.

Let $t$ and $t_p$ represent the current time step and the last event time, respectively, and there thus exists a $k \in \mathbb{N}$ such that $t = t_p + kdt$ where $dt$ is the sampling time of the discrete system. Let $a_t$ denotes the triggering command in event-triggered

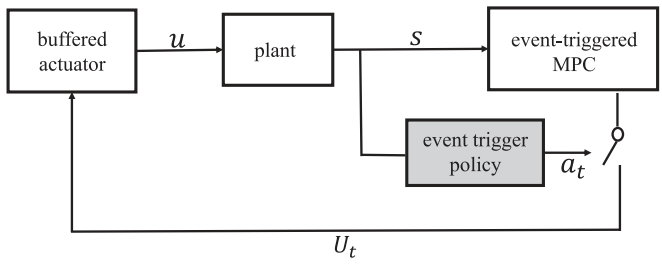



Fig. 1. Scheme of event-triggered model predictive control (eMPC).

MPC at time step $t$. Then when $a_t = 1$, the above optimal control problem is solved and the first element of the optimal control sequence $U_t$ computed at current time step $t$ will be used as control command. When $a_t = 0$, the optimal control sequence $U_{t_p}$ computed at last event when the time instance equals to $t_p$ will be shifted to determine the control command [27]. Then the control input $u$ can be compactly represented as:

$$u_t = \begin{cases} U_t(1), & \text{if} \quad a_t = 1, \\ U_{t_p}(k+1), & \text{if} \quad a_t = 0. \end{cases} \tag{8}$$

To implement (8) for eMPC, a buffer can be used to store the optimal control sequence $U_{t_p}$ computed at last event at time $t_p$. At each time step, the event-trigger policy block generates $a_t$ based on current feedback from the plant. In eMPC, only when $a_t = 1$, a new control sequence $U_t$ is computed by solving (7), whose first element is implemented by actuator as $u$, while the entire sequence is saved into buffer. If $a_t = 0$, indicating the absence of an event, the control sequence currently stored in the buffer will be shifted based on the time elapsed since last event to determine the current control input $u$. This process is depicted in Fig. 1.

In general, the event $a_t$ can be generated by certain event-trigger policy $\pi$, denoted as,

$$a_t \sim \pi_\theta(X_{t_p}, \hat{x}_t), \tag{9}$$

where $X_{t_p}$ is the optimal state sequence computed at last event when $a_{t_p} = 1$ and $\hat{x}_t$ is the real state (or current state estimate if not directly measured), $\theta$ are parameters characterizing the policy. It is worth noting that, for nonlinear constrained MPC, the design of event-trigger policy $\pi$ is challenging and requires extensive calibration and prior knowledge of the closed-loop system behavior. Therefore, the design of event-trigger policy and its calibrations are usually problem specific and non-trivial. To address this limitation, the objective of this paper is to learn the optimal event-trigger policy $\pi$ using model-free deep RL techniques.

If we discretize (1) to obtain a discrete-time model in the form of (6), with $x = [l_x, v_x, l_y, v_y, \psi, r]$ and $u = [T_f, \beta_f]$ where $T_f$ is the axle driving torque and $\beta_f$ is the front steering angle. The stage cost of (7a) is defined as

$$\ell(x, u) = \left|\left| x(3) - 4 \sin\left(\frac{2\pi}{100} x(1)\right) \right|\right|_{Q_t}^2 + ||u - u^r||_{Q_u}^2, \tag{10}$$

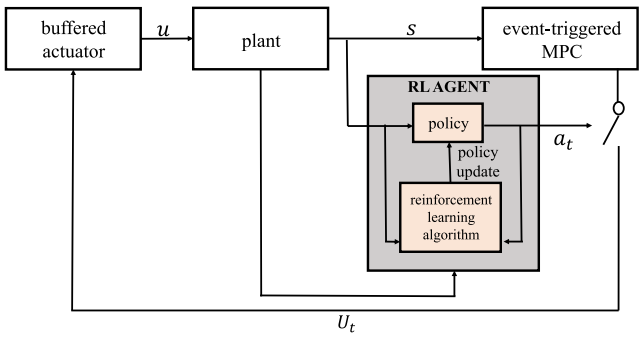



Fig. 2. Scheme of RL based event-triggered MPC.

where the first nonlinear term penalizes the path tracking error and the second term penalizes large control efforts. Here the norm is defined as $||x||_Q^2 = x^T Q x$. More specifically, the MPC cost function $J_{\text{mpc}}$ in (7a) in this case can be equivalently represented as:

$$J_{\text{mpc}}(X_t, U_t) = \sum_{k=1}^{p} \left|\left| x_{t+k}(3) - 4\sin\left(\frac{2\pi}{100} x_{t+k}(1)\right)\right|\right|_{Q_t}^2 + \sum_{k=0}^{p-1} \left(||u_{t+k} - u_{t+k}^r||_{Q_u}^2\right), \quad (11)$$

where $U_t$ and $X_t$ are defined as $U_t = \{u_t, u_{t+1}, \ldots, u_{t+p-1}\}$ and $X_t = \{x_{t+1}, x_{t+2}, \ldots, x_{t+p}\}$, and the terms independent of $X_t$ and $U_t$ are ignored.

## III. Event-Triggered MPC With Deep RL-Based Policy Learning

In this section, we present our proposed deep RL-based policy learning eMPC, or deep-RL-eMPC.

### A. Deep-RL-eMPC Framework

The process of our deep-RL-eMPC framework is shown in Fig. 2. The RL agent learns the event-trigger policy parameter $\theta$ by continuously interacting with the environment. Specifically, at each time step, the agent sends an action $a$ to the environment. The environment then implements the eMPC following (8), simulates the dynamic system following (6), and emits an immediate reward following the designed reward function. The agent then observes the reward signals, update $\theta$, and transitions to next state.

For an eMPC problem, the discrete action space for RL agent is defined as $\mathcal{A} = \{0, 1\}$, where the event will be triggered when $a = 1$ and will not be triggered when $a = 0$. As the feedback from the environment, the immediate reward function is defined as

$$r_t \triangleq -\ell(\hat{x}_t, u_t)dt - \rho_c a_t, \quad (12)$$

where the first term $\ell(\hat{x}_t, u_t)dt$ measures the closed-loop system performance and the second term $\rho_c a_t$ measures the cost of triggering events. Note that $\ell(\hat{x}_t, u_t)$ is the stage cost and is computed using the the real state (or current state estimate if not directly measured) $\hat{x}_t$ and real-time control (8). Furthermore, $\rho_c$ is a hyper-parameter used to balance between control performance index and triggering frequency. One can fine tune this hyperparameter $\rho_c$ to make a tradeoff between control performance and computational cost.

The complete deep-RL-eMPC algorithm is shown in Algorithm 1. In this algorithm, $M$ is the total number of training epochs, $T$ is the length of each episode representing total training time in each epoch, $\gamma$ is the discount factor in the reward function, $dt$ is the discrete time step, and $N$ is the size of sampled experiences at each time (batch size). The output of Algorithm 1 is the system parameters $\theta$. The RL agent interacts with the environment for $M$ number of epochs (Lines 2–24). After initialization, Lines 5 shows how to choose action. Lines 7–12 implement the event-triggered MPC to compute the control command $u$, which is used to simulate the dynamical system (6) (Line 13). After that, the environment emits next state $s_{t+1}$ and immediate reward $r_t$ Line 16), which is observed by RL agent (Line 18). The latest experience tuple $(s_t, a_t, r_t, s_{t+1})$ is then added into an experience buffer $\mathcal{D}$ (Line 19). The RL parameters $\theta$ is updated using a batch of $N$ experiences sampled from the experience buffer $\mathcal{D}$ (Line 20). RL agent then moves to next state Line 21). After each epoch, RL agent is reset for the next epoch (Line 3). Lines 7-16 are part of the environment, whose computation is unknown to the RL agents. Note that the agent only observes the environment outputs, i.e., next state and reward.

### B. Deep RL Algorithms and Improving Technique

The framework shown in Fig. 2 and Algorithm 1 is a general frame which can accommodate different RL algorithm. In this paper, we investigate three different RL agents, including Double Q-learning (DDQN) [46] and Proximal Policy Optimization (PPO) [47], Soft Actor-Critic (SAC) [48], and show the proposed framework is also suitable for other RL algorithms.

In this subsection, we first briefly describe these three deep RL algorithms. Then two improving technique for Rl agent including PER and LSTM are presented.

*1) Double Q-Learning:* Deep Q network is a type of Q-learning which uses neural network as a policy. To address the issues of overestimation of Q values in deep Q network [49], Double Q-learning (DDQN) explicitly separates action selection from action evaluation which allows each step to use a different function approximator and shows a better overall approximation of the action-value function [46]. DDQN improves deep Q network by replacing the target $y^{DQN}$ by $y^{DDQN} = r_t + \gamma Q_{\theta'}(s_{t+1}, \arg\max_a Q_\theta(s_{t+1}, a))$, resulting in the Double Q-learning loss:

$$L_{DDQN}(\theta) = E_{\mathcal{D}}[y^{DDQN} - Q_\theta(s_t, a_t)]^2. \quad (13)$$

*2) PPO:* PPO, an on-policy policy gradient RL algorithm, replaces the KL-divergence used in TRPO [50] with a clipped surrogate objective function (14), which is proved to be better suited for the TRPO and easy to implement.

$$L_{PPO}^{CLIP}(\theta) = E_t[\min(r_t A_t, clip(r_t, 1-\varepsilon, 1+\varepsilon)A_t)]. \quad (14)$$

**Algorithm 1:** RL-based Event-Triggered MPC.

**Input:** $M$, $T$, $dt$, $\gamma$, $N$
**Output:** $\theta$
1 Initialize $\theta$, $\mathcal{D} \leftarrow \emptyset$;
2 **for** $j = 0$ *to* $M - 1$ **do**
3  Initialize $s_t$, $Z$, $U$, $k \leftarrow 0$;
4  **while** $t <= T$ **do**
5  select action $a_t \sim \pi_\theta(X_{t_p}, \hat{x}_t)$
6  *% Simulate Environment*;
7  **if** $a_t = 1$ **then**
8  $k \leftarrow 0$;
9  $(Z, U) \leftarrow$ Solving optimal control problem (7);
10  **else**
11  $k \leftarrow k + 1$;
12  **end**
13  $u \leftarrow U(k)$;
14  $x_{t+1} \leftarrow$ Simulate system dynamics (6) using $u$;
15  $s_{t+1} \leftarrow (x_{t+1}, Z(k))$;
16  $r_t \leftarrow (12)$;
17  *% End of Environment Simulation*;
18  Observe $r_t$ and $s_{t+1}$;
19  Update $\mathcal{D}$ to include $(s_t, a_t, r_t, s_{t+1})$;
20  Sample $N$ experiences from $\mathcal{D}$ and update $\theta$;
21  $s_t \leftarrow s_{t+1}$;
22  $t \leftarrow t + dt$
23  **end**
24 **end**
25 Note: $\mathcal{D}$ can be either conventional on-policy or off-policy experience buffer or priority experience buffer.

*3) Soft Actor-Critic:* SAC achieves the state-of-the-art performance across a wide range of continuous-action control problems and updates the stochastic actor-critic policy in an off-policy way. SAC takes a good exploration-exploitation trade-off via entropy regularization.

In this paper, we adopt SAC and PPO to the discrete action space setting following the discrete categorical distribution design in [51]. For details, refer to DDQN [46], PPO [47] and SAC [48].

The training performance of the proposed deep-RL-eMPC framework depends on the quality of the selected experience sample, so how to choose them is critical when using off-policy RL algorithms. The experience replay buffer utilizes a fixed-size buffer that holds the most recent transitions collected by the policy [52], [53]. In RL, the weights updating and optimization of neural networks are based on the experience replay. The experience replay in the original DDQN uniformly samples the stored experience to train the network weights. However, the importance of experiences are different. Some experiences are more valuable than others in the long run and important experience should be considered more frequently. To address this problem, the prioritized experience replay has been proposed [54] to prioritize more frequent replay transitions leading to high expected learning progress, as measured by the magnitude of their TD error. Specifically, the probability of sampling transition $i$ is defined as follows:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}, \tag{15}$$

where $\alpha \in [0, 1]$ controls how much prioritization is applied; when $\alpha = 0$, the experience will be sampled uniformly. Here $p_i > 0$ represents the priority of transition $i$, which is initialized as 1 and updated based on the TD-error $\delta_i$ during the transition.

More specifically, to alleviate the bias of the gradient magnitudes introduced by the priority replay, importance-sampling (IS) is introduced in [54] as:

$$w_i = \left(\frac{1}{\mathcal{N}} \frac{1}{P(i)}\right)^\beta. \tag{16}$$

where $\beta$ is the hyperparameter annealing the amount of importance-sampling correction over time. $\mathcal{N}$ is size of the experience buffer. The weight $w_i$ is then used in the Q-learning updates by replacing the TD-error $\delta_i$ as $w_i \delta_i$. In practice, we can apply the PER by replacing line 24 in Algorithm 1 with the designed PER scheme.

To encode the historical information in the network, a straightforward way is to feed all historical states to the RL agent, but it increases the state dimension significantly and may distract the attention of the RL agent from recent input states. To address this challenge, recurrent neural network (RNN) has been developed, which is a class of artificial neural networks that can encode and learn temporal information. Traditional RNN does not have the ability for long term memory and suffers from vanishing gradient problem. Long short-term memory (LSTM) [55], a type of RNN architecture, solves this issue by using feedback connections and thus suitable for long-time series data. In this paper, we will explore the use of LSTM as the last hidden layer to extract representations from different state types and encode the history information.

## IV. Autonomous Vehicle Path Following Using Deep-RL-eMPC

In this section we apply the proposed deep-RL-eMPC to a nonlinear autonomous vehicle path tracking problem. The prediction horizon of MPC is set to $p = 5$ with upper and lower bounds for all control inputs. Since autonomous vehicle requires short control sampling time but has limited onboard computation power, this nonlinear path tracking problem is a good example to demonstrate the proposed deep-RL-eMPC.

### A. RL Structure and Settings

In this paper, we encode the input state with a one fully connected (FC) layer with 128 neurons, followed by two 128-neuron FC layers. In the LSTM design, we replace the last FC layer with a 128-unit LSTM layer. The last layer outputs two Q values corresponding to two actions, i.e., trigger and not trigger. The target network in DDQN are updated every $N_0 = 1000$ steps.

The state of the environment is defined to be $s = (\hat{x}, \bar{x})$, where $\hat{x}$ as mentioned above is the state estimate of the dynamical system and $\bar{x}$ is the MPC prediction made at last event. The

TABLE I
EVALUATION RETURN R, TRIGGERING FREQUENCY $A_F$, AND MPC COST $E_{mpc}$ USING DIFFERENT RL AGENTS IN DEEP-RL-EMPC

| | | Threshold | LSTDQ | SAC | DDQN | DDQN+LSTM+PER | PPO | PPO+LSTM |
|---|---|---|---|---|---|---|---|---|
| $\rho_c = 0$ | return | 1.606 | 0.062 | 0.058 | 0.056 | 0.058 | **0.055** | **0.055** |
| | $A_f/E_{mpc}$ | 0.118/1.606 | 0.902/0.062 | 0.99/0.058 | 0.902/0.056 | 0.99/0.058 | 0.99/0.058 | 0.99/0.055 |
| $\rho_c = 0.001$ | return | 1.618 | 0.157 | 0.158 | 0.152 | 0.137 | 0.119 | **0.112** |
| | $A_f/E_{mpc}$ | 0.118/1.606 | 0.931/0.157 | 0.98/0.058 | 0.951/0.055 | 0.794/0.056 | 0.594/0.059 | 0.594/0.059 |
| $\rho_c = 0.01$ | return | 1.728 | 0.66 | 1.015 | 0.627 | **0.431** | 0.634 | 0.529 |
| | $A_f/E_{mpc}$ | 0.118/1.606 | 0.559/0.660 | 0.922/0.075 | 0.5/0.117 | 0.255/0.171 | 0.515/0.114 | 0.515/0.069 |

reward function follows (12), with $\ell(\hat{x}_t, u_t)$ defined as follows:

$$\ell(\hat{x}_t, u_t) = \left|\left|\hat{x}_t(3) - 4\sin\left(\frac{2\pi}{100}\hat{x}_t(1)\right)\right|\right|^2_{Q_t} + ||u_t - u_t^r||^2_{Q_u}, \tag{17}$$

where $\hat{x}_t$ is the real state (or current state estimate if not directly measured) and $u_t$ is the real-time applied control computed by (8). Then the return for one episode in the RL algorithm is as follows:

$$R = \sum_{t=1}^{T_e} r_t = \sum_{t=1}^{T_e} \left(-\ell(\hat{x}_t, u_t)dt - \rho_c a_t\right), \tag{18}$$

where $R$ is the episodic return of RL algorithms, $T_e$ is the number of steps for the episode, $\rho_c$ is a hyper- parameter proposed to balance control performance and event trigger frequency. To evaluate performance of different RL algorithms in our deep-RL-eMPC frame, we adopt the following two evaluation metrics: total MPC cost $E_{mpc}$ and event triggering frequency $A_f$, which are defined as follows:

$$E_{mpc} = \sum_{t=1}^{T_e} \left(\ell(\hat{x}_t, u_t)dt\right) \tag{19}$$

$$A_f = \frac{\sum_{t=1}^{T_e} a_t}{T_e}, \tag{20}$$

We train the off-policy RL algorithms over 50,000 steps, which is around 500 episodes, each with a length of $T = 20$ s and a sampling time of $dt = 200$ ms, i.e., episode horizon is $T_e = 100$ time steps. On-policy algorithms, e.g., PPO, often require longer training time but with improved stability [51], thus we train them for 1000 episodes for better convergence. For MDP, we set the discount factor $\gamma = 0.99$ and batch size $N = 64$. The learning rate and replay buffer size are set as $\eta = 1e-4$ and 5,000, respectively. Also, $\epsilon$-greedy is adopted in DDQN with $\epsilon$ linearly decaying from 1.0 to 0.01 during the first 5000 steps of training.

### B. Simulation Results and Analysis

Numerical simulation results on the evaluation returns for $\rho_c = 0$, 0.001, 0.01 with the threshold-based benchmark and different variants of RL algorithms are summarized in Table I. The simple linear Q-learning method (least-square temporal difference Q-learning, LSTDQ) [42] is also shown here as a benchmark. To measure the computation burden required by different RL algorithms and MPC, we run the simulation 10000 times and use the average time as the time cost. The results show that the average time cost of MPC is about 0.1 s while the average time cost of RL algorithms considered in this paper is about $10^{-6}$ s. In other words, each MPC computation requires $10^5$ times more computation than evaluating RL policies, and hence the time cost spent on the decision making of RL algorithms is negligible. So overall speaking, fewer MPC queries will provide less computation burden.

The threshold-based event-trigger policy [27] depends on a manually-tuned threshold to determine when the event is triggered. However, this method is very sensitive to the tracking error and is susceptible to over-triggering problems when the error is large. This causes the return of the threshold-based method around 1.6 for all three different $\rho_c$, much worse than the RL-based methods as shown in Table I.

Comparing LSTDQ, SAC, DDQN, and PPO, experimental results clearly show that deep-RL-eMPC frameworks achieve better evaluation return than the the conventional threshold-based approach and previous LSTDQ for all three different $\rho_c$. It is also shown that PPO presents the best result under $\rho_c = 0$ and $\rho_c = 0.001$, while DDQN performs better when $\rho_c = 0.01$ in terms of evaluation return, partly due to the low overestimation. To show the flexibility of the proposed framework, PER buffer and LSTM are employed to foster the exploration and efficiency of the training of DDQN and PPO. PPO is an on policy RL method and PER cannot be applied to this method, so only PPO+LSTM is tested. Specifically, DDQN+LSTM+PER and PPO+LSTM are implemented and compared. The experimental results show that LSTM and PER significantly increase the evaluation return of the system, outperforming the baseline methods. SAC performs well when $\rho_c = 0$, while it fails in the more challenging cases when $\rho_c = 0.001$ or 0.01. The intrinsic reason for the poor performance of SAC deserves to be investigated in the future work.

Recall that the hyperparameter $\rho_c$ can be used to balance control performance and triggering frequency. When $\rho_c = 0$, RL triggers MPC at nearly every time step and achieves the smallest tracking error. As the value of $\rho_c$ increases, the rewards function (12) penalizes more on triggering MPC, resulting in less frequent events and higher MPC costs $J_{\text{mpc}}$. The bigger the $\rho_c$ is, the larger penalty the system will give for triggering the events. From Table I, we can see when $\rho_c$ is larger, the system

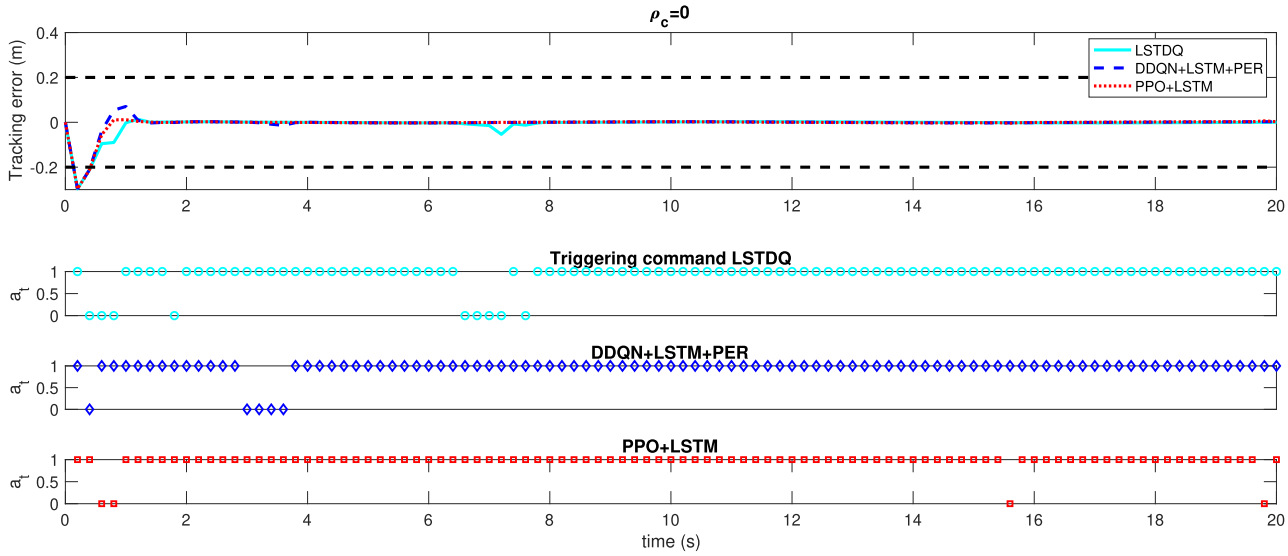



Fig. 3. Simulation results of deep-RL-eMPC for the reward function with $\rho_c = 0$. The comparison of the tracking error using three different RL algorithm in deep-RL-eMPC (first row). The corresponding triggering commands $a_t$ during the process when using LSTDQ (second row), DDQN+LSTM+PER (third row) and PPO+LSTM (fourth row).

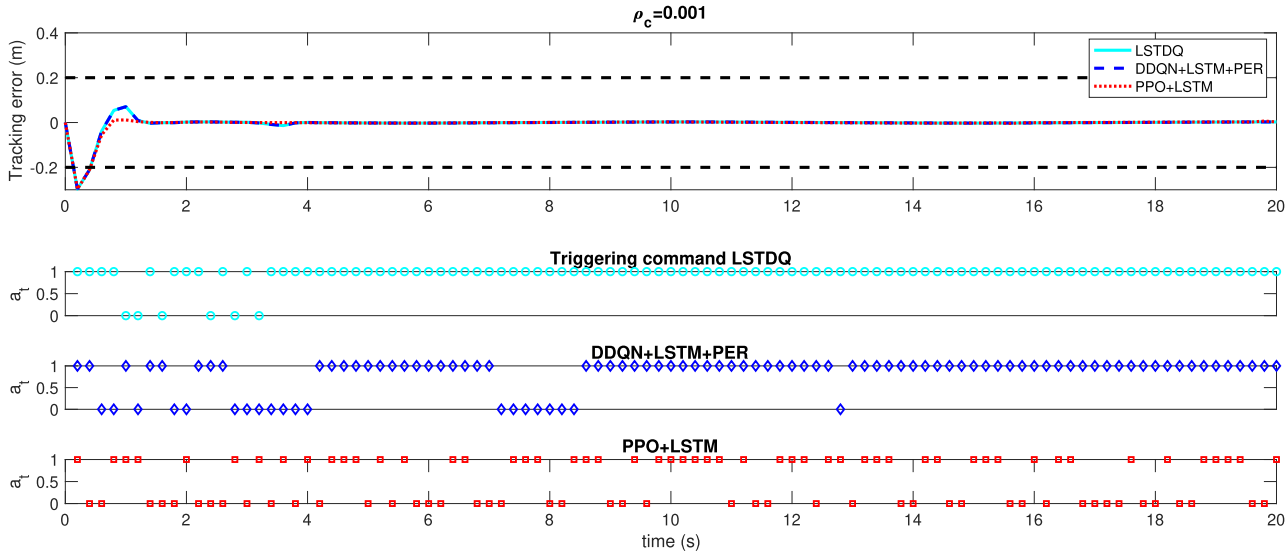



Fig. 4. Simulation results of deep-RL-eMPC for the reward function with $\rho_c = 0.001$. The comparison of the tracking error using three different RL algorithm in deep-RL-eMPC (first row). The corresponding triggering commands $a_t$ during the process when using LSTDQ (second row), DDQN+LSTM+PER (third row) and PPO+LSTM (fourth row).

tends to give smaller returns because of the larger punishment of triggering the events.

Fig. 3, Fig. 4 and Fig. 5 shows the path following error and event triggering command $a_t$ when using three different RL algorithm (LSTDQ, DDQN+LSTM+PER, PPO+LSTM) in the deep-RL-MPC framework when using different $\rho_c$ in reward equation (18). The first row shows the comparison of the tracking error using three different RL algorithm in deep-RL-eMPC. The corresponding triggering commands $a_t$ during the process is showed in second row when using LSTDQ, is showed in third row when using DDQN+LSTM+PER and is showed in fourth row when using PPO+LSTM. The best results from deep-RL-eMPC when $\rho_c = 0$ and $\rho_c = 0.001$ are from PPO+LSTM and when $\rho_c = 0.01$ is from DDQN+LSTM+PER. In LSTDQ, when $\rho_c = 0$, $E_{mpc} = 0.062$ and the triggering frequency is 0.902. When $\rho_c = 0.001$, $E_{mpc} = 0.157$ and the triggering frequency is 0.931. When $\rho_c = 0.01$, $E_{mpc} = 0.66$ and the triggering frequency is 0.559. In PPO+LSTM, when $\rho_c = 0$, $E_{mpc} = 0.055$ and the triggering frequency is 0.99. In this situation, there is no penalty on triggering MPC, and the RL agent triggers MPC for nearly every sampling time, and the path tracking error is the smallest. It results in a triggering frequency of 5 Hz as the

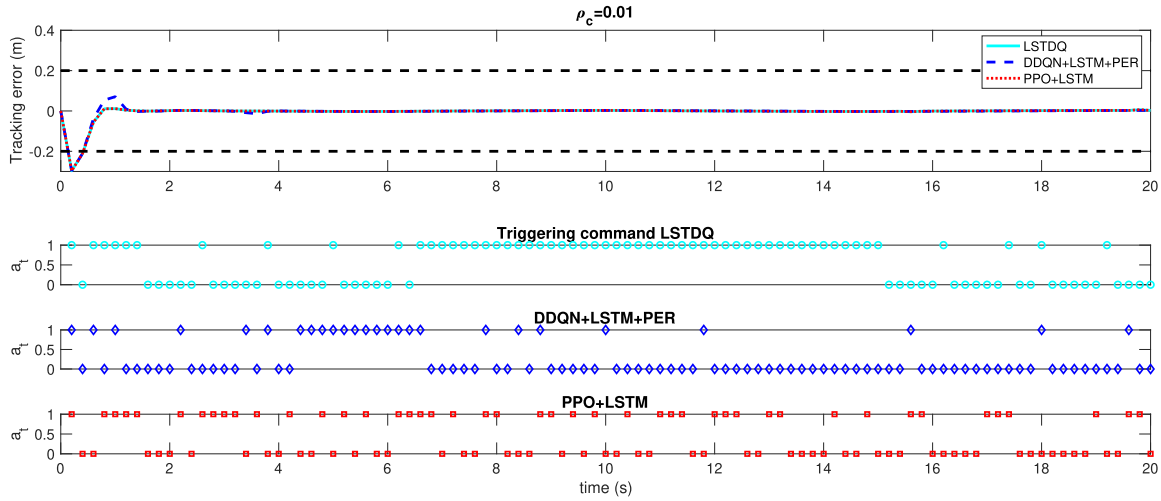



Fig. 5. Simulation results of deep-RL-eMPC for the reward function with $\rho_c = 0.01$. The comparison of the tracking error using three different RL algorithm in deep-RL-eMPC (first row). The corresponding triggering commands $a_t$ during the process when using LSTDQ (second row), DDQN+LSTM+PER (third row) and PPO+LSTM (fourth row).

sampling time is $dt = 0.2$ s. When $\rho_c = 0.001$, $E_{mpc} = 0.059$ and the triggering frequency is 0.594. In this situation, the RL agent tends to trigger an event when the tracking error is large, and keeps silent when the error is going to be around 0. When $\rho_c = 0.01$, DDQN+LSTM+PER achieves the best performance with $E_{mpc} = 0.171$ and the triggering frequency is 0.255. In this situation, the event-trigger pattern is similar to that of $\rho_c = 0.001$, but with a lower triggering frequency. It is worth noting that, for each case, DDQN+LSTM+PER triggers MPC less frequently (resulting in less MPC computation) while incurring smaller MPC cost (resulting in better control performance). We can then conclude that DDQN+LSTM+PER and PPO+LSTM outperforms the previous LSTDQ method as presented in [42].

### C. Additional Remarks

It is worth noting that the simulation environment used in this paper is based on a nonlinear autonomous vehicle model with simultaneous control of both longitudinal and lateral dynamics. While such a simulation environment is complex enough for proof-of-concept demonstration, future improvement can be made by using more sophisticated tool, such as CARLA [56] or SUMO [57]. Moreover, future improvement can be made by incorporating random noise to differentiate the training and validation environments.

## V. Conclusion

This paper investigats a path following problem for autonomous driving. We present a novel eMPC framework with the triggering policy obtained from deep reinforcement learning to solve the problem. A reward function is proposed to balance control performance and event trigger frequency through a hyper-parameter $\rho_c$. In comparison to existing eMPC, the proposed algorithm does not require any knowledge of the closed-loop dynamics (i.e., model-free) and delivers superior performance. We also demonstrate that incorporating techniques such as priority experience replay and long-short term memory can significantly enhance the performance. The learnt deep RL-based triggering policy effectively reduces the computational burden while achieving satisfactory control performance. In future work, we will explore the time-varying computational budgets and costs within this deep-RL-eMPC framework for autonomous driving path following. Additionally, we will investigate a more complex simulation environment and some professional simulation software like CARLA. Furthermore, we will examine the stability and convergence of the proposed deep-RL-eMPC framework in hardware experiments.

## References

[1] F.-Y. Wang et al., "Verification and validation of intelligent vehicles: Objectives and efforts from China," *IEEE Trans. Intell. Veh.*, vol. 7, no. 2, pp. 164–169, Jun. 2022.
[2] J. Lu, L. Han, Q. Wei, X. Wang, X. Dai, and F.-Y. Wang, "Event-triggered deep reinforcement learning using parallel control: A case study in autonomous driving," *IEEE Trans. Intell. Veh.*, vol. 8, no. 4, pp. 2821–2831, Apr. 2023.
[3] B. B. K. Ayawli, R. Chellali, A. Y. Appiah, and F. Kyeremeh, "An overview of nature-inspired, conventional, and hybrid methods of autonomous vehicle path planning," *J. Adv. Transp.*, vol. 2018, 2018, Art. no. 8269698.
[4] A. Muraleedharan, H. Okuda, and T. Suzuki, "Real-time implementation of randomized model predictive control for autonomous driving," *IEEE Trans. Intell. Veh.*, vol. 7, no. 1, pp. 11–20, Mar. 2022.
[5] L. Yang, C. Lu, G. Xiong, Y. Xing, and J. Gong, "A hybrid motion planning framework for autonomous driving in mixed traffic flow," *Green Energy Intell. Transp.*, vol. 1, no. 3, 2022, Art. no. 100022.
[6] S. Feng, Z. Song, Z. Li, Y. Zhang, and L. Li, "Robust platoon control in mixed traffic flow based on tube model predictive control," *IEEE Trans. Intell. Veh.*, vol. 6, no. 4, pp. 711–722, Dec. 2021.
[7] S. Teng et al., "Motion planning for autonomous driving: The State of the Art and future perspectives," *IEEE Trans. Intell. Veh.*, vol. 8, no. 6, pp. 3692–3711, Jun. 2023.
[8] T. Qie, W. Wang, C. Yang, Y. Li, W. Liu, and C. Xiang, "A path planning algorithm for autonomous flying vehicles in cross-country environments with a novel TF-RRT* method," *Green Energy Intell. Transp.*, vol. 1, no. 3, 2022, Art. no. 100026.

[9] F. Poinsignon, L. Chen, S. Jiang, K. Gao, H. Badia, and E. Jenelius, "Autonomous vehicle fleets for public transport: Scenarios and comparisons," *Green Energy Intell. Transp.*, vol. 1, no. 3, 2022, Art. no. 100019.

[10] C. Hu, L. Zhao, and G. Qu, "Event-triggered model predictive adaptive dynamic programming for road intersection path planning of unmanned ground vehicle," *IEEE Trans. Veh. Technol.*, vol. 70, no. 11, pp. 11228–11243, Nov. 2021.

[11] Z. Zhou, J. Chen, M. Tao, P. Zhang, and M. Xu, "Experimental validation of event-triggered model predictive control for autonomous vehicle path tracking," in *Proc. IEEE Int. Conf. Electro Inf. Technol.*, 2023, pp. 35–40.

[12] P. F. Lima, M. Nilsson, M. Trincavelli, J. Mårtensson, and B. Wahlberg, "Spatial model predictive control for smooth and accurate steering of an autonomous truck," *IEEE Trans. Intell. Veh.*, vol. 2, no. 4, pp. 238–250, Dec. 2017.

[13] S. Mata, A. Zubizarreta, and C. Pinto, "Robust tube-based model predictive control for lateral path tracking," *IEEE Trans. Intell. Veh.*, vol. 4, no. 4, pp. 569–577, Dec. 2019.

[14] Y. Lin, J. McPhee, and N. L. Azad, "Comparison of deep reinforcement learning and model predictive control for adaptive cruise control," *IEEE Trans. Intell. Veh.*, vol. 6, no. 2, pp. 221–231, Jun. 2021.

[15] Z. Zuo et al., "MPC-based cooperative control strategy of path planning and trajectory tracking for intelligent vehicles," *IEEE Trans. Intell. Veh.*, vol. 6, no. 3, pp. 513–522, Sep. 2021.

[16] Z. Zhou, F. Zhu, D. Xu, B. Chen, S. Guo, and Y. Dai, "Event-triggered multi-lane fusion control for 2-D vehicle platoon systems with distance constraints," *IEEE Trans. Intell. Veh.*, vol. 8, no. 2, pp. 1498–1511, Feb. 2023.

[17] I. Ahmad, X. Ge, and Q.-L. Han, "Communication-constrained active suspension control for networked in-wheel motor-driven electric vehicles with dynamic dampers," *IEEE Trans. Intell. Veh.*, vol. 7, no. 3, pp. 590–602, Sep. 2022.

[18] S. Wen, G. Guo, B. Chen, and X. Gao, "Cooperative adaptive cruise control of vehicles using a resource-efficient communication mechanism," *IEEE Trans. Intell. Veh.*, vol. 4, no. 1, pp. 127–140, Mar. 2019.

[19] M. Mammarella, T. Alamo, F. Dabbene, and M. Lorenzen, "Computationally efficient stochastic MPC: A probabilistic scaling approach," in *Proc. IEEE Conf. Control Technol. Appl.*, 2020, pp. 25–30.

[20] C. Liu, C. Li, and W. Li, "Computationally efficient MPC for path following of underactuated marine vessels using projection neural network," *Neural Comput. Appl.*, vol. 32, no. 11, pp. 7455–7464, 2020.

[21] Y. Ding, L. Wang, Y. Li, and D. Li, "Model predictive control and its application in agriculture: A review," *Comput. Electron. Agriculture*, vol. 151, pp. 104–117, 2018.

[22] C. Li et al., "A review on the application of the MPC technology in wind power control of wind farms," *J. Energy Power Technol.*, vol. 3, no. 3, pp. 1–22, 2021.

[23] G. Serale, M. Fiorentini, A. Capozzoli, D. Bernardini, and A. Bemporad, "Model predictive control (MPC) for enhancing building and HVAC system energy efficiency: Problem formulation, applications and opportunities," *Energies*, vol. 11, no. 3, 2018, Art. no. 631.

[24] K. Zheng, D. Shi, Y. Shi, and J. Wang, "Non-parametric event-triggered learning with applications to adaptive model predictive control," *IEEE Trans. Autom. Control*, vol. 68, no. 6, pp. 3469–3484, Jun. 2023.

[25] F. D. Brunner, W. Heemels, and F. Allgöwer, "Robust event-triggered MPC with guaranteed asymptotic bound and average sampling rate," *IEEE Trans. Autom. Control*, vol. 62, no. 11, pp. 5694–5709, Nov. 2017.

[26] H. Li and Y. Shi, "Event-triggered robust model predictive control of continuous-time nonlinear systems," *Automatica*, vol. 50, no. 5, pp. 1507–1513, 2014.

[27] J. Chen and Z. Yi, "Comparison of event-triggered model predictive control for autonomous vehicle path tracking," in *Proc. IEEE Conf. Control Technol. Appl.*, 2021, pp. 808–813.

[28] N. He and D. Shi, "Event-based robust sampled-data model predictive control: A non-monotonic Lyapunov function approach," *IEEE Trans. Circuits Syst. I: Reg. Papers*, vol. 62, no. 10, pp. 2555–2564, Oct. 2015.

[29] A. Eqtami, D. V. Dimarogonas, and K. J. Kyriakopoulos, "Novel event-triggered strategies for model predictive controllers," in *Proc. IEEE 50th Conf. Decis. Control Eur. Control Conf.*, 2011, pp. 3392–3397.

[30] Z. Zhou, C. Rother, and J. Chen, "Event-triggered model predictive control for autonomous vehicle path tracking: Validation using CARLA simulator," *IEEE Trans. Intell. Veh.*, vol. 8, no. 6, pp. 3547–3555, Jun. 2023.

[31] R. Wan, S. Li, and Y. Zheng, "Model predictive control of nonlinear system with event-triggered parametric identification," in *Proc. Chin. Automat. Congr.*, 2020, pp. 5773–5777.

[32] E. Kang, H. Qiao, Z. Chen, and J. Gao, "Tracking of uncertain robotic manipulators using event-triggered model predictive control with learning terminal cost," *IEEE Trans. Automat. Sci. Eng.*, vol. 19, no. 4, pp. 2801–2815, Oct. 2022.

[33] P. K. Wong and D. Ao, "A novel event-triggered torque vectoring control for improving lateral stability and communication resource consumption of electric vehicles," *IEEE Trans. Intell. Veh.*, doi: 10.1109/TIV.2023.3284220.

[34] J. Lu, Q. Wei, T. Zhou, Z. Wang, and F.-Y. Wang, "Event-triggered near-optimal control for unknown discrete-time nonlinear systems using parallel control," *IEEE Trans. Cybern.*, vol. 53, no. 3, pp. 1890–1904, Mar. 2023.

[35] L. Sedghi, Z. Ijaz, M. Noor-A-Rahim, K. Witheephanich, and D. Pesch, "Machine learning in event-triggered control: Recent advances and open issues," *IEEE Access*, vol. 10, pp. 74671–74690, 2022.

[36] P. Tabuada, "Event-triggered real-time scheduling of stabilizing control tasks," *IEEE Trans. Autom. control*, vol. 52, no. 9, pp. 1680–1685, Sep. 2007.

[37] X. Su, F. Xia, L. Wu, and C. P. Chen, "Event-triggered fault detector and controller coordinated design of fuzzy systems," *IEEE Trans. Fuzzy Syst.*, vol. 26, no. 4, pp. 2004–2016, Aug. 2018.

[38] S. Yang, W. Chen, and M. P. Wan, "A machine-learning-based event-triggered model predictive control for building energy management," *Building Environ.*, vol. 233, 2023, Art. no. 110101.

[39] J. Yoo, E. Nekouei, and K. H. Johansson, "Event-based observer and MPC with disturbance attenuation using ERM learning," in *Proc. IEEE Eur. Control Conf.*, 2018, pp. 1894–1899.

[40] J. Yoo and K. H. Johansson, "Event-triggered model predictive control with a statistical learning," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 51, no. 4, pp. 2571–2581, Apr. 2021.

[41] H. Zhao, X. Dai, Q. Zhang, and J. Ding, "Robust event-triggered model predictive control for multiple high-speed trains with switching topologies," *IEEE Trans. Veh. Technol.*, vol. 69, no. 5, pp. 4700–4710, May 2020.

[42] J. Chen, X. Meng, and Z. Li, "Reinforcement learning-based event-triggered model predictive control for autonomous vehicle path following," in *Proc. IEEE Amer. Control Conf.*, 2022, pp. 3342–3347.

[43] D. Baumann, J.-J. Zhu, G. Martius, and S. Trimpe, "Deep reinforcement learning for event-triggered control," in *Proc. IEEE Conf. Decis. Control*, 2018, pp. 943–950.

[44] R. Rajamani, *Vehicle Dynamics and Control*. Berlin, Germany: Springer, 2011.

[45] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli, "Kinematic and dynamic vehicle models for autonomous driving control design," in *Proc. IEEE Intell. Veh. Symp.*, 2015, pp. 1094–1099.

[46] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

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

[48] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 1861–1870.

[49] V. Mnih et al., "Playing atari with deep reinforcement learning," 2013, *arXiv:1312.5602*.

[50] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1889–1897.

[51] P. Christodoulou, "Soft actor-critic for discrete action settings," 2019, *arXiv:1910.07207*.

[52] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Mach. Learn.*, vol. 8, no. 3, pp. 293–321, 1992.

[53] W. Fedus et al., "Revisiting fundamentals of experience replay," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 3061–3071.

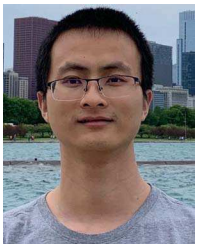
[54] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2015, *arXiv:1511.05952*.

[55] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[56] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proc. 1st Annu. Conf. Robot Learn.*, 2017, pp. 1–16.

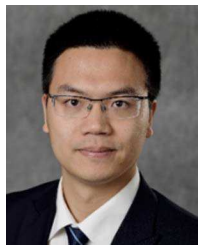
[57] P. A. Lopez et al., "Microscopic traffic simulation using sumo," in *Proc. IEEE 21st Int. Conf. Intell. Transp. Syst.*, 2018, pp. 2575–2582.

**Fengying Dang** received the B.S. degree in detection, gudiance and control from Northwestern Polytechnical University, Xi'an, China, and the Ph.D. degree in electrical and computer engineering from George Mason University, Fairfax, VA, USA. She is currently a Postdoctoral Research Associate with the University of Michigan Transportation Research Institute, Ann Arbor, MI, USA. Her research interests include robotics and control, sensing and perception, data-driven modeling, and learning-based control.



**Dong Chen** (Member, IEEE) received the B.E. degree from the University of Electronic Science and Technology of China, Sichuan, China, in 2017. He is currently working toward the Ph.D. degree with the Department of Electrical and Computer Engineering, Michigan State University, East Lansing, MI, USA. His primary research interests include reinforcement learning and multi-agent systems.

**Jun Chen** (Senior Member, IEEE) received the bachelor's degree in automation from Zhejiang University, Hangzhou, China, in 2009, and the Ph.D. degree in electrical engineering from Iowa State University, Ames IA, USA, in 2014. He is currently an Assistant Professor with the ECE Department, Oakland University, Rochester, MI, USA. His research interests include artificial intelligence and optimal control, with applications in intelligent vehicles and energy systems. Dr. Chen is the recipient of the NSF CAREER Award, Best Paper Award from IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING, Best Paper Award from IEEE INTERNATIONAL CONFERENCE ON ELECTRO INFORMATION TECHNOLOGY, Faculty Recognition Award for Research from Oakland University, Publication Achievement Award from Idaho National Laboratory, Research Excellence Award from Iowa State University, and Outstanding Student Award from Zhejiang University. He is currently a Member of SAE.



**Zhaojian Li** (Senior Member, IEEE) received the M.S. and Ph.D. degrees in aerospace engineering (flight dynamics and control) from the University of Michigan, Ann Arbor, MI, USA, in 2013 and 2015, respectively. He is currently an Assistant Professor with the Department of Mechanical Engineering, Michigan State University, East Lansing, MI, USA. As an undergraduate, he studied with the Nanjing University of Aeronautics and Astronautics, Department of Civil Aviation, Nanjing, China. He was an Algorithm Engineer with General Motors from January 2016 to July 2017. He is the author of more than 20 top journal articles and several patents. His research interests include learning-based control, nonlinear and complex systems, and robotics and automated vehicles. He is currently an Associate Editor for the *Journal of Evolving Systems*, American Control Conference, and ASME Dynamics and Control Conference. He was the recipient of the NSF CAREER Award.