2018 IEEE Conference on Decision and Control (CDC)
Miami Beach, FL, USA, Dec. 17-19, 2018

Deep Reinforcement Learning for Event-Triggered Control

Dominik Baumann[1,*], Jia-Jie Zhu[2,*], Georg Martius[2], and Sebastian Trimpe[1]

Abstract— **Event-triggered control (ETC) methods can achieve high-performance control with a significantly lower number of samples compared to usual, time-triggered methods. These frameworks are often based on a mathematical model of the system and specific designs of controller and event trigger. In this paper, we show how deep reinforcement learning (DRL) algorithms can be leveraged to simultaneously learn control and communication behavior from scratch, and present a DRL approach that is particularly suitable for ETC. To our knowledge, this is the first work to apply DRL to ETC. We validate the approach on multiple control tasks and compare it to model-based event-triggering frameworks. In particular, we demonstrate that it can, other than many model-based ETC designs, be straightforwardly applied to nonlinear systems.**

I. Introduction

In modern engineering systems, feedback loops are often closed over (wired or wireless) communication networks [1], [2]. Examples for these networked control systems (NCSs) include smart buildings, where sensors and actuators are deployed to regulate the indoor climate; swarms of drones, which exchange information for coordinated flight; or autonomous driving, where communication between the vehicles allows for adaptive traffic control. When multiple control loops use the same network as in these examples, communication becomes a shared and therefore limited resource. Classical control methods typically ignore this fact and take sufficient communication resources for granted. Data is exchanged in a time-triggered fashion between components of the control loops irrespective of whether an update is actually needed. In recent years, the research community in event-triggered control (ETC) has had remarkable success in showing that the amount of samples in feedback loops can be reduced significantly compared to those time-triggered approaches (see experimental studies [3]–[5] for example). In ETC, transmission of data and thus closing of the feedback loop is triggered only on certain events, *e.g.*, an error growing too large. This way, feedback happens only when necessary, and significant communication savings can be realized.

There exists a large variety of methods to design event-triggered controllers, see *e.g.*, [6], [7] for an overview. Most of these methods are rooted in classical control theory and based on a model of the process to be controlled. In contrast to this, we show herein that *deep reinforcement learning* (DRL) algorithms can be leveraged in order to learn both control *and* communication law from scratch without the need for a dynamics model. We formulate resource-aware control as a reinforcement learning (RL) problem, where the learning agent optimizes its actions (control input and communication decision) so as to maximize some expected reward over a time horizon. The reward function is composed of two terms, one capturing control performance, and one that gives rewards for time steps without communication. This way, the agent learns to control the system with good performance, but without communicating all the time.

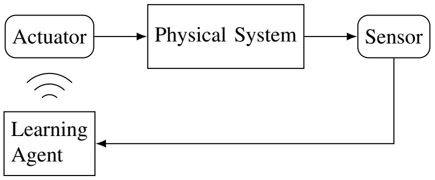

Fig. 1. Learning of event-triggered control. The learning agent continuously receives sensor inputs, but has to transmit control signals over a resource-limited wireless network. The agent learns both control and communication; that is, (i) what actuator command to send, and (ii) when to send it.

More specifically, we consider the setup in Fig. 1. While the learning agent is directly connected to the sensor and thus receives its measurements continuously, it must transmit actuator commands over a wireless network link. Thus, we seek to reduce communication of control inputs. We propose two approaches for learning ETC in this setting. In the first approach, we assume that a time-triggered feedback controller is given, and we learn only the communication policy. As an alternative, we learn both control and communication policy simultaneously. This can be regarded as *end-to-end learning*. In DRL, end-to-end learning (*e.g.*, [8]) typically refers to learning the complete control policy from raw sensor data to actuator commands 'end to end,' without (artificially) separating into sub-tasks such as filtering, planning, and tracking. In the context of ETC, end-to-end thus emphasizes learning of both control and communication simultaneously, rather than separating the two. This is particularly interesting as the separation principle does not generally hold in ETC; that is, optimizing controller and communication structure separately, as often done in practice, does not necessarily yield the overall optimal event-triggered control law [9]. End-to-end DRL is a way to overcome this separation.

By means of numerical examples, we demonstrate that end-to-end learning of ETC is feasible. Moreover, we com-

[*] Equal contribution.

[1] Intelligent Control Systems Group, Max Planck Institute for Intelligent Systems, Stuttgart/Tübingen, Germany. Email: {dominik.baumann, sebastian.trimpe}@tuebingen.mpg.de.

[2] Autonomous Learning Group, Max Planck Institute for Intelligent Systems, Tübingen, Germany. Email: {jia-jie.zhu, georg.martius}@tuebingen.mpg.de.

This work was supported in part by the German Research Foundation (DFG) within the priority program SPP 1914 (grant TR 1433/1-1), the European Unions Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No 798321, the Cyber Valley Initiative, and the Max Planck Society.

978-1-5386-1395-5/18/$31.00 ©2018 IEEE

pare to some common model-based ETC approaches. The comparison reveals that, for linear settings with an accurate model available, model-based ETC typically cannot be outperformed by the proposed DRL approach—at least, at medium to high average communication rates. In some cases, however, DRL can find superior policies at very low communication rates, where model-based ETC yields unstable solutions. In contrast to common ETC methods, the proposed learning approach straightforwardly applies also to nonlinear control problems.

Contributions: The contributions of this work can be summarized as follows:

- Proposal of deep reinforcement learning (DRL) to learn event-triggered controllers from data;
- learning of communication policy only (with a given controller) with policy gradients [10];
- end-to-end learning of control *and* communication policy with deep deterministic policy gradient (DDPG) algorithm [11];
- demonstration of feasibility of DRL in numerical benchmark problems; and
- comparison to model-based ETC methods.

Related work: Using machine learning techniques to learn feedback controllers from data has been considered in previous works, see *e.g.*, [8], [10]–[19] and references therein. These works typically consider learning of control policies only, without incorporating the cost of communication such as when controller and plant are connected over a network link.

There exists a large body of work on ETC methods, see *e.g.*, [6], [7] and references therein for an overview. Using RL for ETC is not discussed there and has generally received less attention. Model-free RL for event-triggered controllers has for example been proposed in [20], where an actor-critic method is used to learn an event-triggered controller and stability of the resulting system is proved. However, the authors consider a predefined communication trigger (a threshold on the difference between current and last communicated state); that is, they do not learn the communication policy from scratch. Similarly, in [21], an approximate dynamic programming approach using neural networks is implemented to learn event-triggered controllers, again with a fixed error threshold for triggering communication. In [22], the authors propose an algorithm to update the weights of a neural network in an event-triggered fashion. Model-based RL is used in [23] to simultaneously learn an optimal event-triggered controller with a predefined fixed communication threshold, and a model of the system. To the authors' knowledge, no prior work considers end-to-end learning of control and communication in ETC, which is the main contribution herein.

In [24], learning is proposed to improve communication behavior for event-triggered state estimation. Other than here, the idea is to improve accuracy of state predictions through model-learning. A second event-trigger is introduced that triggers learning experiments only if the mathematical model deviates from the real system.

Outline: The next section continues with a short introduction to ETC and a more detailed description of DRL with particular focus on approaches for continuous state-action spaces. The proposed approaches for DRL of ETC are then introduced in Sec. III. Section IV presents numerical results, and the paper concludes with a discussion in Sec. V.

II. Background

We consider a nonlinear, discrete-time system disturbed by additive Gaussian noise,

$$x_{k+1} = f\left(x_k, u_k\right) + v_k \tag{1a}$$

$$y_k = x_k + w_k, \tag{1b}$$

with k the discrete-time index, $x_k, y_k, v_k, w_k \in \mathbb{R}^n$, $u_k \in \mathbb{R}^l$, and v_k, w_k mutually independent Gaussian random variables with probability density functions (PDFs) $\mathcal{N}\left(v_k; 0, \Sigma_\mathrm{p}\right)$ and $\mathcal{N}\left(w_k; 0, \Sigma_\mathrm{m}\right)$, and variances Σ_p and Σ_m.

A. Event-triggered Control

In ETC communication is not triggered by a clock, but by the occurrence of certain events. The input is defined as

$$u_k = \begin{cases} \mathcal{K}_k\left(x_k\right) & \text{if } \gamma_k = 1 \\ u_{k-1} & \text{if } \gamma_k = 0 \end{cases} \tag{2}$$

where γ_k denotes the communication decision and $\mathcal{K}_k$ the control law. Whether to communicate is decided based on a triggering law $\mathcal{C}_k$,

$$\gamma_k = 1 \iff \mathcal{C}_k\left(x_k, \hat{x}_k\right) \geq 0, \tag{3}$$

where $\hat{x}_k$ defines the state of the system at the last time instant a control input has been applied. An example for such a triggering law would be

$$\mathcal{C}_k\left(x_k, \hat{x}_k\right) \geq 0 \iff \left\|x_k - \hat{x}_k\right\|_2 \geq \delta, \tag{4}$$

with δ being a predefined threshold. Intuitively speaking this would mean we communicate the control input if the current state of the system deviates too much from the state at the last communication slot. There are many other ETC schemes following similar ideas; we refer to [6], [7] for more detailed overviews.

In this work, we want to learn both the control law $\mathcal{K}_k$ and the triggering policy (γ_k as a function of the observables) using DRL approaches.

B. Deep Reinforcement Learning

We give a brief introduction to RL in general and present the two baseline algorithms we later focus on in Sec. III.

The main goal in RL is to learn an optimal policy by trial and error while interacting with the environment. Mathematically, this can be formulated as a Markov decision process (MDP). In an MDP, we consider the setting where an agent interacts with the environment. At every time step, the agent selects an action a_k, from the action space A, based on its current state s_k, from the state space S, according to a policy $\pi\left(a_k|s_k\right)$.[1] The agent receives a reward r_k and the state transitions to the next state s_{k+1} according to the state transition probability $p(s',r|s=s_k,a=a_k)$. The goal of the RL agent is to maximize the expected discounted reward $\mathbb{E}[R_k]=\mathbb{E}\left[\sum_{i=0}^{\mathrm{T}-1}\zeta^i r_{k+i}\right]$, where $\zeta\in(0,1]$ is the discount factor.[2] There are generally two types of RL methods: model-free and model-based. One model-free method to achieve the goal is to learn a value function $v_\pi(s)\overset{!}{=}\mathbb{E}\left[R_k|s_k=s\right]$, which denotes the expected return in case policy π is followed from state s onwards. The value function $v_\pi(s)$ follows the Bellman equation [25],

$$v_\pi(s)=\sum_a \pi\left(a|s\right)\sum_{s',r}p\left(s',r|s,a\right)\left[r+\zeta v_\pi(s')\right], \qquad (5)$$

which can then be maximized to find the optimal state values.

Similarly, one can estimate a state-action value function (Q-function) $Q_\pi\left(s,a\right)\overset{!}{=}\mathbb{E}\left(R_k|s_k=s,a_k=a\right)$ which determines the expected return for selecting action a in state s and following the policy π thereafter. In an MDP, the optimal action in the current state can be derived by maximizing the Q-function. If the transition probabilities p are available, this can, *e.g.*, be done using (exact) dynamic programming (DP). In cases where the model is not known, we resort to RL (simulation-based approximate DP) methods. In such cases, the Q-function can be learned using the Q-learning algorithm presented in [26],

$$\begin{aligned}Q(s_k,a_k)\leftarrow{}& Q(s_k,a_k)\\&+\alpha\left(r_k+\zeta\max_{a'_k}Q\left(a'_k,s'_k\right)-Q(s_k,a_k)\right).\end{aligned} \qquad (6)$$

The Q-learning algorithm updates the Q-function using the collected experience (s_k,a_k,r_k,s'_k). For a more detailed introduction to RL, see [27].

This basic RL approach has successfully been applied to low-dimensional tasks with discrete state and action space [27]. For controlling a dynamical system, we usually deal with a continuous state and action space, which might be of high dimension for complex systems. Continuous spaces could be discretized, but the discretization needs to be very fine for high-performance control. This, in turn, leads to very high-dimensional state and action spaces imposing unreasonable computational complexity and hampering convergence speed drastically.

To the rescue come parametrized function approximators. In machine learning, deep neural networks (DNNs) have widely been used to handle high-dimensional tasks. Recently, they have also been applied to RL, giving rise to the field of DRL. For instance, in deep Q-learning [28], the state-action function Q is approximated with a DNN, making it possible to solve complex tasks in high-dimensional continuous state spaces. However, this algorithm only works for discrete action spaces.

One possible solution to this problem is the actor-critic architecture [27]. The actor outputs continuous actions while the critic estimates the value function. Both can be implemented using DNNs. One such algorithm is deep deterministic policy gradient (DDPG) [11], [29], which we introduce in the following.

As an alternative, we look at policy search methods that directly learn a policy without a Q-function. Specifically, we will present the policy gradient algorithm [10] and the trust region policy optimization (TRPO) algorithm [30].

1) DDPG: The DDPG algorithm, and a variation of it, are presented in [11], [31]. For completeness, we restate the main derivations.

DDPG is an actor-critic algorithm with two networks. One is the actor network μ, parametrized by θ^μ that takes the state s_k as input and outputs an action a_k. Additionally, we have the critic network Q, parametrized by θ^Q, which takes state and action as input and outputs a scalar estimate of the value function, the Q-value $Q\left(s_k,a_k\right)$. The updates of the critic network are close to the original formulation of the Q-learning algorithm given in (6). Adapting (6) to the described neural network setting leads to minimizing the loss function

$$\begin{aligned}L_Q\left(s_k,a_k|\theta^Q\right)=\Bigg(&Q\left(s_k,a_k|\theta^Q\right)\\&-\left(r_k+\zeta\max_{a'_k}Q\left(s'_k,a'_k|\theta^Q\right)\right)\Bigg)^2.\end{aligned} \qquad (7)$$

For continuous action spaces, equation (7) is not tractable, as we would have to maximize over the next-state action a'_k. Instead, we take the next-state action $a'_k=\mu\left(s'_k|\theta^\mu\right)$ of the actor network. Inserting this in equation (7) leads to

$$\begin{aligned}L_Q\left(s_k,a_k|\theta^Q\right)=\big(&Q\left(s_k,a_k|\theta^Q\right)\\&-\left(r_k+\zeta Q\left(s'_k,\mu\left(s'_k|\theta^\mu\right)|\theta^Q\right)\right)\big)^2.\end{aligned} \qquad (8)$$

Based on this loss function, the critic can learn the value function via gradient descent. Clearly, a crucial point is the quality of the actor's policy. The actor tries to minimize the difference between its current output a and the optimal policy a^*,

$$L_\mu\left(s_k|\theta^\mu\right)=\left(a_k-a_k^*\right)=\left(\mu\left(s_k|\theta^Q\right)-a_k^*\right)^2. \qquad (9)$$

The true optimal action a_k^* is of course unknown. As simply estimating it would require to solve a global optimization problem in continuous space, the critic network can instead provide a gradient that leads to higher estimated Q-values: $\nabla_{a_k}Q\left(s_k,a_k|\theta^Q\right)$. Computing this gradient is much faster. This was first introduced in [32]. The gradient implies a change in actions, which is used to update the actor network in this direction by backpropagation. In particular, for an observed state s_k and action a_k, the parameters of the actor network are changed according to

$$\nabla_{\theta^\mu}J=\nabla_{a_k}Q\left(s_k,a_k|\theta^Q\right)\nabla_{\theta^\mu}\mu\left(s_k|\theta^\mu\right) \qquad (10)$$

approximating the minimization of (9).

Two general problems arise from this approach. For most optimization algorithms, it is usually assumed that samples

[1] If we want to learn a controller for a dynamical system, often $s_k\equiv x_k$ and $a_k\equiv u_k$ holds. However, this is not necessarily the case and, in particular, not in the setup we shall develop herein.

[2] To simplify the formulation, we consider the episodic case with $k\in[0,T-1]$.

are independent and identically distributed. This is obviously not the case if we sequentially explore an environment. To resolve this, a replay buffer of fixed size that stores tuples (s_k, a_k, r_k, s_{k+1}) is used. Actor and critic are now updated by uniformly sampling mini-batches from this replay buffer.

The second problem is that the update of the Q-network uses the current Q-network to compute the target values (see (8)). This has proved to be unstable in many environments. Therefore, copies of actor and critic networks, $Q'\left(s_k, a_k|\theta^{Q'}\right)$ and $\mu'\left(s_k|\theta^{\mu'}\right)$ are created and used to calculate the target values. The copies are updated by slowly tracking the learned network,

$$\theta' = \kappa\theta + (1-\kappa)\,\theta', \tag{11}$$

with $\kappa \ll 1$. This typically leads to more robust learning.

In Sec. III-A, we will show how this algorithm can be used to jointly learn controller and communication behavior.

2) Trust Region Policy Optimization (TRPO): A second approach is to perform direct parameter search without a value function. This is referred to as direct policy search or policy gradient [10]. A parametrized policy π is adapted directly to maximize the expected reward. Since, without a model, analytical gradients of the reward function are not available, policy gradient methods use stochastic policies and adapt them to increase the likelihood of a high reward. Formally, let the policy $\pi(s_k;\theta^\pi)$ representing $p(a_k|s_k)$ be parametrized by θ^π in a differentiable way. Now we aim to maximize the utility $J(\theta^\pi) = \mathbb{E}_{a_k\sim\pi(s_k;\theta^\pi)} R(s_k, a_k)$. Policy gradient methods follow the gradient estimator of J for a given trajectory:

$$\nabla_{\theta^\pi} J(\theta^\pi) = \sum_{k=0}^{T-1} R_k \nabla_{\theta^\pi} \log \pi(s_k;\theta^\pi)\,. \tag{12}$$

A recent advance of policy gradient methods is given by the Trust Region Policy Optimization [30] (TRPO) that uses a surrogate optimization objective and a trust region approach for updating the policy efficiently. In terms of theoretical guarantees, this algorithm ensures monotonic improvement of the policy performance, given the amount of training samples is large. We will use this method in Sec. III-B to learn the controller and triggering policy independently.

III. Approach

We present two approaches to learn resource-aware control. First, we consider learning communication structure and controller end-to-end. The policy should then output both, the communication decision and the control input,

$$(\gamma_k, u_k) = \pi_{\text{combined}}\left(s_k\right) = \pi_{\text{combined}}\left(x_k, u_{k-1}\right), \tag{13}$$

where γ_k is a binary variable with $\gamma_k = 0$ indicating no communication. Alternatively, we start with a control strategy for the system without communication constraints, either learned or designed. The goal is then to learn the communication structure, *i.e.*, a policy

$$\gamma_k = \pi_{\text{comm}}\left(s_k\right) = \pi_{\text{comm}}\left(x_k, u_k, u_{k-1}\right). \tag{14}$$

This strategy requires us to separate the design of controller and communication structure.

For both settings, the state of the RL agent includes the current state x_k of the system and the last control input u_{k-1}. This is necessary, as in case of no communication, u_{k-1} will be applied again, so knowledge of the last control input is needed for the problem to form an MDP. In (14), the state is further augmented and also includes the current control input u_k. The RL agent learns a communication policy, *i.e.*, it needs to decide, whether u_k or u_{k-1} will be applied. Therefore it needs knowledge of both. The action a_k of the RL agent consists of the communication decision for the separated policy, and of communication decision and control input in the combined case.

In RL, the reward function typically depends on the states and actions of the system. We additionally consider communication, thus we arrive at a reward function of the form

$$r_k = -x_k^\top Q x_k - u_k^\top R u_k - \lambda\gamma_k, \tag{15}$$

where λ is a hyper-parameter. During training, the agent receives negative rewards for bad performance and for communication. In an episodic reinforcement learning task, where agents' interaction with the environment is divided into episodes, an additional constant positive reward is often given to the agent to prevent undesired early termination of the episodes, *e.g.*, the pole dropping for the cart-pole system.

A. Joint Learning of Communication and Control

To learn resource-aware controllers, we consider both the discrete action space (the decision whether to communicate) and the continuous action space (the control input that should be applied). This is related to the idea of reinforcement learning in parameterized action space [31], [33].

This framework considers a parameterized action space Markov decision process (PAMDP), which involves a set of discrete actions $A_\text{d} = \{d_1, d_2, \ldots, d_k\}$. Each discrete action $d \in A_\text{d}$ is associated with m_d continuous parameters $\left\{p_1^\text{d}, p_2^\text{d}, \ldots, p_{m_\text{d}}^\text{d}\right\} \in \mathbb{R}^{m_\text{d}}$. An action is represented by a tuple $\left(d, p_1^\text{d}, \ldots, p_{m_\text{d}}^\text{d}\right)$. This leads to the following action space: $A = \cup_{\text{d}\in\text{A}_\text{d}}\left(d, p_1^\text{d}, \ldots, p_{m_\text{d}}^\text{d}\right)$.

In our case, there are two discrete actions, d_1 and d_2, where d_1 corresponds to the decision to communicate the control input ($\gamma_k = 1$). Accordingly, the action d_1 has $m_{\text{d}_1} = 1$ continuous parameter $p_1^{\text{d}_1} = u_k$, which is the control input. Action d_2 does not have any continuous parameter, as we apply the last control input again.

As stated in Sec. II-B, we consider the DDPG algorithm[3], where both actor and critic network are implemented using DNNs. The architecture is depicted in Fig. 2. The actor network outputs continuous values for all actions in the action space, *i.e.*, we have

$$a_k = (d_1, d_2, u_k) = \pi_{\text{combined}}\left(x_k, u_{k-1}\right). \tag{16}$$

[3] Our implementation is based on the non-parameterized DDPG framework provided in [34].

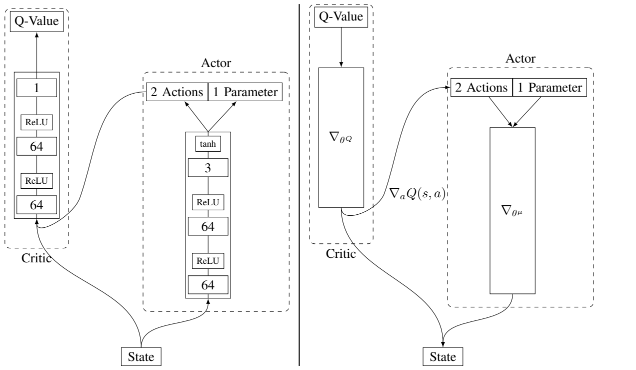

Fig. 2. Visualization of the actor-critic network structure adapted from [31]. On the left, the general network architecture, showing the units and activation function of each layer. Each block represents one layer of the network with the number describing the number of neurons. The smaller blocks indicate the activation functions. On the right, the update of the actor using back-propagation.

This is different from (13), as we do not receive a discrete parameter γ_k, but continuous values for all parameters a_k. To obtain a discrete decision, we determine the communication decision by

$$\gamma_k = \begin{cases} 1 & \text{if } d_{1,k} > d_{2,k} \\ 0 & \text{otherwise.} \end{cases} \tag{17}$$

The continuous parameter (the control input u_k) is directly obtained as an output of the actor. The output of the actor and the current state then serve as input for the critic, which estimates the Q-function value. This structure has been applied to a gaming environment in [31].

During training, exploration is done in an ϵ-greedy fashion. With probability ϵ, we select a random discrete action (whether to communicate). Besides the ϵ-greedy exploration we add exploration noise in form of an Ornstein Uhlenbeck process to the output of the actor as has been successfully demonstrated in [11]. Pseudo-code of this approach is presented in Algorithm 1.

B. Learning Communication only

An alternative to the aforementioned end-to-end approach is to separately learn the communication strategy and the stabilizing controller. In this approach, a control policy is first fully trained using a high-performing RL algorithm, *e.g.*, TRPO [30]. Instead of hand-engineering the communication strategy, we propose to use policy gradients [10] to learn this communication structure. In essence, the trained controller computes the control input in every time-step, whereas another learning agent controls whether to send this control input to the system, thus implementing (14).

The general scheme is related to hierarchical reinforcement learning [35] and gated recurrent neural networks [36]. We discuss preliminary experimental results of this alternative approach, in relatively challenging tasks, in Sec. IV-D.

IV. Validation

In this section, we validate the proposed DRL approaches through several numerical simulations. For the algorithm introduced in Sec. III-A, which jointly learns communication behavior and controller, we show the general applicability as a proof of concept on the inverted pendulum, compare to several model-based ETC algorithms on the same platform, and show its general applicability for nonlinear tasks. In Sec. IV-D, we demonstrate learning resource-aware locomotion tasks using the algorithm presented in Sec. III-B.[4]

Algorithm 1 Jointly learn communication and controller (adapted from [11]).

```
Initialize $\epsilon$
Randomly initialize critic DNN $Q\left(s_k, a_k|\theta^Q\right)$ and actor DNN $\mu\left(s_k|\theta^\mu\right)$ with weights $\theta^Q$ and $\theta^\mu$.
Initialize target networks $Q'$ and $\mu'$ with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer $\mathcal{R}$
for episode $= 1$ to $N$ do
    Receive initial observation state $s_1$
    for k $= 1$ to $M$ do
        Generate uniformly distributed $\phi \in [0, 1]$
        if $\phi < \epsilon$ then
            Generate $\xi \sim B\,(2, 0.5)$ from Bernoulli distribution
            if $\xi == 1$ then
                Choose discrete action $d_1$
            else
                Choose discrete action $d_2$
            end if
        else
            Select $a_k = \mu\left(s_k|\theta^Q\right)$ and apply exploration noise to the actor output
            Get communication decision $\gamma_k$ using (17)
        end if
        Execute action $a_k$, receive reward $r_k$ and state $s_{k+1}$
        Store transition $(s_k, a_k, r_k, s_{k+1})$ in $\mathcal{R}$
        Sample random mini-batch from $\mathcal{R}$
        Update critic by minimizing loss function (8)
        Update actor policy using sampled policy gradient (10)
        Update target networks according to equation (11)
    end for
end for
```

The numerical simulations presented in this section were carried out in environments adapted from the OpenAI Gym[5]. The OpenAI Gym provides simulation models of different classical control tasks, such as the inverted pendulum and the cart-pole system, as well as physics simulation systems, Atari games, and many more. For our approaches, we augment the reward functions provided in the OpenAI Gym according to (15). The simulations are carried out on a cluster utilizing parallel runs for the training and testing processes with randomized seeds.

A. Proof of Concept

As a proof of concept, we apply the learning algorithm presented in Sec. III-A to the inverted pendulum. The inverted pendulum consists of a pendulum attached to a motor with the goal to keep the pendulum close to its upright position at $\theta = 0\,\mathrm{rad}$. We assume process and measurement

[4] Code of representative examples and video of resource-aware locomotion are available at `https://sites.google.com/view/drlcom/`.

[5] `https://gym.openai.com/`

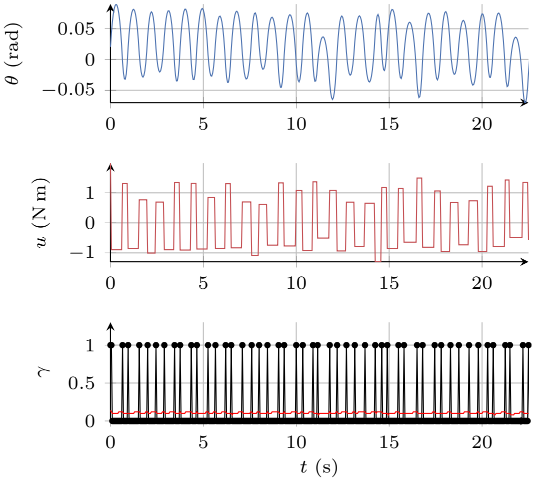

Fig. 3. Stabilization of the inverted pendulum with an event-triggered controller learned with the method presented in Sec. III-A. The plots show, from top to bottom, the angle of the pendulum θ, the control input u, and the communication (decision γ in black, average communication in red). The average communication here and in following plots is computed as a moving average over 50 samples.

noise as in (1) and the initial state also a Gaussian distributed random variable with $x(0) \sim \mathcal{N}(x(0); 0, \Sigma_0)$. The standard deviation of noise and initial position was chosen to be 10^{-4}.

The simulation environment provides upper and lower bounds of $\pm 2\,\mathrm{N\,m}$ on the input torque that may be applied to the pendulum. One discrete time step lasts $50\,\mathrm{ms}$.

We train the controller using the joint learning approach detailed in Sec. III-A. The hyper-parameter λ in (15) is tuned by a grid search of 25 values between 0.01 and 100. Different hyper-parameter values correspond to different communication rates and controller performances. For each hyper-parameter setting and task, we carry out 5 randomized training processes using different random seeds, each consists of one million training iterations. During performance evaluation, we carry out 100 randomized test episodes for each of the 5 trained agents for each hyper-parameter setting with each episode lasting 500 discrete time steps.

Results of one such test episode can be seen in Fig. 3. The plot is a representative example for the results obtained from the different agents and test episodes. The pendulum system remains stable with the angle staying well within $\pm 0.1\,\mathrm{rad}$, while significantly saving communication. Here we observe a saving rate of around 90 %. Further it can be seen that the learning approach does not converge to a triggering law with fixed threshold. The threshold at which communication of a new control input is triggered is dynamically changing throughout the experiment.

In general, stability is very important in control tasks. However, most works in reinforcement learning aim at achieving optimal control instead of stability. For policy iteration methods (which we use herein), guarantees on monotonic improvement of the policy can be given, as discussed for instance in [30], [37]. However, the resulting controller derived in this work is approximated by a deep neural network, which is highly nonlinear, thus analyzing the stability of the system is not straightforward. Further, finding optimal control policies does not necessarily imply stability, but the connection is more subtle (cf. [38], [39]). For the time being, this renders the effort of analyzing stability intractable. While stability of DRL is an important topic for research, this example is a proof of concept that joint control and communication policies can be found with DRL.

B. Comparison

We compare the performance of the learning approach to common model-based ETC designs on the inverted pendulum. For balancing, the inverted pendulum can be approximated as a linear system and methods from linear control theory may be used. We consider the intuitive ETC algorithm, where we only communicate, if the state deviates too much from its desired position. The state of the inverted pendulum consists of its angle θ and its angular velocity $\dot{\theta}$, the desired value for both is zero. Hence, we apply the following control law

$$u_k = \begin{cases} Kx_k & \text{if } \|x_k\|_2 > \delta \\ u_{k-1} & \text{otherwise,} \end{cases} \tag{18}$$

where the matrix K is designed with an LQR approach using Q and R as in the reward function of the learning algorithm. Additionally, we compare to the approaches introduced in [40] and [41]. In both cases, we use the formulation as a periodic event-triggered control algorithm provided in [42], which is $\gamma_k = 1 \iff \|K\hat{x}_k - Kx_k\| > \delta \|Kx_k\|$ for [40], and $\gamma_k = 1 \iff \|\hat{x}_k - x_k\| > \delta \|x_k\|$ for [41]. The algorithms only give a communication threshold, but require a stabilizing controller K. For both, we used the same LQR as for (18).

The algorithms have different triggering laws but are all based on a fixed threshold δ. For comparison, we vary this threshold. After every experiment, we compute the quadratic cost and the average communication. These simulations revealed that communication savings up to around 60 % for [41], 70 % for [42], and 80 % for (18) are possible. When running similar simulations with the DDPG approach from Sec. III-A, we noted that the model-based approaches clearly outperform the learning approach. However, communication savings of 90 %, as observed in Fig. 3, cannot be achieved with these model-based approaches as they become unstable before. The learning agent, in contrast, is still able to come up with a good policy.

C. Swing-up

As previously stated, the presented DRL approach can also be applied to more challenging, *e.g.*, nonlinear, systems. In this section, we take on such a setting where the aforementioned ETC designs do not apply. The training and evaluation procedures in this section follow the same paradigm detailed in Section IV-A.

In Fig. 4, the inverted pendulum is presented again, but with the initial angle well beyond the linear region. As can

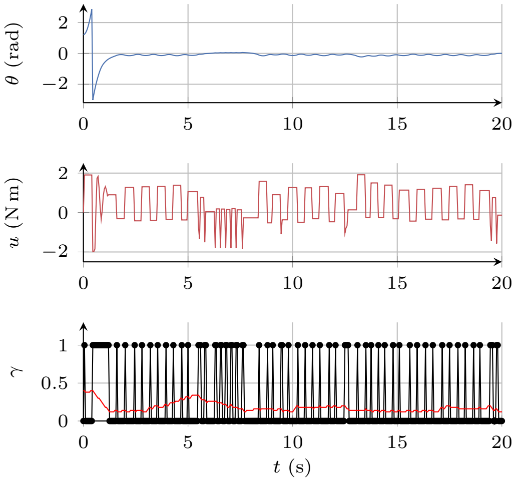

Fig. 4. Resource-aware swing-up of the inverted pendulum, showing the angle θ (top) and the communication (bottom, discrete decision in black, average communication in red). The jump observed in the beginning is due to the pendulum crossing π and thus immediately switching from π to $-\pi$.

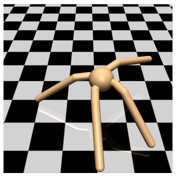

Fig. 5. The ant robot.

be seen, the agent is able to learn a resource-aware swing-up policy and then stabilize the pendulum around $\theta = 0\,\mathrm{rad}$ while saving around 80 % communication.

We also trained the learning agent on the cart-pole system, where it was similarly able to learn a stable policy while saving around 90 % of communication (with an underlying sample time of 25 ms).

D. Simulated locomotion

So far, we have addressed canonical tasks in optimal control using the proposed end-to-end approach. In this section, we move the focus to advanced tasks, *i.e.*, locomotion. We applied the proposed parameterized DDPG approach of Sec. III-A to resource-aware locomotion, but only with moderate success. This is possibly due to the lack of reward hand-engineering and is left for future work considerations. During our experiments, we discovered that learning controller and communication behavior separately, as explained in Sec. III-B, allows us to address even challenging tasks such as robotic locomotion. In this approach, we first train the agent with full communication using TRPO, typically using iteration numbers on the 10^6 order-of-magnitude. After the TRPO agent is trained, we train the communication strategy using a policy gradient approach with augmented reward as in (15) until we observe desired behaviors trading off performance and communication saving. Our experimental environment is based on the Mujoco physics simulation engine [43].

As an example we trained an ant (quadruped) robot (Fig. 5) in a simulated 3D locomotion task. It is a relatively challenging task considering the high-dimensional state space (111 states with 13 for position, 14 for velocity, 84 for external force) and under-actuation (8 actuators). To make matters worse, it can be easily toppled and is then subsequently not able to stand up. The underlying sampling time is 50 ms.

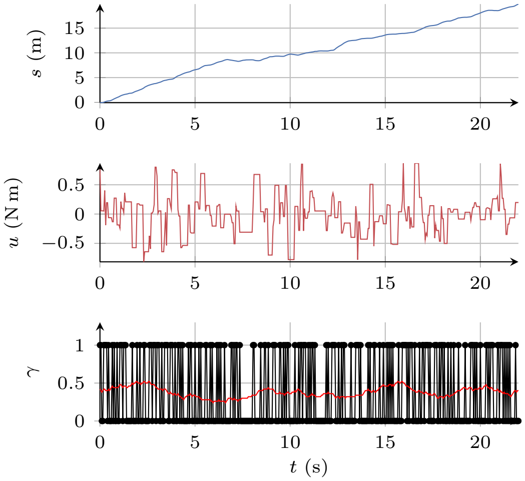

Fig. 6. Simulation of the Ant robot (see Fig. 5) learning to walk while saving communication, showing from top to bottom the position s of the center of mass, the input u, which is the torque applied to the hip motor, communication instants in black and the average communication in red.

As shown in Fig. 6, the ant learns to walk saving around 60 % of communication. We did observe that, during some of the runs, the resource-aware ant falls and causes worse performance. However, this happens only around 10 % of the time. As our method is task agnostic and not specifically engineered for locomotion tasks, we consider the performances and communication savings non-trivial.

V. Discussion

Most existing approaches in event-triggered control (ETC) rely on the availability of accurate dynamics models for the design of control law and event trigger. In contrast, we proposed the use of deep reinforcement learning (DRL) for simultaneously learning control *and* communication policies from simulation data without the need of an analytical dynamics model. For scenarios where an accurate linear model is available, the numerical comparisons herein have shown that common model-based ETC approaches are superior to the learning approach. This is to be expected because the model-based design fully exploits the model structure. For some cases, however, the DRL approach succeeded in finding stabilizing controllers at very low average communication rates, which the model-based design were unable to obtain. What is more, the key advantage of the learning-based approach lies in its versatility and generality. As the examples herein have shown, the same algorithm can be used to also learn control and communication policies for nonlinear problems, including complex ones like locomotion. In the presented example, significant communication savings of around 60 % were obtained.

One limitation of our current approaches is the zero-order hold (ZOH) employed at the actuator. Instead of ZOH, some model-based approaches perform predictions based on the dynamics model in case of no communication, and thus achieve better performance. This could also be done if learning agents are used and would lead to a two agent problem. The first agent continuously receives measurement updates and decides when to transmit data to the second agent. The second agent can continuously apply control inputs, which includes the possibility of making predictions based on a learned model. Investigating such more general learning architectures is an interesting and challenging topic for future work. Whether theoretical guarantees such as on stability and robustness can also be obtained for the learned controllers is another topic worthwhile to be investigated.

References

[1] J. P. Hespanha, P. Naghshtabrizi, and Y. Xu, "A survey of recent results in networked control systems," *Proc. IEEE*, vol. 95, no. 1, pp. 138–162, 2007.

[2] J. Lunze, *Control theory of digitally networked dynamic systems*. Springer, 2014.

[3] S. Trimpe and R. D'Andrea, "An experimental demonstration of a distributed and event-based state estimation algorithm," in *18th IFAC World Congress*, 2011, pp. 8811–8818.

[4] J. Araújo, M. Mazo, A. Anta, P. Tabuada, and K. H. Johansson, "System architectures, protocols and algorithms for aperiodic wireless control systems," *IEEE Trans. Ind. Informat.*, vol. 10, no. 1, pp. 175–184, 2014.

[5] V. S. Dolk, J. Ploeg, and W. M. H. Heemels, "Event-triggered control for string-stable vehicle platooning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 12, pp. 3486–3500, 2017.

[6] W. P. M. H. Heemels, K. H. Johansson, and P. Tabuada, "An introduction to event-triggered and self-triggered control," in *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, Dec 2012, pp. 3270–3285.

[7] M. Miskowicz, *Event-Based Control and Signal Processing*. CRC Press, 2016.

[8] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.

[9] C. Ramesh, H. Sandberg, L. Bao, and K. H. Johansson, "On the dual effect in state-based scheduling of networked control systems," in *Proc. of the 2011 American Control Conference*, June 2011, pp. 2216–2221.

[10] J. Peters and S. Schaal, "Reinforcement learning of motor skills with policy gradients," *Neural networks*, vol. 21, no. 4, pp. 682–697, 2008.

[11] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[12] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[13] S. Schaal and C. G. Atkeson, "Learning control in robotics," *IEEE Robotics & Automation Magazine*, vol. 17, no. 2, pp. 20–29, 2010.

[14] A. Marco, P. Hennig, S. Schaal, and S. Trimpe, "On the design of LQR kernels for efficient controller learning," in *Proc. of the 56th IEEE Conference on Decision and Control*, Dec. 2017.

[15] A. Doerr, D. Nguyen-Tuong, A. Marco, S. Schaal, and S. Trimpe, "Model-based policy search for automatic tuning of multivariate PID controllers," in *Proc. 2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 5295–5301.

[16] D. Baumann, A. Ascoli, R. Tetzlaff, L. Chua, and M. Hild, "Memristor-enhanced humanoid robot control system–Part II: Circuit theoretic model and performance analysis," *International Journal of Circuit Theory and Applications*, vol. 46, no. 1, pp. 184–220, 2018.

[17] P. Abbeel, A. Coates, and A. Y. Ng, "Autonomous helicopter aerobatics through apprenticeship learning," *The International Journal of Robotics Research*, vol. 29, no. 13, pp. 1608–1639, 2010.

[18] R. Calandra, A. Seyfarth, J. Peters, and M. P. Deisenroth, "Bayesian optimization for learning gaits under uncertainty," *Annals of Mathematics and Artificial Intelligence*, vol. 76, no. 1-2, pp. 5–23, 2016.

[19] P. P. Khargonekar and M. A. Dahleh, "Advancing systems and control research in the era of ML and AI," *Annual Reviews in Control*, 2018.

[20] K. G. Vamvoudakis and H. Ferraz, "Model-free event-triggered control algorithm for continuous-time linear systems with optimal performance," *Automatica*, vol. 87, pp. 412 – 420, 2018.

[21] X. Zhong, Z. Ni, H. He, X. Xu, and D. Zhao, "Event-triggered reinforcement learning approach for unknown nonlinear continuous-time system," in *2014 International Joint Conference on Neural Networks (IJCNN)*, July 2014, pp. 3677–3684.

[22] A. Sahoo, H. Xu, and S. Jagannathan, "Neural network-based event-triggered state feedback control of nonlinear continuous-time systems," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 3, pp. 497–509, 2016.

[23] X. Yang, H. He, and D. Liu, "Event-triggered optimal neuro-controller design with reinforcement learning for unknown nonlinear systems," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. PP, no. 99, pp. 1–13, 2017.

[24] F. Solowjow, D. Baumann, J. Garcke, and S. Trimpe, "Event-triggered learning for resource-efficient networked control," in *Proc. of the 2018 American Control Conference (ACC)*, 2018.

[25] R. Bellman, *Dynamic Programming*. Princeton University Press, 1957.

[26] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[27] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[28] 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, p. 529, 2015.

[29] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *ICML*, 2014.

[30] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proc. of the 32nd International Conference on Machine Learning*, Lille, France, Jul 2015, pp. 1889–1897.

[31] M. Hausknecht and P. Stone, "Deep reinforcement learning in parameterized action space," *arXiv preprint arXiv:1511.04143*, 2015.

[32] R. Hafner and M. Riedmiller, "Reinforcement learning in feedback control," *Machine Learning*, vol. 84, no. 1, pp. 137–169, Jul 2011.

[33] W. Masson, P. Ranchod, and G. Konidaris, "Reinforcement learning with parameterized actions," in *AAAI*, 2016, pp. 1934–1940.

[34] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Openai baselines," https://github.com/openai/baselines, 2017.

[35] R. S. Sutton, "Td models: Modeling the world at a mixture of time scales," in *Machine Learning Proceedings 1995*. Elsevier, 1995, pp. 531–539.

[36] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[37] S. Kakade and J. Langford, "Approximately optimal approximate reinforcement learning," in *ICML*, vol. 2, 2002, pp. 267–274.

[38] R. E. Kalman *et al.*, "Contributions to the theory of optimal control," *Bol. Soc. Mat. Mexicana*, vol. 5, no. 2, pp. 102–119, 1960.

[39] D. P. Bertsekas, "Stable optimal control and semicontractive dynamic programming," *SIAM Journal on Control and Optimization*, vol. 56, no. 1, pp. 231–252, 2018.

[40] M. Donkers and W. Heemels, "Output-based event-triggered control with guaranteed $\mathcal{L}_\infty$-gain and improved and decentralized event-triggering," *IEEE Trans. Autom. Control*, vol. 57, no. 6, pp. 1362–1376, 2012.

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

[42] W. H. Heemels, M. Donkers, and A. R. Teel, "Periodic event-triggered control for linear systems," *IEEE Trans. Autom. Control*, vol. 58, no. 4, pp. 847–861, 2013.

[43] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 5026–5033.