Event-Triggered Model Predictive Control with Deep Reinforcement Learning

Fengying Dang[1], Dong Chen[2], Jun Chen[3] and Zhaojian Li[1*]

***Abstract*—Model predictive control (MPC) is a popular optimal control paradigm that has found great success in a myriad of application domains. However, its high computational complexity has limited its adoption in communication- or computation-constrained systems. To alleviate the computation and/or communication burden, event-triggered model predictive control (eMPC) has been proposed, which, however, generally requires *a priori* knowledge of the closed-loop system behavior along with the communication characteristics for designing the event-trigger policy. This knowledge is many times difficult or not economical to obtain, roadblocking the wider use of eMPC. In this paper, an efficient eMPC framework is developed, in which a model-free reinforcement learning (RL) agent is used to learn the optimal event-trigger policy without the need of complete dynamical system and communication knowledge. Furthermore, techniques including prioritized experience replay (PER) buffer and long-short term memory (LSTM) are employed to foster exploration and improve training efficiency. To demonstrate its effectiveness, the proposed framework is applied to an autonomous vehicle path following problem with three deep RL algorithms, i.e., Double Q-learning (DDQN), Proximal Policy Optimization (PPO), and Soft Actor-Critic (SAC). Experimental results show that all three deep RL-based eMPC (deep-RL-eMPC) can achieve better evaluation return than the conventional threshold-based and previous linear Q-based approach. In particular, PPO-eMPC with LSTM and DDQN-eMPC with PER and LSTM obtains a superior balance between the closed-loop control performance and event-trigger frequency. The associated code is open-sourced and available at: https://github.com/DangFengying/RL-based-event-triggered-MPC.**

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

I. INTRODUCTION

OVER the past few decades, model predictive control (MPC) has been employed successfully in a wide variety of applications, including autonomous vehicles, chemical processes, and power systems. MPC can be dated back to the 1980s when engineers in the process industry first began to deploy it in real-world practice [1]. Since then, the increasing computing power of microprocessors has greatly improved its application scope [2]. MPC uses a system model to predict its future behavior, and selects the best control action by solving an optimization problem [3]–[7]. It is capable of handling multi-input multi-output (MIMO) systems with various constraints, making it suitable for many real-world engineering systems.

In a typical MPC implementation, it repeatedly solves an optimization problem online in order to calculate control inputs that minimize some predefined performance measure evaluated over a prediction horizon. The optimal control sequence is therefore calculated at each sampling instant but only the first element is sent to command the actuators [8]. Then the entire process is repeated at next sampling instant with updated measurements. Despite the advances of MPC over the years [9]–[13], 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 a large class of systems that require short sampling time but have limited computation power.

To reduce computational burden without significantly degrading control performance, 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 [14]–[19]. In such framework, a triggering event can be defined based on either the deviation of the system states [14]–[16] or the cost function value [17], [18]. 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 [6].

The most common event-trigger policy is threshold-based event-trigger policy, where an event is triggered if the predicted state trajectory and real-time feedback diverge beyond a certain threshold [14]–[16]. However, the threshold calibration 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, our prior work [20] 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

This work is supported by National Science Foundation grant #2045436.

[1] Fengying Dang and Zhaojian Li are with the Department of Mechanical Engineering at Michigan State University, East Lansing, MI 48824, USA. Email: *dangfen1@msu.edu, lizhaoj1@egr.msu.edu*

[2] Dong Chen is with the Department of Electrical and Computer Engineering at Michigan State University, East Lansing, MI 48824, USA. Email: *chendon9@msu.edu*

[3] Jun Chen is with the Department of Electrical and Computer Engineering at Oakland University, Rochester, MI 48309, USA. Email: *junchen@oakland.edu*

∗Zhaojian Li is the corresponding author.

applicable to a larger application scope with better trade-offs between system performance and computation cost.

The main contributions of our paper include the following. First, we develop a *model-free* deep-RL-eMPC framework that uses deep RL to learn the event-trigger policy online, 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. Second, techniques including prioritized experience replay (PER) buffer and long-short term memory (LSTM) are exploited to significantly improve the training efficiency and control performance. Lastly, the proposed approach is applied to the autonomous vehicle path following problem, where we show that our approach clearly outperforms the conventional threshold-based approach in [16] and the previous linear Q-learning based approach in [20].

The remainder of the paper is organized as follows. Section II reviews relevant literature on RL and MPC integration, to provide more context for the presented work. Section III formulates the event trigger MPC problem. Section IV 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 V. Finally, conclusion remarks are provided in Section VI.

II. Relevant Work on RL/MPC Integration

Utilizing RL to aid MPC is not new in literature. For example, [21], [22] propose an off-policy actor-critic algorithm called DMPC, where an off-policy critic learns a value function while the actor utilizes MPC to interact with the environment. It is assumed that the system dynamic is known, but the cost function that MPC should minimize is unknown and is learnt by the critic's value function estimation. Both analytical and numerical results demonstrate improvements on learning convergence.

RL can also be used to learn the system dynamics that are then used by MPC for prediction [23]–[31]. This approach is called model-based RL in [23]–[31]. Specifically, [23]–[27] studies learning based probabilistic MPC in the framework of RL, where the system dynamics and environment uncertainties are modeled as Gaussian Process (GP), whose parameters are iteratively identified through trial and error. Authors of [29]–[31] use a GP model to learn errors between measurement and a nominal model, which are then used to set up optimal control problem for MPC to guarantee constraints robustness.

Reference [28] combines RL and MPC in the context of surgical robot control. The system dynamics are modeled by artificial neural network (ANN), whose parameters are identified through RL or learning from demonstration. In RL approach, the agent explored the action space using ϵ-greedy, collected observations, and iteratively trained the ANN to model system dynamic, while an MPC is used to optimize action based on the trained ANN. In the learning from demonstration approach, the ANN parameters are initialized using observations collected from human operators.

Finally, RL can also be used to directly optimize MPC control law. For example, [32] proposes a robust MPC where the control law is restricted to an affine function of the feedback with the gain being pre-computed offline and the offset being learnt by RL. Reference [32] additionally shows that the robust MPC can also reject disturbance when the Gaussian process model is unknown and learnt online. Authors in [33] investigated the use of gradient based Partially Observable Markov Decision Processes (GPOMDP) algorithm to train the RL recomputation policy for event-triggered MPC control to save energy [34]. However, the solutions of GPOMDP algorithm often suffer from the high variance of the gradient estimate [35].

To the best of our knowledge, the use of deep RL to trigger MPC has not been reported in literature. In this paper, we attempt to fill this gap by investigating deep RL-based event-triggered MPC, or deep-RL-eMPC, which learns the optimal event-trigger policy without requiring any knowledge on the closed-loop dynamics and therefore significantly reduces the amount of calibrations.

III. Problem Formulation

This paper aims to develop a systematic, algorithmic framework so that 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.

Consider a discrete-time system with the following dynamics

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

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:

$$\begin{aligned}
\min_{X_t, U_t} \quad & J_{\text{mpc}} = \sum_{k=0}^{p} \ell(x_{t+k}, u_{t+k}) && \text{(2a)}\\
\text{s.t.} \quad & x_t = \hat{x}_t && \text{(2b)}\\
& x_{t+k} = f(x_{t+k-1}, u_{t+k-1}), \quad 1 \le k \le p && \text{(2c)}\\
& x_{min} \le x_{t+k} \le x_{max}, \quad 1 \le k \le p && \text{(2d)}\\
& u_{min} \le u_{t+k} \le u_{max}, \quad 0 \le k \le p-1 && \text{(2e)}\\
& \Delta_{min} \le u_{t+k} - u_{t+k-1} \le \Delta_{max}, \\
& \qquad\qquad 0 \le k \le p-1, && \text{(2f)}
\end{aligned}$$

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 MPC

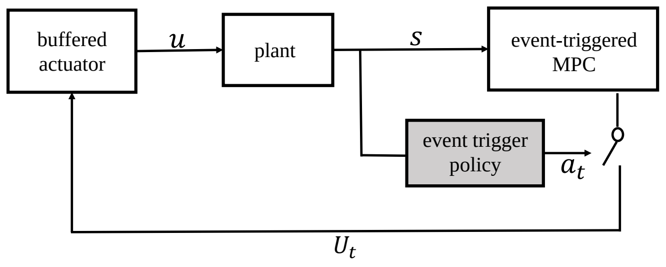

Fig. 1: The scheme of event-triggered model predictive control (eMPC).

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 [16]. Then the control input u can be compactly represented as:

$$u_t = \begin{cases} U_t(1), & \text{if} \quad a_t = 1, \\ U_{t_p}(k+1), & \text{if} \quad a_t = 0, \end{cases} \tag{3}$$

To implement (3) 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 (2), 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 π, denoted as,

$$a_t \sim \pi_\theta(X_{t_p}, \hat{x}_t), \tag{4}$$

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), θ are parameters characterizing the policy. It is worth noting that, for nonlinear constrained MPC, the design of event-trigger policy π 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 π using model-free deep RL techniques.

IV. 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 θ by continuously interacting with the environment.

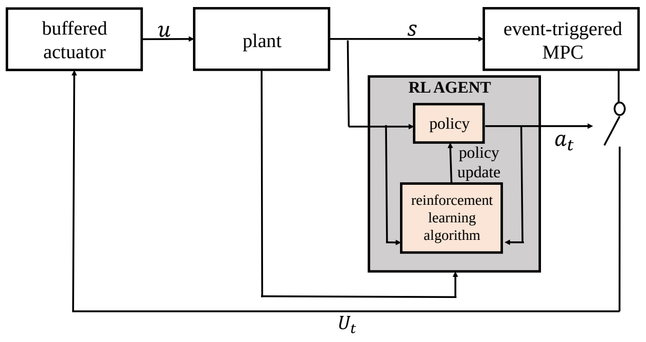

Fig. 2: The scheme of RL based event-triggered MPC.

Specifically, at each time step, the agent sends an action a to the environment. The environment then implements the eMPC following (3), simulates the dynamic system following (1), and emits an immediate reward following the designed reward function. The agent then observes the reward signals, update θ, and transitions to next state.

To 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

$$r_t \triangleq -\ell(\hat{x}_t, u_t)dt - \rho_c a_t, \tag{5}$$

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 (3). Furthermore, ρ_c is a hyper-parameter used to balance between control performance index and triggering frequency. One can fine tune this hyperparameter ρ_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, γ 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 θ. 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 (1) (Line 13). After that, the environment emits next state s_{t+1} and immediate reward r_t (Lines 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 θ is updated using a batch of N experiences sampled from the experience buffer $\mathcal{D}$ (Line 20). RL agent then moves to next state (Lines 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

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
                 (2);
10         else
11             $k \leftarrow k + 1$;
12         end
13         $u \leftarrow U(k)$;
14         $x_{t+1} \leftarrow$ Simulate system dynamics (1) using $u$;
15         $s_{t+1} \leftarrow (x_{t+1}, Z(k))$;
16         $r_t \leftarrow$ (5);
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.
```

agent only observes the environment outputs, i.e., next state and reward.

B. Deep RL Algorithms

In addition to the framework shown in Fig. 2 and Algorithm 1, we investigate three different RL agents, including Double Q-learning (DDQN) [36] and Proximal Policy Optimization (PPO) [37], Soft Actor-Critic (SAC) [38], and show the proposed framework is also suitable for other RL algorithms. In this subsection, we briefly describe these three deep RL algorithms.

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 [39], 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 [36]. 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:

$$L_{DDQN}(\theta) = E_{\mathcal{D}}[y^{DDQN} - Q_\theta(s_t, a_t)]^2. \tag{6}$$

2) PPO: PPO, an on-policy policy gradient RL algorithm, replaces the KL-divergence used in TRPO [40] with a clipped surrogate objective function (7), which is proved to be better suited for the TRPO and easy to implement.

$$L_{PPO}^{CLIP}(\theta) = E_t[\min(r_t A_t, clip(r_t, 1-\varepsilon, 1+\varepsilon)A_t)]. \tag{7}$$

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 [41]. For details, refer to DDQN [36], PPO [37] and SAC [38].

C. The Prioritized Experience Replay (PER)

A critical component of off-policy RL algorithms is experience replay buffer [42], [43]. The experience replay utilizes a fixed-size buffer that holds the most recent transitions collected by the policy. 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 [44] 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:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}, \tag{8}$$

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 δ_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 [44] as:

$$w_i = (\frac{1}{\mathcal{N}}\frac{1}{P(i)})^\beta. \tag{9}$$

where β 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 δ_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.

D. Long Short-term Memory (LSTM)

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) [45], 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.

V. 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. Autonomous Vehicle Dynamics and Path Following Problem

For a single track vehicle model, the equations for vehicle center of gravity (CG) and wheel dynamics are given by

$$\dot{l}_x = v_x \cos\psi - v_y \sin\psi, \tag{10a}$$
$$\dot{v}_x = v_y r + \frac{2}{m}\sum_{i=f,r} F_{x,i} - g\sin\sigma_g - \frac{1}{m}F_a, \tag{10b}$$
$$\dot{l}_y = v_x \sin\psi + v_y \cos\psi, \tag{10c}$$
$$\dot{v}_y = -v_x r + \frac{2}{m}\sum_{i=f,r} F_{y,i}, \tag{10d}$$
$$\dot{\psi} = r, \tag{10e}$$
$$\dot{r} = \frac{1}{I}\left(2L_{x,f}F_{y,f} - 2L_{x,r}F_{y,r}\right), \tag{10f}$$

where l_x and l_y are the longitudinal and lateral position of the center of gravity of vehicle, respectively; ψ 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 [46] 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 (10b), (10d) in vehicle frame can be modeled by

$$F_{x,i} = \bar{F}_{x,i}\cos\beta_i - \bar{F}_{y,i}\sin\beta_i \tag{11a}$$
$$F_{y,i} = \bar{F}_{x,i}\sin\beta_i + \bar{F}_{y,i}\cos\beta_i, \tag{11b}$$

where β_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

$$\bar{F}_{x,i} = \frac{T_i}{2R}, \tag{12a}$$
$$\bar{F}_{y,i} = C_i\mu_i F_{z,i}\alpha_i, \tag{12b}$$

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 μ_i characterize the road surface, α_i is the slip angle. We refer readers to [16] for a detailed computation of the slip angle α_i.

The normal force $F_{z,i}$ in (10f) can be modeled by static load transfer,

$$F_{z,i} = \frac{L_{x,i}mg}{2(L_{x,f} + L_{x,r})}. \tag{13}$$

One can then discretize (10) to obtain a discrete-time model in the form of (1), 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 β_f is the front steering angle. In this paper, we consider a problem of autonomous vehicle following a sinusoidal trajectory using the proposed deep-RL-eMPC method [16], [47], as given by

$$l_y = g(l_x) = 4\sin\left(\frac{2\pi}{100}l_x\right). \tag{14}$$

Therefore, the stage cost of (2a) is defined as

$$\ell(x,u) = \left|\left|x(3) - 4\sin\left(\frac{2\pi}{100}x(1)\right)\right|\right|^2_{Q_t} + ||u - u^r||^2_{Q_u}, \tag{15}$$

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

$$J_{\text{mpc}}(X_t, U_t) = \sum_{k=1}^{p}\left|\left|x_{t+k}(3) - 4\sin\left(\frac{2\pi}{100}x_{t+k}(1)\right)\right|\right|^2_{Q_t} + \sum_{k=0}^{p-1}\left(||u_{t+k} - u^r_{t+k}||^2_{Q_u}\right), \tag{16}$$

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

B. 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 reward function follows (5), with $\ell(\hat{x}_t, u_t)$ defined as follows:

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

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 (3). 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. 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 $T = 20s$ 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 [41], 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, ϵ-greedy is adopted in DDQN with ϵ linearly decaying from 1.0 to 0.01 during the first 5000 steps of training.

C. 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 Tab. I. The simple linear Q-learning method (least-square temporal difference Q-learning, LSTDQ) [20] 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 0.1s 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 [16] 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 ρ_c, much worse than the RL-based methods as shown in Tab. 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 ρ_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 ρ_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 ρ_c increases, the rewards function (5) penalizes more on triggering MPC, resulting in less frequent events and higher MPC costs J_{mpc}. The bigger the ρ_c is, the larger penalty the system will give for triggering the events. From Tab. I, we can see when ρ_c is larger, the system tends to give smaller returns because of the larger punishment of triggering the events.

Fig. 3 shows the MPC cost and event triggering frequency when using LSTDQ and the best results from deep-RL-eMPC. Due to the space limit, the complete experimental results can be accessed at https://github.com/DangFengying/RL-based-event-triggered-MPC/blob/main/deepRLeMPCsup.pdf. 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 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

TABLE I: Evaluation return R, triggering frequency A_f, and MPC cost E_{mpc} using different RL agents in deep-RL-eMPC.

		Threhold	LSTDQ	SAC	DDQN	DDQN+LSTM+PER	PPO	PPO+LSTM
$\rho_\mathrm{c} = 0$	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
$\rho_\mathrm{c} = 0.001$	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
$\rho_\mathrm{c} = 0.01$	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

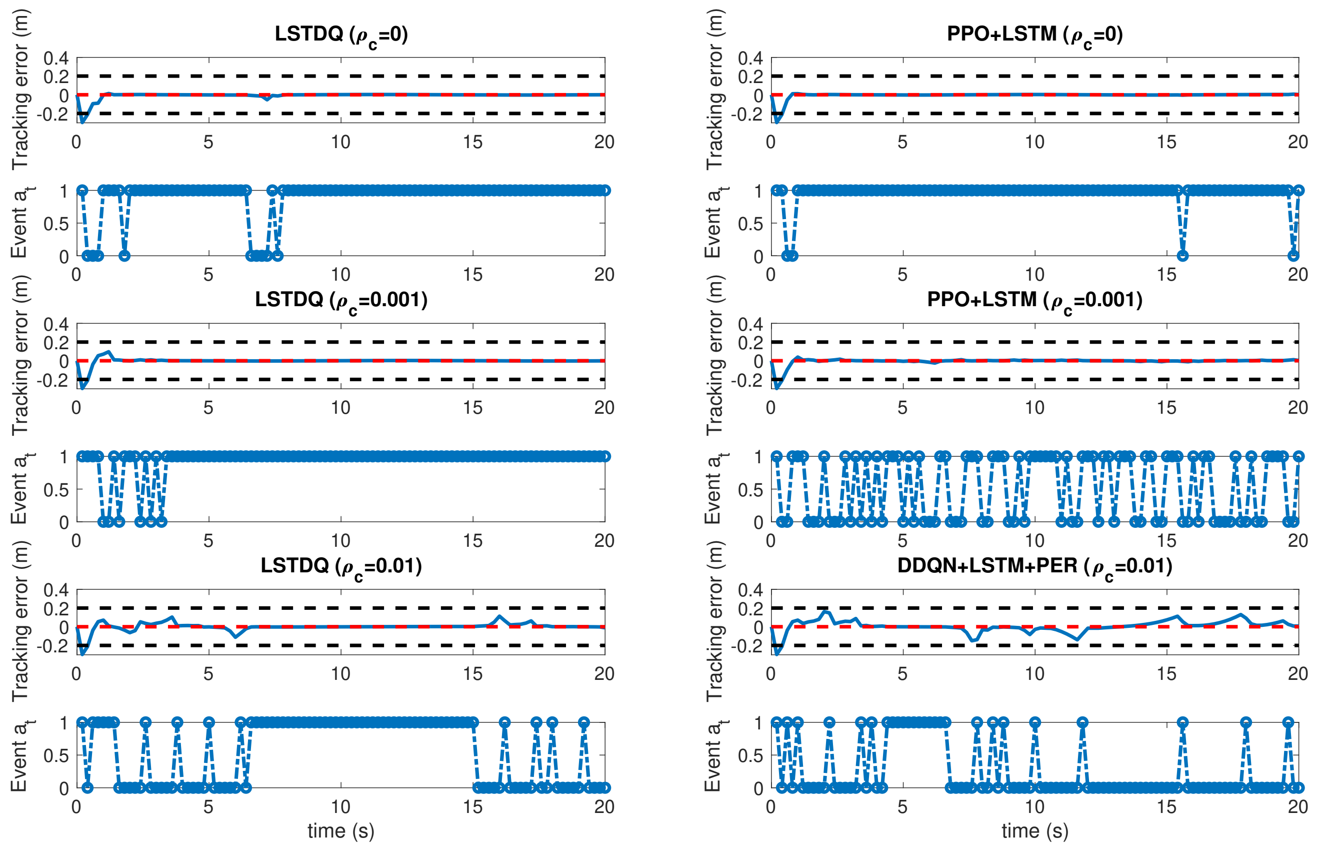

Fig. 3: Experimental results of deep-RL-eMPC for three different reward functions including $\rho_c = 0$ (first and second rows), $\rho_c = 0.001$ (third and fourth rows), $\rho_c = 0.01$ (fifth and sixth rows) for (5).

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 [20].

VI. Conclusion

This paper investigated an event-triggered model predictive control framework with the triggering policy obtained from deep reinforcement learning. A reward function was proposed to balance control performance and event trigger frequency through a hyper-parameter ρ_c. Compared to existing eMPC, the proposed algorithm does not require any knowledge of the closed-loop dynamics (i.e., model-free) and offers better performance. The proposed framework was verified on an autonomous vehicle path following problem and we show 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 decreases the computational burden while achieving satisfactory control performance.

In future work, we will examine the stability and convergence of the proposed framework. Additionally, we will also consider time-varying computational budget and cost, as well as other applications of relevance and impact.

References

[1] J. L. Garriga and M. Soroush, “Model predictive control tuning methods: A review,” *Industrial & Engineering Chemistry Research*, vol. 49, no. 8, pp. 3505–3515, 2010.

[2] G. O'Regan, *Introduction to the history of computing: a computing history primer*. Springer, 2016.

[3] D. Baumann, J.-J. Zhu, G. Martius, and S. Trimpe, "Deep reinforcement learning for event-triggered control," in *2018 IEEE Conference on Decision and Control (CDC)*. IEEE, 2018, pp. 943–950.

[4] D. Baurnann, F. Solowjow, K. H. Johansson, and S. Trimpe, "Event-triggered pulse control with model learning (if necessary)," in *2019 American Control Conference (ACC)*. IEEE, 2019, pp. 792–797.

[5] A. H. Hosseinloo and M. A. Dahleh, "Event-triggered reinforcement learning; an application to buildings' micro-climate control." in *AAAI Spring Symposium: MLPS*, 2020.

[6] L. Sedghi, Z. Ijaz, K. Witheephanich, D. Pesch *et al.*, "Machine learning in event-triggered control: Recent advances and open issues," *arXiv preprint arXiv:2009.12783*, 2020.

[7] J. Yoo and K. H. Johansson, "Event-triggered model predictive control with a statistical learning," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 51, no. 4, pp. 2571–2581, 2021.

[8] M. Morari and J. H. Lee, "Model predictive control: past, present and future," *Computers & Chemical Engineering*, vol. 23, no. 4-5, pp. 667–682, 1999.

[9] M. Mammarella, T. Alamo, F. Dabbene, and M. Lorenzen, "Computationally efficient stochastic mpc: A probabilistic scaling approach," in *2020 IEEE Conference on Control Technology and Applications (CCTA)*. IEEE, 2020, pp. 25–30.

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

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

[12] C. Li, J. Hu, J. Yu, J. Xue, R. Yang, Y. Fu, and B. Sun, "A review on the application of the mpc technology in wind power control of wind farms," *Journal of Energy and Power Technology*, vol. 3, no. 3, pp. 1–1, 2021.

[13] 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, p. 631, 2018.

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

[15] 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.

[16] J. Chen and Z. Yi, "Comparison of event-triggered model predictive control for autonomous vehicle path tracking," in *2021 IEEE Conference on Control Technology and Applications (CCTA)*, San Diego, CA, August 8–11, 2021.

[17] N. He and D. Shi, "Event-based robust sampled-data model predictive control: A non-monotonic lyapunov function approach," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 62, no. 10, pp. 2555–2564, 2015.

[18] A. Eqtami, D. V. Dimarogonas, and K. J. Kyriakopoulos, "Novel event-triggered strategies for model predictive controllers," in *2011 50th IEEE Conference on Decision and Control and European Control Conference*, Orlando, FL, December 12-15, 2011, pp. 3392–3397.

[19] S. Huang and J. Chen, "Event-triggered model predictive control for autonomous vehicle with rear steering," *SAE Technical Paper*, no. 2022-01-0877, 2022.

[20] J. Chen, X. Meng, and Z. Li, "Reinforcement learning-based event-triggered model predictive control for autonomous vehicle path following," in *2022 American Control Conference*, Atlanta, GA, June 8–10, 2022. [Online]. Available: https://jchen2020.net/pubs/c_acc2022-r.pdf

[21] F. Farshidian, D. Hoeller, and M. Hutter, "Deep value model predictive control," *arXiv preprint arXiv:1910.03358*, 2019.

[22] N. Karnchanachari, M. I. Valls, D. Hoeller, and M. Hutter, "Practical reinforcement learning for mpc: Learning from sparse objectives in under an hour on a real robot," in *Learning for Dynamics and Control*. PMLR, 2020, pp. 211–224.

[23] Y. Cui, S. Osaki, and T. Matsubara, "Reinforcement learning boat autopilot: a sample-efficient and model predictive control based approach," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 2868–2875.

[24] ——, "Autonomous boat driving system using sample-efficient model predictive control-based reinforcement learning approach," *Journal of Field Robotics*, vol. 38, no. 3, pp. 331–354, 2021.

[25] C.-Y. Kuo, Y. Cui, and T. Matsubara, "Sample-and-computation-efficient probabilistic model predictive control with random features," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 307–313.

[26] S. Kamthe and M. Deisenroth, "Data-efficient reinforcement learning with probabilistic model predictive control," in *International conference on artificial intelligence and statistics*. PMLR, 2018, pp. 1701–1710.

[27] C. J. Ostafew, A. P. Schoellig, T. D. Barfoot, and J. Collier, "Learning-based nonlinear model predictive control to improve vision-based mobile robot path tracking," *Journal of Field Robotics*, vol. 33, no. 1, pp. 133–152, 2016.

[28] C. Shin, P. W. Ferguson, S. A. Pedram, J. Ma, E. P. Dutson, and J. Rosen, "Autonomous tissue manipulation via surgical robot using learning based model predictive control," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 3875–3881.

[29] L. Hewing, J. Kabzan, and M. N. Zeilinger, "Cautious model predictive control using gaussian process regression," *IEEE Transactions on Control Systems Technology*, vol. 28, no. 6, pp. 2736–2743, 2019.

[30] J. Kabzan, L. Hewing, A. Liniger, and M. N. Zeilinger, "Learning-based model predictive control for autonomous racing," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3363–3370, 2019.

[31] A. Carron, E. Arcari, M. Wermelinger, L. Hewing, M. Hutter, and M. N. Zeilinger, "Data-driven model predictive control for trajectory tracking with a robotic arm," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3758–3765, 2019.

[32] R. Soloperto, M. A. Müller, S. Trimpe, and F. Allgöwer, "Learning-based robust model predictive control with state-dependent uncertainty," *IFAC-PapersOnLine*, vol. 51, no. 20, pp. 442–447, 2018.

[33] E. Bøhn, S. Gros, S. Moe, and T. A. Johansen, "Optimization of the model predictive control update interval using reinforcement learning," *IFAC-PapersOnLine*, vol. 54, no. 14, pp. 257–262, 2021.

[34] J. Baxter and P. L. Bartlett, "Infinite-horizon policy-gradient estimation," *Journal of Artificial Intelligence Research*, vol. 15, pp. 319–350, 2001.

[35] D. Xu and Q. Liu, "Acis: An improved actor-critic method for pomdps with internal state," in *2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 2015, pp. 369–376.

[36] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

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

[38] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*. PMLR, 2018, pp. 1861–1870.

[39] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[40] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*. PMLR, 2015, pp. 1889–1897.

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

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

[43] W. Fedus, P. Ramachandran, R. Agarwal, Y. Bengio, H. Larochelle, M. Rowland, and W. Dabney, "Revisiting fundamentals of experience replay," in *International Conference on Machine Learning*. PMLR, 2020, pp. 3061–3071.

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

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

[46] R. Rajamani, *Vehicle dynamics and control*. Springer Science & Business Media, 2011.

[47] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli, "Kinematic and dynamic vehicle models for autonomous driving control design," in *2015 IEEE Intelligent Vehicles Symposium*, Seoul, Korea, June 28–July 1, 2015, pp. 1094–1099.