**2022 American Control Conference**
**Atlanta, USA, June 8-10, 2022**

# Reinforcement Learning-based Event-Triggered Model Predictive Control for Autonomous Vehicle Path Following

Jun Chen, *Senior Member, IEEE*, Xiangyu Meng, *Member, IEEE* and Zhaojian Li, *Senior Member, IEEE*

*Abstract*— **Event-triggered model predictive control (MPC) has been proposed in literature to alleviate the high computational requirement of MPC. Compared to conventional time-triggered MPC, event-triggered MPC solves the optimal control problem only when an event is triggered. Several event-trigger policies have been studied in literature, typically requiring *a prior* knowledge of the MPC closed-loop system behavior. This paper addresses such limitation by investigating the use of model-free reinforcement learning (RL) to trigger MPC. Specifically, the optimal event-trigger policy is learnt by an RL agent through interactions with the MPC closed-loop system, whose dynamical behavior is assumed to be unknown to the RL agent. A reward function is defined to balance the closed-loop control performance and event frequency. As an illustrative example, the autonomous vehicle path following problem is used to demonstrate the applicability of using RL to learn and execute trigger policy for event-triggered MPC.**

## I. Introduction

Model predictive control (MPC) can be applied to both linear and nonlinear systems and can handle explicit constraints on both inputs and states. Therefore, MPC has been one of the most popular control methods and has been applied in various systems [1]–[7]. Despite the advantage of dealing with system constraints, MPC does require high computational power, since it solves, at each time step, an optimal control problem (OCP) in the form of a constrained optimization. The computation required to solve the OCP further increases when the system dimension and/or prediction horizon increases. To address this limitation, several techniques have been investigated to reduce the online computation, such as constraints removal [8], explicit MPC [2], and warm-started solver [9].

Another promising direction to reduce the MPC computation without significantly degrading the control performance is event-triggered MPC, in which MPC is triggered to formulate and solve the OCP only when it is needed (as opposed to being time-triggered at a fixed sampling rate). See for example [10]–[16], and the reference therein. By performing optimization only when it's necessary, event-triggered MPC can reduce the online computation significantly. In our prior work, we applied event-triggered MPC to AV path tracking problem, where the MPC is set to track the vehicle path in both longitudinal and lateral directions, with axle driving torque and front steering input as the control variables [16]. Two formulations of event-triggered MPC were investigated in [16]. In the first formulation, the optimal control sequence computed from last event was shifted to determine the control action when MPC was not triggered. In the second formulation, the control action in the absence of an event was determined by a time-triggered linear parametric varying MPC (LPV-MPC) with shorter prediction horizon to compensate prediction error and disturbance. Both formulations employed a threshold-based event-trigger policy with a carefully selected threshold, i.e., an event is triggered if the predicted state trajectory and real-time feedback diverge beyond a certain threshold. Compared to time-triggered MPC, benefits of event-triggered MPC on computation saving were clearly demonstrated in [16].

Though event-triggered MPC has shown success in the aforementioned works, the event-trigger policy is usually designed by utilizing domain knowledge on the MPC closed-loop systems, and the calibration of the event-trigger policy to achieve optimal balance between control behavior and event frequency is non-trivial. In this work, we attempt to address this issue by investigating the use of model-free reinforcement learning (RL) to trigger MPC. Specifically, we propose RL-based event-triggered MPC, or RLeMPC, where an RL agent learns the optimal event-trigger policy by continuously interacting with the environment, i.e., MPC closed-loop system, whose dynamical behavior is assumed to be unknown to the RL agent. At every time step, RL agent sends an action to the environment to either trigger or not trigger an event. The environment triggers MPC to solve a new OCP if an event is triggered by RL agent, or shifts the previous optimal control sequence in the absence of an event. The dynamic system evolves accordingly, and an immediate reward is emitted, which is observed by RL agent to update the policy. In this case, a reward function is defined to balance the closed-loop control behavior and event frequency. In order to save online computation, Q-learning with linear value function approximation is adopted for the RL agent. As an illustrative example, the autonomous vehicle path following problem is used to demonstrate the applicability of using RL to learn and execute trigger policy for event-triggered MPC. Comparison with threshold-based event-trigger mechanism used in [16] is also conducted to show the advantages of the proposed RLeMPC.

The idea of using RL to trigger control is not new. See for example [17]–[20] where RL has been investigated in the

This work is supported in part by SECS Faculty Startup Fund and URC Faculty Research Fellowship at Oakland University.

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

Xiangyu Meng is with the Division of Electrical and Computer Engineering, Louisiana State University, Baton Rouge, LA 70803, USA (email: xmeng5@lsu.edu).

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

context of event-triggered control. Compared to [17]–[20], our work is different as the primary objective of using event-triggered MPC is to save online computation, as opposed to saving communication in [17]–[20]. Furthermore, in [17]–[20], zero order hold is applied to control command when an event is not triggered, while in event-triggered MPC, the previous optimized control sequence is shifted to obtain the latest control command. Therefore, the control command will still be time-varying during the absence of event. Such differences make it harder for the RL agent to learn the optimal event-trigger policy.

The rest of this paper is organized as follows. Section II provides necessary background information on RL, while Section III presents the main algorithm on RL-based event-triggered MPC, or RLeMPC. Numerical simulation results on AV path following is presented in Section IV, and the paper is concluded in Section V.

## II. Preliminaries on RL

This section provides preliminary on RL. More details can be found in [21]–[23].

### A. Reinforcement Learning and Q-Learning

In the RL literature, at time step $t$, $s_t$ denotes the states of the environment, $a_t$ denotes the action that an agent applies to the environment, $r(s_t, a_t)$ is a scalar reward function that maps state-action pair to a scalar value indicating the immediate reward that the agent receives from the environment after applying action $a_t$ at state $s_t$. The goal of the agent is to learn an optimal policy $\pi^* : s \to a$ that maximizes the expected cumulative future rewards

$$G = E_\pi \left[ \sum_{t=0}^{\infty} \gamma^t r_t \right], \tag{1}$$

where $r_t = r(s_t, a_t)$ and the scalar $\gamma \in [0, 1]$ is the discount factor that weights more on short-term rewards compared to delayed rewards. The RL agent learns the optimal policy $\pi^*$ through interactions with the environment, during which the agent applies action $a_t$, observes the system evolution from $s_t$ to $s_{t+1}$, and receives immediate reward $r_t$. In general, the environment can be stochastic, either due to unknown disturbance or the nature of the system itself.

To measure the value of a state $s$, a state value function $V^\pi(s)$ can be defined as the value of state $s$ under the policy $\pi$, which is the expected return starting from $s$ following policy $\pi$. In other words,

$$V^\pi(s) = E_\pi \left[ \sum_{k=0}^{\infty} \gamma^k r_{t+k} \middle| s_t = s \right]. \tag{2}$$

Similarly, the state-action value function $Q^\pi(s, a)$ of a state-action pair $(s, a)$ can be defined as the expected return starting from $s$ following by first taking action $a$ and then the policy $\pi$. In other words,

$$Q^\pi(s, a) = E_\pi \left[ \sum_{k=0}^{\infty} \gamma^k r_{t+k} \middle| s_t = s, a_t = a \right]. \tag{3}$$

Once the optimal Q-function $Q^*(s, a) = \max_\pi Q^\pi(s, a)$ is learnt, one can then apply the greedy control to obtain the optimal policy

$$\pi^*(s) = \arg\max_a Q^*(s, a). \tag{4}$$

Therefore, the goal of learning an optimal policy $\pi^*$ reduces to that of learning the optimal Q-function $Q^*(s, a)$. This is called Q-learning, and is a model-free method that works well on discrete action space. In practice, one usually parameterizes the Q-function with certain parameters $\phi$, denoted as $Q_\phi(s, a)$, and uses it to approximate the Q-function.

### B. Least-Square Temporal Difference Q-learning

Least-square temporal difference Q-learning, or LSTDQ [23], uses linear function to approximate $Q(s, a)$. Given a state-action pair $(s, a)$ and a feature vector defined as

$$X(s, a) = \begin{bmatrix} X_1(s, a) \\ X_2(s, a) \\ \vdots \\ X_n(s, a) \end{bmatrix}, \tag{5}$$

one can then represent the Q-function as a linear combination of features, i.e.,

$$Q_\phi(s, a) = X(s, a)^T \phi. \tag{6}$$

To learn the parameter $\phi$ in (6), an experience replay buffer $\mathcal{D}$ can be constructed to collect previously encountered transitions $(s_t, a_t, r_t, s_{t+1})$. Given a sampled experience $(s_t, a_t, r_t, s_{t+1})$ from $\mathcal{D}$, the TD (temporal difference) update is given by

$$a'_{t+1} = \arg\max_a Q_\phi(s_{t+1}, a) \tag{7}$$
$$\delta = r_t + \gamma Q_\phi(s_{t+1}, a'_{t+1}) - Q_\phi(s_t, a_t) \tag{8}$$
$$\Delta\phi = \alpha \delta X(s_t, a_t), \tag{9}$$

which can be obtained by applying stochastic gradient descent update towards the TD target $r_t + \gamma Q_\phi(s_{t+1}, a'_{t+1})$. This update utilizes only one experience from the replay buffer $\mathcal{D}$. To efficiently use multiple transitions, the LSTDQ algorithm utilizes a batch of $N$ samples from $\mathcal{D}$, and solves for total update to be zero, i.e., $\Delta\phi = 0$, resulting the following update:

$$\phi = \left[ \sum_{t=1}^{N} X(s_t, a_t) \left( X(s_t, a_t) - \gamma X(s_{t+1}, a'_{t+1}) \right)^T + \epsilon I \right]^{-1} \times \sum_{t=1}^{N} r_t X(s_t, a_t), \tag{10}$$

where the $\epsilon I$ was added to ensure the matrix inversion always exists. The detailed derivation of (10) can be found in [23].

## III. RL-BASED EVENT-TRIGGERED MPC

Before we present our main algorithm on RL-based event-triggered MPC, or RLeMPC, we first review the event-triggered MPC problem. Consider the following discrete-time system dynamics

$$\zeta_{t+1} = f(\zeta_t, u_t), \tag{11}$$

where $\zeta_t \in \mathbb{R}^n$ is the system state at discrete time $t$, $u_t \in \mathbb{R}^m$ is the control input. Given a prediction horizon $p$, the MPC aims to find the optimal control sequence $U_t$ and optimal state sequence $Z_t$ by solving an optimal control problem (OCP), defined as:

$$\min_{Z_t, U_t} \quad \sum_{k=1}^{p} J^k_{\text{mpc}}(Z_t, U_t) \tag{12a}$$

$$\text{s.t.} \quad \zeta_t = \hat{\zeta}_t \tag{12b}$$

$$\zeta_{t+k} = f(\zeta_{t+k-1}, u_{t+k-1}), \quad 1 \le k \le p \tag{12c}$$

$$\zeta_{min} \le \zeta_{t+k} \le \zeta_{max}, \quad 1 \le k \le p \tag{12d}$$

$$u_{min} \le u_{t+k} \le u_{max}, \quad 0 \le k \le p-1 \tag{12e}$$

$$\Delta_{min} \le u_{t+k} - u_{t+k-1} \le \Delta_{max}, \quad 0 \le k \le p-1, \tag{12f}$$

where $U_t$ and $Z_t$ are defined as $U_t = \{u_t, u_{t+1}, \ldots, u_{t+p-1}\}$ and $Z_t = \{\zeta_{t+1}, \zeta_{t+2}, \ldots, \zeta_{t+p}\}$. For (12b), $\hat{\zeta}_t$ denotes the current state estimation, and for (12f), $u_{t-1}$ denotes the control action applied at the previous control loop. For conventional time-triggered MPC, the above OCP is solved for every sampling time $t$, and the first element of $U_t$, i.e., $u_t$ is applied to actuators as control command, while the remaining of the optimal sequence $U_t$ is then abandoned. Unlike time-triggered MPC, event-triggered MPC solves the OCP (12) only when an event $\gamma_{\text{ctrl}}$ is triggered, i.e., $\gamma_{\text{ctrl}} = 1$. When $\gamma_{\text{ctrl}} = 0$, the optimal sequence $U_{t_1}$ computed at last event (at time $t_1$) can be used to determine the control command [16], i.e.,

$$u = \begin{cases} \text{Solution of (12)} & \text{if} \quad \gamma_{\text{ctrl}} = 1 \\ U_{t_1}(k+1) & \text{Otherwise.} \end{cases} \tag{13}$$

In general, the event-trigger policy can be defined as,

$$\gamma_{\text{ctrl}} = \pi(Z_{t_1}, \hat{\zeta}_t | \theta), \tag{14}$$

where $Z_{t_1}$ is the optimal state sequence computed at last event and $\hat{\zeta}_t$ is the current state feedback.

It is then trivial to see that it requires careful selection of the event-trigger policy $\pi$ according to the closed-loop system dynamics, as well as extensive calibration of the parameter $\theta$. However, it is usually non-trivial to obtain an analytical form of the MPC closed-loop systems, especially for constrained MPC with nonlinear objective function (12a) and nonlinear prediction model (11). Therefore, the design of event-trigger policy and its calibrations are usually problem specific and non-trivial. To address this limitation, we propose to use an RL agent (more specifically an LSTDQ agent) to learn the optimal event-trigger policy $\pi(Z_{t_1}, \hat{\zeta}_t | \theta)$, without assuming the knowledge of the closed-loop system dynamics. We start by defining the following key elements.

- *Action space*: The action space for RL agent is defined as $\{0, 1\}$ where 1 indicates a trigger event and 0 means no trigger event.
- *State space*: The state space of the environment is defined to be $(\hat{\zeta}, \bar{\zeta})$, where $\hat{\zeta}$ as mentioned above is the state estimate of the dynamical system (11) and $\bar{\zeta}$ is the MPC prediction made at last event.
- *Reward function*: We design the following immediate reward function emitted at time $t$:

$$r_t \triangleq -J^1_{\text{mpc}} dt - \rho a, \tag{15}$$

where the first term measures the closed-loop control performance and the second term encourages less events to reduce online computation.

*Remark 1:* In current work, we consider the state space to be $(\hat{\zeta}, \bar{\zeta})$, which includes the MPC prediction $\bar{\zeta}$ on current state made at last event. An alternative approach is to include the whole optimal state sequence $Z_{t_1}$ in the state space. This will be explored as future work.

*Remark 2:* Note also that the balance between control performance and event frequency is achieved through $\rho$ in the reward function (15), which is deemed as a hyper-parameter in current study. This allows the designer to directly specify the compromise between control performance and MPC computational requirement. In some cases, $\rho$ can be treated as part of the environment, and is known to the RL agent as state. In other words, the compromise between control performance and MPC computational requirement is a dynamic nature of the system. Examples of such scenarios include time-varying availability of microprocessor resources and variable prices of cloud-based computing. In such case, the RL agent needs to learn the optimal event-trigger policy through entire spectrum of $\rho$. This potentially more difficult problem is reserved for future study.

The LSTDQ agent learns parameter $\phi$ by interacting with the environment, in our case, the closed-loop dynamic systems (11) and (13). At each time step, the agent would send an action $a$ to the environment. The environment then sets $\gamma_{\text{ctrl}} = a$, implements the event-triggered MPC (13), simulates the dynamic system (11), and emits an immediate reward (15). The LSTDQ agent then observes the rewards, update $\phi$ according to (10), and transitions to next state. The complete RLeMPC algorithm is shown in Algorithm 1, where the hyper-parameters include the total number of episodes $M$, length $T$ of each episode, discrete time step $dt$, discount factor $\gamma$, and batch size $N$. The output of Algorithm 1 is the weights $\phi$ for the linear value function (6). The RL agent interacts with the environment for $M$ number of epochs (Lines 2-28). In Lines 5-9, $\epsilon$-greedy is used to balance exploration and exploitation. Lines 11-17 implement the event-triggered MPC to compute the control command $u$, which is used to simulate the dynamical system (11) at Line 18. After that, the environment emits next state $s_{t+1}$ and immediate reward $r_t$ at Lines 19-20, which is observed by RL agent at Line 22. The latest experience tuple is then added into an experience buffer $\mathcal{D}$ at Line 23. The RL parameters $\phi$ is updated at Line 24 using a batch of $N$ experiences sampled

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

**Input**: $M$, $T$, $dt$, $\gamma$, $N$
**Output**: $\phi$

```
1:  Initialize $\phi$, $\mathcal{D} \leftarrow \emptyset$
2:  for $j = 0$ to $M - 1$ do
3:      Initialize $s_t$, $Z$, $U$, $k \leftarrow 0$
4:      while $t <= T$ do
5:          if explore then
6:              Sample $a_t$ from $\{0, 1\}$
7:          else
8:              $a_t \leftarrow \arg\max_a X(s_t, a_t)^T \phi$
9:          end if
10:         <Simulate Environment>
11:         if $a_t = 1$ then
12:             $k \leftarrow 0$;
13:             $(Z, U) \leftarrow$ Solving OCP (12)
14:         else
15:             $k \leftarrow k + 1$;
16:         end if
17:         $u \leftarrow U(k)$;
18:         $\zeta_{t+1} \leftarrow$ Simulate (11) using $u$
19:         $s_{t+1} \leftarrow (\zeta_{t+1}, Z(k))$
20:         $r_t \leftarrow$ (15)
21:         <End of Environment Simulation>
22:         Observe $r_t$ and $s_{t+1}$
23:         Update $\mathcal{D}$ to include $(s_t, a_t, r_t, s_{t+1})$
24:         Sample $N$ experiences from $\mathcal{D}$ and $\phi \leftarrow$ (10)
25:         $s_t \leftarrow s_{t+1}$
26:         $t \leftarrow t + dt$
27:     end while
28: end for
```

from the experience buffer $\mathcal{D}$. RL agent then moves to next state at Lines 25-26. After each epoch, RL agent is reset for the next epoch at Line 3.

Note that Lines 11-20 is part of the environment, whose computation is unknown to the RL agents. The LSTDQ agent only observes the environment outputs (next state and reward) at Line 22.

## IV. RLeMPC for AV Path Following

In this section we apply the proposed RL-based event-triggered MPC, or RLeMPC, to nonlinear model predictive control in autonomous vehicle path tracking problem, as considered in [16].

### A. AV Path Following Problem

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

$$\dot{x} = v_x \cos\psi - v_y \sin\psi \tag{16a}$$

$$\dot{v}_x = v_y r + \frac{2}{m} \sum_{i=f,r} F_{x,i} - g \sin\sigma_g - \frac{1}{m} F_a \tag{16b}$$

$$\dot{y} = v_x \sin\psi + v_y \cos\psi \tag{16c}$$

$$\dot{v}_y = -v_x r + \frac{2}{m} \sum_{i=f,r} F_{y,i} \tag{16d}$$

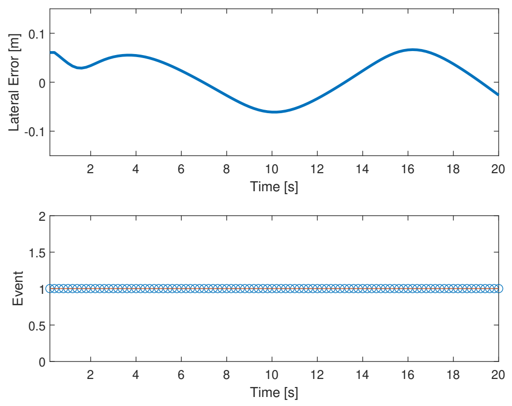

Fig. 1. Results for $\rho = 0$. *Top*: Lateral position error. *Bottom*: Event with moving average.

$$\dot{\psi} = r \tag{16e}$$

$$\dot{r} = \frac{1}{I} \left( 2L_{xf} F_{y,f} - 2L_{xr} F_{y,r} \right), \tag{16f}$$

where $x$, $y$ and $\psi$ are the vehicle CG longitudinal position, lateral position, and rotational angle, and $v_x$, $v_y$, and $r$ are the vehicle longitudinal velocity, lateral velocity, and yaw rate. Note that $x$, $y$, and $\psi$ are in *global* inertial frame while $v_x$, $v_y$, and $r$ are in *vehicle* frame. Furthermore, $m$ is the vehicle mass, $I$ is the vehicle rotational inertia on yaw dimension, and finally, $L_{xf}$ and $L_{xr}$ are the distance from CG to the middle of front and rear axle, respectively. Please refer to [16] for a detailed computation of the aerodynamic drag force $F_a$ and tire forces $F_x$ and $F_y$, where both $F_x$ and $F_y$ are functions of steering angle $u$. One can then discretize (16) to obtain a discrete-time model of the form (11), with $\zeta = [x, v_x, y, v_y, \psi, r]$.

For AV path following, a path planner generates desired path, which is then tracked by the MPC. In this paper, we consider the similar driving maneuver that was investigated in [5], [16], where the vehicle tracks a sinusoidal trajectory. More specifically, the lateral position is a function of the longitudinal position, as given by

$$y = g(x) = 4 \sin\left( \frac{2\pi}{100} x \right). \tag{17}$$

To simplify the simulation, we only consider front steering angle as control input, and the longitudinal control is assumed to be done by a separate controller.

The cost function in this case is defined as follows:

$$J_N(Z_t, U_t) = \sum_{k=1}^{p} \left|\left| \zeta_{t+k}(3) - 4\sin\left( \frac{2\pi}{100} \zeta_{t+k}(1) \right) \right|\right|^2_{Q_t} + \sum_{k=0}^{p-1} \left( ||u_{t+k} - u^r_{t+k}||^2_{Q_u} \right), \tag{18}$$

where the first term penalizes the path tracking error and is nonlinear.

Similar to [16], the prediction horizon is set to $p = 10$. Finally. the upper bounds and lower bounds for input

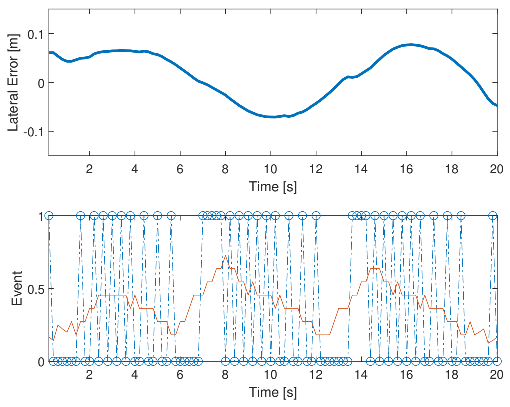

Fig. 2. Results for $\rho = 0.001$. *Top*: Lateral position error. *Bottom*: Event with moving average.

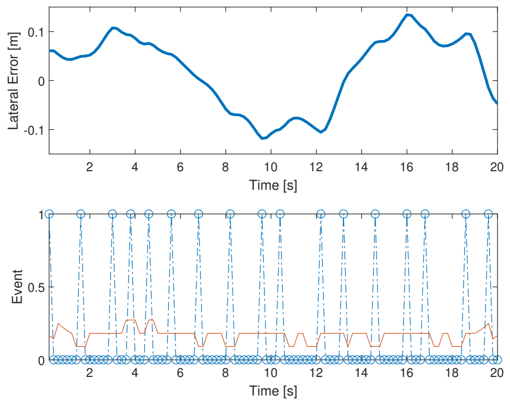

Fig. 3. Results for $\rho = 0.01$. *Top*: Lateral position error. *Bottom*: Event with moving average.

constraints are given by

$$u_{max} = 0.54105 \qquad \Delta_{max} = 0.034907$$
$$u_{min} = -0.54105 \qquad \Delta_{min} = -0.034907.$$

## B. Numerical Results

We train the LSTDQ RL agent for 500 episodes, each with a length of $T = 20$ seconds and a sampling time of $dt = 200$ms. Discount factor is set to $\gamma = 1$, and batch size is $N = 32$.

Numerical results for $\rho = 0, 0.001, 0.01$, are shown in Figs. 1-3, where the lateral position error is shown on top while the event is shown on the bottom. Note the bottom also plots the average event counts over a moving window of 2 seconds. When $\rho = 0$, there is no penalty on triggering MPC and, as shown in Fig. 1, the RL agent correctly triggers MPC for every sampling time, and the path tracking error is the smallest (also see Fig. 4). When $\rho = 0.001$, the RL agent tends to trigger an event when the tracking error is large, and keeps silent when the error is and/or going to be around 0, as shown in Fig. 2. In this case, the average relative event frequency is around 0.5. Finally, when $\rho = 0.01$, the penalty on event frequency outweighs the other term in (15). Therefore, the event pattern seems to span the time space evenly. In other words, the RL agent simply extends the sampling time by roughly 5 times. Note that this is different from extending sampling time of a time-triggered MPC, as in the latter case the control action is held constant during the extended sample period, while in RLeMPC it is time varying.

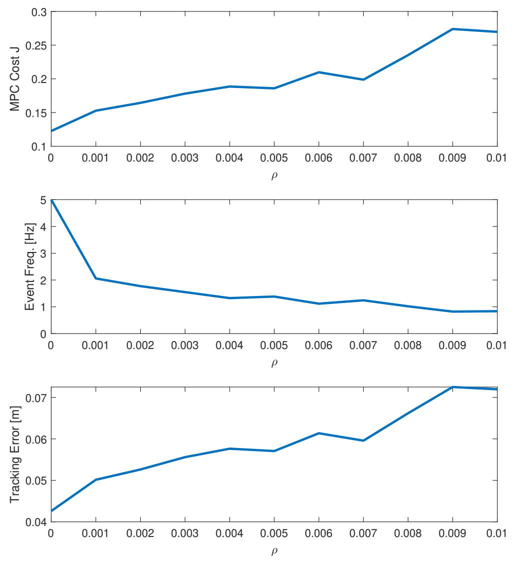

Fig. 4. Results on different level of $\rho$. *Top*: MPC running cost function $J$. *Middle*: Average event frequency throughout episode. *Bottom*: Average lateral position error.

Furthermore, Fig. 4 compares the performance of event-triggered MPC with RL agents trained by different hyper-parameter $\rho$. When $\rho = 0$, RL triggers MPC at every time step, resulting in an event frequency of 5 Hz (corresponding to the sampling time of $dt = 200$ms) and smallest tracking error. As the value of $\rho$ increases, the rewards function (15) penalizes more on triggering MPC, resulting in less frequent event and higher MPC cost $J_{\text{mpc}}$. Note that in prior work [16], the event frequency is indirectly influenced by the threshold parameters, whose impact on the event frequency is not intuitive to understand. In contrast, in the proposed RLeMPC framework, one can view $\rho$ as a calibratible parameter that directly impacts the balancing between control performance and MPC computational load. Finally, Fig. 4 also plots the average path tracking error. Note that since MPC cost function (18) includes penalty term on excessive steering, the MPC cost $J_{\text{mpc}}$ as used in (15) is higher than the path tracking error.

Finally, as comparison, we implemented the threshold-based event-trigger policy used in [16], where the results are shown in Fig. 5 and Table I. The threshold-based event-trigger policy is manually calibrated such that the closed-loop system achieves an average tracking error that is comparable to RLeMPC with $\rho = 0.001$ as shown in Fig. 2. From Fig. 5, it is clear that the threshold-based policy is very sensitive to the lateral error. When the error is high, it triggers event at almost every sampling time. In contrast, RLeMPC distributes the event more evenly, such that it triggers event only every

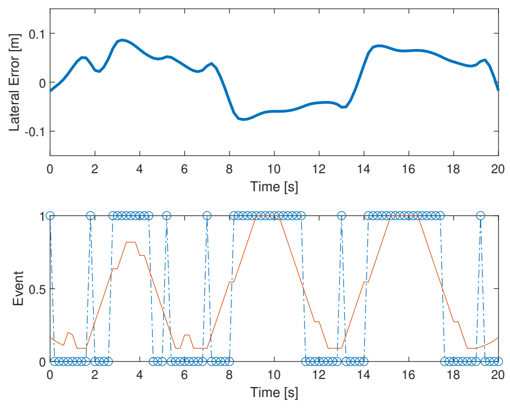

Fig. 5. Results with manually designed threshold-based event-trigger policy. *Top*: Lateral position error. *Bottom*: Event with moving average.

TABLE I
COMPARISON BETWEEN RL-BASED AND THRESHOLD-BASED POLICIES.

| Policy | RL ($\rho = 0.001$) | Threshold-based |
|---|---|---|
| Ave Inter-Event Time [ms] | 526 | 417 |
| Ave Tracking Error [$m$] | 0.0509 | 0.0514 |
| Max Tracking Error | 0.0772 | 0.0863 |

other sampling time even when the error is high (see Fig. 2). Key metrics are compared in Table I. Specifically, RLeMPC clearly outperforms in terms of average event frequency (as measured by inter-event time) and control performance (as measured by average and maximum tracking error).

## V. CONCLUSION

In this paper, we proposed an algorithm called RLeMPC to utilize reinforcement learning (RL) to trigger events in the event-triggered model predictive control (MPC) framework. More specifically, the optimal event-trigger policy was learnt by an RL agent through interactions with the MPC closed-loop system, whose dynamical behavior was assumed to be unknown to the RL agent. In particular, the least-square temporal difference Q-learning algorithm was used to train the RL agent with linear Q-function. The balance between control performance and event frequency can be specified through a weighting factor in the immediate reward function. Compared to existing literature on event-triggered MPC, the proposed algorithm does not require any knowledge of the MPC closed-loop system behavior, and provides a direct calibratible parameter to balance between control performance and MPC computation load. Finally, we applied the proposed RLeMPC to an autonomous vehicle path following problem as demonstration, where it was shown that RLeMPC outperformed threshold-based event-trigger mechanism used in literature. Future work directions include the use of deep Q-learning to capture the nonlinear event-trigger policy, as well as training the RL agent that works for all possible hyper-parameter $\rho$.

## REFERENCES

[1] M. Donkers, W. Heemels, D. Bernardini, A. Bemporad, and V. Shneer, "Stability analysis of stochastic networked control systems," *Automatica*, vol. 48, no. 5, pp. 917–925, 2012.

[2] M. Rubagotti, D. Barcelli, and A. Bemporad, "Robust explicit model predictive control via regular piecewise-affine approximation," *International Journal of Control*, vol. 87, no. 12, pp. 2583–2593, 2014.

[3] J. Chen, M. Liang, and X. Ma, "Probabilistic analysis of electric vehicle energy consumption using MPC speed control and nonlinear battery model," in *2021 IEEE Green Technologies Conference*, Denver, CO, April 7–9, 2021.

[4] S. Di Cairano, H. E. Tseng, D. Bernardini, and A. Bemporad, "Vehicle yaw stability control by coordinated active front steering and differential braking in the tire sideslip angles domain," *IEEE Trans. Control Syst. Techn.*, vol. 21, no. 4, pp. 1236–1248, 2012.

[5] 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 (IV)*, Seoul, Korea, 2015, pp. 1094–1099.

[6] J. Chen, A. Behal, and C. Li, "Active cell balancing by model predictive control for real time range extension," in *2021 IEEE Conference on Decision and Control*, Austin, TX, USA, December 13–15, 2021.

[7] J. Wurts, J. Dallas, J. L. Stein, and T. Ersal, "Adaptive nonlinear model predictive control for collision imminent steering with uncertain coefficient of friction," in *2020 American Control Conference*. Denver, CO, USA, July 2020.

[8] M. Jost, G. Pannocchia, and M. Mönnigmann, "Online constraint removal: accelerating MPC with a Lyapunov function," *Automatica*, vol. 57, pp. 164–169, 2015.

[9] D. Liao-McPherson, M. M. Nicotra, A. L. Dontchev, I. V. Kolmanovsky, and V. Veliov, "Sensitivity-based warmstarting for nonlinear model predictive control with polyhedral state and control constraints," *IEEE Transactions on Automatic Control*, 2019.

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

[11] 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.

[12] 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.

[13] 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.

[14] R. Badawi and J. Chen, "Enhancing enumeration-based model predictive control for dc-dc boost converter with event-triggered control," in *2022 European Control Conference*, London, UK, July 12–15, 2022.

[15] F. D. Brunner, M. Heemels, and F. Allgöwer, "Robust self-triggered MPC for constrained linear systems: A tube-based approach," *Automatica*, vol. 72, pp. 73–83, 2016.

[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] 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)*, Miami, FL, USA, 2018, pp. 943–950.

[18] 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.

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

[20] 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.

[21] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[22] C. J. C. H. Watkins, "Learning from delayed rewards," 1989.

[23] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *The Journal of Machine Learning Research*, vol. 4, pp. 1107–1149, 2003.