Learning Event-triggered Control from Data through Joint Optimization

Niklas Funk [a], Dominik Baumann [c,a], Vincent Berenz [b], Sebastian Trimpe [c,a]

[a]*Intelligent Control Systems Group, Max Planck Institute for Intelligent Systems, Stuttgart, Germany*

[b]*Empirical Inference Department, Max Planck Institute for Intelligent Systems, Tübingen, Germany*

[c]*Institute for Data Science in Mechanical Engineering, RWTH Aachen University, Aachen, Germany*

Abstract

We present a framework for model-free learning of event-triggered control strategies. Event-triggered methods aim to achieve high control performance while only closing the feedback loop when needed. This enables resource savings, e.g., network bandwidth if control commands are sent via communication networks, as in networked control systems. Event-triggered controllers consist of a communication policy, determining when to communicate, and a control policy, deciding what to communicate. It is essential to jointly optimize the two policies since individual optimization does not necessarily yield the overall optimal solution. To address this need for joint optimization, we propose a novel algorithm based on hierarchical reinforcement learning. The resulting algorithm is shown to accomplish high-performance control in line with resource savings and scales seamlessly to nonlinear and high-dimensional systems. The method's applicability to real-world scenarios is demonstrated through experiments on a six degrees of freedom real-time controlled manipulator. Further, we propose an approach towards evaluating the stability of the learned neural network policies.

Key words: Event-triggered Control; Reinforcement Learning; Stability Verification; Neural Networks

1 Introduction

In modern control systems, control commands often need to be transmitted over (wired or wireless) communication networks [1,2]. Examples of such networked control systems include swarms of drones, where communication is needed for drones to fly in formation; autonomous cars, where exchanging information between vehicles may increase traffic throughput and reduce fuel consumption; or smart homes, where distributed sensors, actuators, and computing units need to cooperate to regulate the indoor climate. In all examples, multiple systems utilize the same network for communication. If all systems transmit their information at high periodic rates, this can easily overload the network and result in an increased probability of message loss and longer transmission delays [3]. Further, in many applications, distributed sensors and computing units should
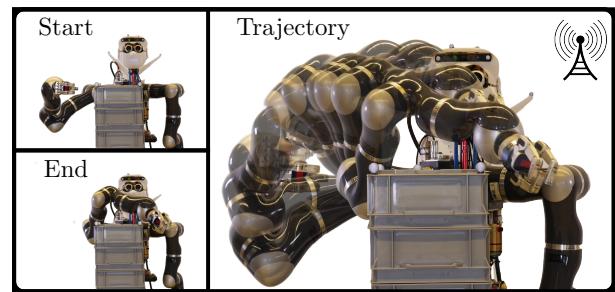

Fig. 1. Performing event-triggered control on the Apollo robot. The overlayed frames coincide with the time instants in which a new control command is computed and applied. The robot successfully avoids the obstacle with only a few recomputations of the control signal and thus saving 90 % of communication.

be untethered and thus battery-driven. In these cases, limiting communication can significantly improve battery life. Event-triggered methods have been developed explicitly to serve this need of controlling systems at reduced communication rates, see for instance [4,5,6,7] for an overview.

Email addresses: nwfunk@gmx.net (Niklas Funk),
dbaumann@tuebingen.mpg.de (Dominik Baumann),
vberenz@tuebingen.mpg.de (Vincent Berenz),
trimpe@dsme.rwth-aachen.de (Sebastian Trimpe).

In event-triggered control (ETC), closing the feedback loop and, thus, transmitting information, is triggered by the occurrence of certain events, e.g., an error growing too large. Practical investigations have shown that ETC can significantly reduce the amount of communication while still achieving high-performance control [8,9,10]. In most event-triggered approaches, the design of the control and communication strategy is based on a known mathematical model of the system [4]. Yet, for complex systems, an accurate description may not be readily available. Further, the majority of results only consider linear and low-dimensional systems. Available results for nonlinear systems mostly optimize the communication and control policies separately or fix one of both [11,12]. This is problematic as in ETC, the separation principle does not hold in general [13]. That is: even if both policies are individually optimal, their combination does not necessarily yield the overall optimal solution. For optimal ETC, the control and communication policy need to be optimized *jointly*. In this article, we propose an algorithm based on model-free reinforcement learning (RL) that jointly learns the control and communication policies from data. By exploiting model-free algorithms, we mitigate the need for an accurate dynamics model, and since we do not make any assumptions on dimensionality or linearity, the resulting framework can equally be applied to linear and nonlinear, low- and high-dimensional systems.

A key hurdle of learning ETC with RL is the hybrid action space of such controllers. At each time step, the controller takes a discrete decision, whether or not to communicate. In case of communication, a continuous control input is transmitted. Hierarchical RL [14] naturally captures this hybrid decision structure. It provides a top-level policy that decides which action to take (in our case, whether or not to communicate). Depending on this choice, the corresponding low-level policy is evaluated, which, for instance, yields the control input in case of communication. To the best of our knowledge, hierarchical RL has not yet been used for learning ETC. However, to successfully learn event-triggered controllers, it is not sufficient to apply the existing hierarchical algorithms. This is mainly because the ETC setting restricts exploration. Unlike in periodic control, in ETC, varying the control action for exploration is only possible on communication instances. Thus, exploration directly conflicts with the goal of saving communication. We extend the hierarchical algorithms accordingly, which results in the first method that successfully derives ETC strategies through joint optimization in nonlinear and high-dimensional environments.

One of the drawbacks of learning-based approaches, especially those based on neural networks (NNs), is that they often do not provide stability guarantees. However, such statements are crucial when deploying the controllers at scale in potentially safety-critical, real-world applications. We approach this challenge by proposing

a verification framework, capable of checking the stability of the learned policy and refining it if necessary. The framework combines output range analysis of the learned control policy with model knowledge. That way, we can utilize a popular NN verification framework [15] to provide stability guarantees based on control invariant sets for linear systems.

Contributions. We make the following contributions:

- Leveraging and extending hierarchical RL algorithms to obtain resource-aware, event-triggered controllers;
- Presenting an algorithm, capable of end-to-end learning of the control and communication policy through joint optimization for high-dimensional nonlinear systems;
- Demonstrating the algorithm's practical significance by applying learned ETC strategies on a real robotic system, as illustrated in Fig. 1;
- Presenting an algorithm to check the stability of linear event-triggered systems controlled by NN policies with rectified linear unit (ReLU) activations. We also provide a method for refining the NNs in case of initial instability.

Outline. We start with an overview of related work before we provide the problem formulation and necessary background. Next, we introduce the developed algorithm and present results in challenging, high-dimensional simulation environments and on a real robotic system. Lastly, we discuss the stability verification procedure and conclude with a discussion.

2 Related Work

ETC overview. For a general introduction and overview of ETC, we refer the reader to [2,4,5,7]. While [4] focuses on linear systems, the other references also discuss approaches for event-triggered control of nonlinear systems. All methods, regardless of whether they are designed for linear or nonlinear systems, rely on an accurate model of the system's dynamics. In practical applications, especially when considering complex nonlinear systems, such a model may not be available. We address the more general and more challenging problem of learning event-triggered control and communication policies without assuming any knowledge of the system dynamics.

Learning ETC – imposing structure. Recently, there have been several other works that learn event-triggered communication and control policies in a model-free way. In [11], the authors propose to use RL to learn a scheduling strategy for controlling a multi-agent system. The approach focuses on arbitrating the constrained communication bandwidth among the agents, while the agents' control strategy is fixed beforehand

and not subject to active optimization. Vamvoudakis et al. [12] exploit concepts from Q-learning, and propose a model-free algorithm that comes up with an ETC strategy for linear systems in continuous time. In their approach, the general triggering condition is predefined. Therefore, the optimization algorithm only operates on the triggering threshold of the communication strategy (i.e., when to trigger). The authors of [16] and [17] also use the same predefined triggering condition, but derive ETC strategies for nonlinear systems. While [16] uses an adaptive dynamic programming approach, [17] relies on an identifier-critic architecture, actively identifying the unknown system dynamics. The authors of [18] use NNs to parametrize the control policy, while the triggering condition depends on the state and the weights of the controller. Further, the NN is updated in an aperiodic fashion, i.e., only on triggering instances. The method proposed herein is more general. We jointly learn the control and communication strategy from scratch without imposing any additional structure, e.g., concerning the triggering condition. This is especially crucial as the separation principle does not hold in general for ETC [13]. Moreover, we showcase the performance of our algorithm in substantially more complex and higher-dimensional simulation environments than those presented in the above references, as well as on a real 6 degrees of freedom (DoF) real-time controlled manipulator.

Learning ETC – without imposing structure. There are only few other approaches [19,20] that learn ETC without a-priori restrictions on the triggering condition. Baumann et al. [19] discuss both a joint optimization of control and communication and a separate optimization procedure. However, only the separate optimization can be applied to high-dimensional tasks. The authors of [20] propose to use Gaussian processes to learn a model of an unknown system, which is then exploited to derive an optimal, self-triggered control strategy through approximated value iteration. Due to the computational complexity, this approach is limited to low-dimensional systems, and a maximum inter-communication time needs to be fixed a priori. Thus, neither of those approaches can jointly optimize the control and communication policy in high-dimensional settings, as we do herein. Further, neither work provides any stability guarantees or results on real hardware.

Deep RL for learning control. Learning control policies from data for high-dimensional and nonlinear systems has been studied extensively in recent years [21,22,23,24,25]. However, all these works consider periodic communication and are, therefore, not applicable to the problem considered herein. ETC is a challenging problem in that it leads to a hybrid action space. Approaches that deal with such hybrid action spaces will be discussed next. In [26,27], the authors propose to rephrase the problem using continuous variables following the concept of a parameterized action space Markov decision process. This approach has been used to jointly learn communication and control policies in [19]. Yet, it has only been shown to be successful in low-dimensional tasks. More recently, [28] introduced a hybrid RL algorithm that can optimize such problems without reformulation. So far, this has not been applied to ETC. Hierarchical RL frameworks [14,29,30] represent a third approach to address the hybrid problem setting. Originally, the hierarchical structure stems from the concept of temporal abstraction. However, it also naturally captures the structure of ETC. The discrete high-level decision on which sub-policy to execute next coincides with the communication decision in ETC, while the sub-policy yields the continuous control command. We are not aware of any other work that extends hierarchical RL algorithms to make them applicable for ETC, as we do herein.

Stability analysis. Due to the nonlinear activation functions and many parameters involved, it is usually difficult to provide stability guarantees for learned policies parametrized by NNs. Thus, this problem is typically not addressed. Two recent exceptions related to the approach herein are [31] and [32]. In [31], the authors guarantee input to state stability of long short term memory NNs. This can effectively be exploited when using NNs for modeling or system identification. Its use for controller design has not yet been demonstrated. Karg et al. [32] examine the stability of NN controllers via output range analysis. This way, they can model the closed-loop behavior of the controlled system and define requirements for asymptotic stability. To achieve this property, they propose to refine the final layer of the NN, based on a predefined linear quadratic regulator policy. Our approach is more general in that it refines the entire network instead of only the final layer and is not restricted to a particular type of controllers. Further, compared to both approaches, we provide results for larger networks and present an approach for checking the stability of ETC policies, which is generally more challenging due to the sporadic updates, triggered by the communication policy.

3 Problem Formulation

System. We consider a dynamic system whose state is monitored by sensors, and that is connected to a learning agent (cf. Fig. 2). While the sensor measurements are directly available to the learning agent, control commands need to be transmitted over a communication network to the actuators. If not stated differently, we assume the dynamics of the system to be unknown and of the form

$$x[k+1] = f(x[k], u[k], v[k]) , \qquad (1)$$

where $x[k] \in \mathbb{R}^n$ denotes the state, $u[k] \in \mathbb{R}^m$ the input, $v[k] \in \mathbb{R}^n$ process noise, and $k \in \mathbb{N}$ the discrete-time
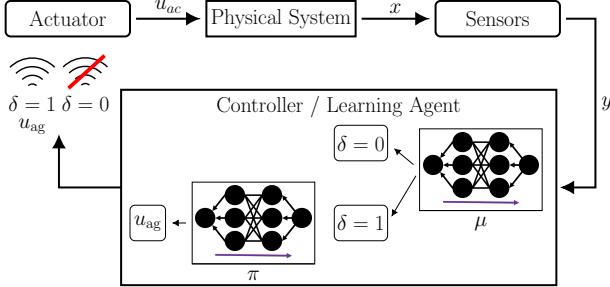
Fig. 2. Schematic of the learning ETC setting addressed in this article. An actuator is acting on a physical system. The sensory information about the state of the system is delivered to the learning agent. The agent then has the choice to either communicate a control command to the actuator ($\delta = 1$) or to skip the current communication slot to save resources ($\delta = 0$). The communication decision is retrieved from NN μ. In case of communication, the control action is computed via NN π. During training time, the weights of the NNs are refined via backpropagation.

index. The measurements $y[k] \in \mathbb{R}^p$ are assumed to be given by

$$y[k] = g(x[k], w[k]) \,, \tag{2}$$

with $w[k] \in \mathbb{R}^p$ representing measurement noise. For ease of presentation and consistency with existing literature, we assume $g(x, w) = x$ for the derivation of the algorithm in Secs. 4 and 5. However, in the experimental evaluations in Secs. 6 and 7, we demonstrate its applicability in settings with more complex measurement functions.

Event-triggered control. Unlike in periodic control, in ETC the feedback loop is closed adaptively. Based on the measurements, the agent decides whether ($\delta[k] = 1$) or not ($\delta[k] = 0$) to communicate with the actuator - i.e., whether to exploit the limited resources or to skip the current communication slot. As is typically done in ETC [4], we assume zero-order hold (ZOH) at the actuator in between communication events. Mathematically, this can be expressed as

$$u_{\mathrm{ac}}[k] = \begin{cases} u_{\mathrm{ag}}[k], & \text{if } \delta[k] = 1 \\ u_{\mathrm{ac}}[k-1], & \text{if } \delta[k] = 0 \,, \end{cases} \tag{3}$$

where u_{ac} denotes the control action applied at the actuator and u_{ag} represents the action calculated by the agent (cf. Fig. 2).

Performance objective. The proposed learning algorithm should result in both a control and communication strategy that ensure that the physical system behaves in the desired way (e.g., reaching a desired setpoint) while the resource constraints are taken into account. Both ob-

jectives are combined in the following reward function

$$\begin{aligned} R &= \sum_{k=0}^{N} \gamma^k (R_{\mathrm{ctrl}}[k] + R_{\mathrm{comm}}[k]) \\ &= \sum_{k=0}^{N} \gamma^k (R_{\mathrm{ctrl}}[k] - \lambda\delta[k]) \,, \end{aligned} \tag{4}$$

where λ is the weight on penalizing communication, γ the discount factor, and $R_{\mathrm{ctrl}}[k]$ the control reward. If the discount factor γ is less than 1, it limits the horizon up to which future rewards are reflected in the value of the function. We express the communication savings Γ as the percentage of the available communication slots that have not been used, i.e.,

$$\Gamma = 1 - \Big(\sum_{k=1}^{N} \delta[k]\Big)/N \tag{5}$$

for a trajectory of length N. That is, $\Gamma = 0\,\%$ means all communication slots have been used, and $\Gamma = 100\,\%$ corresponds to no communication.

Problem statement. The goal of this paper is to devise a learning algorithm that maximizes (4). To address the drawbacks of existing methods, the algorithm shall meet the following requirements: *(i)* it should be model-free to avoid the need for an accurate dynamics model; *(ii)* it should jointly optimize the control and communication policy since the separation principle does not hold in general for ETC; *(iii)* it should be applicable to linear and nonlinear, *(iv)* low- and high-dimensional systems.

Towards stability. In addition to learning ETC, which is the focus of this work, we are also concerned with providing stability guarantees for the policies. This is essential for the practicability of the method, especially considering its potential use in safety critical environments. However, this is particularly difficult when dealing with learned, highly parametric, and nonlinear NN control strategies in the ETC setting. We approach this challenge by defining system stability through control invariant sets, assuming known, linear dynamics, and using NNs with ReLU activations. This allows us to devise a second algorithm capable of verifying and refining the NN policies to guarantee system stability.

4 Background: Hierarchical Reinforcement Learning

ETC represents a hybrid control problem, which is difficult to solve for most standard RL algorithms. In the following, we introduce the options framework established by Precup et al. [14,30]. This hierarchical RL framework

is particularly suitable for learning control policies in hybrid action spaces as it naturally splits the discrete and continuous variables.

To represent the hierarchy, the options framework requires one additional variable, called option $o \in \mathcal{O}$, besides the state $x \in \mathcal{X}$ and the control action $u \in \mathcal{U}$. $\mathcal{O}$, $\mathcal{X}$ and $\mathcal{U}$ represent the set of options, the state, and the action space. To keep notation uncluttered, we omit time dependence in the next two sections and write $x = x[k]$, $x' = x[k+1]$, and similar for other variables. The framework is parametrized by three policies: the policy over options $\mu(o|x)$, the intra option policy $\pi_o(u|x)$, and the termination function $\beta_o(x)$. The principle in few words: the policy over options decides which option is to be executed. Depending on this choice, the action is sampled from the corresponding intra option policy until the termination function indicates to stop the execution of the current option, which triggers a restart of the procedure. Mathematically, the policy over options $\mu(o|x)$ determines the probability of choosing option o, the intra option policy $\pi_o(u|x)$ determines the distribution over actions u, and the termination function $\beta_o(x)$ determines the probability for terminating the execution of option o. As shown in [29], this can be rephrased as having a flat action space without the hierarchy,

$$\pi(u|x,o) = (1-\beta_o(x))\pi_o(u|x)+\beta_o(x)\sum_{\tilde{o}\in\mathcal{O}}\mu(\tilde{o}|x)\pi_{\tilde{o}}(u|x) \ . \tag{6}$$

After executing action u, the next state and option are given by x' and o'.

To optimize the agent's behavior, it is essential to estimate the expected reward of its actions using the Q-function. This estimate directly indicates which actions are more rewarding. In standard RL, the Q-function assigns a value to a state-action pair (x, u). In the hierarchical setting, we define it as assigning a value to a state-option pair (x, o),

$$Q(x,o) = \int_{\tilde{u}\in\mathcal{U}} \pi_o(\tilde{u}|x)\hat{Q}(x,o,\tilde{u}) \ \mathrm{d}\tilde{u} \ , \tag{7}$$

where $\hat{Q}$ can be computed via $\hat{Q}(x,o,u) = r(x,o,u) + \gamma Q(x',o')$ and r defines the single step reward. In conclusion, the Q-function describes the future reward to be expected when starting from state x and option o. $\hat{Q}$ the reward starting from x, o and action u.

The options framework aims at maximizing the expected reward, which for a given state x and option o reads

$$R(x,o) = [1 - \beta_o(x)]Q(x,o)$$
$$+ \beta_o(x)\sum_{\tilde{o}\in\mathcal{O}}\mu(\tilde{o}|x)Q(x,\tilde{o}) \ . \tag{8}$$

Assuming the termination function β is parametrized by parameters θ_β, the policy over options μ by θ_μ, and the intra option policy π by θ_π, the gradient of the reward (8) with respect to the policies is given by

$$\frac{\partial R(x,o,\theta_\beta,\theta_\mu)}{\partial \theta_\mu}$$
$$= \beta_o(x,\theta_\beta)\mathbb{E}[\frac{\partial}{\partial \theta_\mu}\log(\mu(o,\theta_\mu|x))Q(x,o)] \tag{9}$$

and

$$\frac{\partial R(x,o,\theta_\beta,\theta_\mu)}{\partial \theta_\beta} =$$
$$\frac{\partial \beta_o(x,\theta_\beta)}{\partial \theta_\beta}(\sum_{\tilde{o}\in\mathcal{O}}\mu(\tilde{o},\theta_\mu|x)Q(x,\tilde{o}) - Q(x,o)) \ . \tag{10}$$

For each intra option policy, we seek to maximize

$$Q(x,o) = \int_{\tilde{u}\in\mathcal{U}} \pi_o(\tilde{u},\theta_\pi|x)\hat{Q}(x,o,\tilde{u}) \ \mathrm{d}\tilde{u} \ . \tag{11}$$

Taking the gradient results in

$$\frac{\partial Q(x,o,\theta_\pi)}{\partial \theta_\pi} = \mathbb{E}[\frac{\partial}{\partial \theta_\pi}\log(\pi_o(u,\theta_\pi|x))\hat{Q}(x,o,u)] \ . \tag{12}$$

For continuous action spaces, [33] proposes to use proximal policy optimization (PPO) [34] for updating the intra option policy. PPO stabilizes the learning process for continuous action tasks by limiting the covariate shift through including a clipping function into the optimization objective L. Let θ denote the parameters to be optimized, and π' denote the new policy to be found ($\pi' = \pi(\theta)$), whereas π denotes the policy under the old parameters ($\pi = \pi(\theta_{\text{old}})$) which were used for sampling the state action transitions. The goal of the PPO algorithm is to optimize

$$L(\pi') = R(\pi') - R(\pi)$$
$$= r(x,o,u^{\pi'}) + \gamma Q^\pi(x',o') - Q^\pi(x,o) \tag{13}$$
$$= A^\pi(x,o,u^{\pi'}) \approx \frac{\pi'_o(u|x)}{\pi_o(u|x)}A^\pi(x,o,u^\pi) \ .$$

Following the derivations presented in [33], we have

$$\frac{\partial L(\theta)}{\partial \theta} = \mathbb{E}[\frac{\partial}{\partial \theta}\min[\frac{\pi'_o(u|x)}{\pi_o(u|x)}A^\pi(x,o,u^\pi),$$
$$\mathrm{clip}(\frac{\pi'_o(u|x)}{\pi_o(u|x)}, 1 - \epsilon, 1 + \epsilon)A^\pi(x,o,u^\pi)]] \tag{14}$$

for the update of the intra option policy, with the advantage $A^\pi(x,o,u^\pi) = r(x,o,u^\pi) + \gamma Q^\pi(x',o') - Q^\pi(x,o)$,

and the clipping function

$$\text{clip}(a, b, c) = \begin{cases} a, & \text{if } a \in [b, c] \\ b, & \text{if } a < b \\ c, & \text{if } a > c \, , \end{cases} \quad (15)$$

assuming $c \geq b$, and ϵ the range in which no clipping is applied.

Combining (9), (10), and (14) leads to the proximal policy option-critic (PPOC) framework for continuous action spaces, as presented in [33]. Further, [33] uses generalized advantage estimation (GAE) [35] to calculate $A^\pi(x, o, u^\pi)$. The GAE algorithm allows to tune the bias-variance tradeoff when estimating this advantage term.

5 Learning Event-triggered Control and Communication Policies

We now establish the link between the previously introduced hierarchical RL algorithm and the ETC problem formulation and present an algorithm capable of learning ETC strategies from data through joint optimization. In Sec. 5.1 we bridge the gap between ETC and hierarchical RL and show in which ways the problems they solve are related. In Sec. 5.2 we first explain why simply applying plain hierarchical RL is not enough to solve ETC problems before we show how existing algorithms and concepts need to be extended to be applicable.

5.1 Relating Hierarchical Reinforcement Learning to Event-triggered Control

The hierarchical RL algorithm presented in the preceding section naturally allows us to represent problems with hybrid action spaces. The policy over options μ performs a discrete decision (which option to execute next), while the intra option policy π_o returns a continuous action, if needed. Thus, in the context of ETC, the policy over options represents the triggering law, while the intra option policy yields the control action in case of communication. We always assume that option 0 (o_0) corresponds to no communication and to performing the ZOH, while option 1 (o_1) corresponds to sampling a continuous action from a NN policy, i.e., $u_{\text{ag}}[k] = \pi_{o_1}(u|x)$ (cf. (3)). Thus, the event-triggered policy saves not only communication, but also computational resources since the intra option policy does not need to be evaluated in case of no communication. Using this definition of options, (3) can be rewritten as

$$u_{\text{ac}}[k] = \begin{cases} u_{\text{ac}}[k-1], & \text{if } \delta[k] = 0, \ o = o_0 \\ u_{\text{ag}}[k] = \pi_{o_1}(u|x), & \text{if } \delta[k] = 1, \ o = o_1 \, . \end{cases} \quad (16)$$

5.2 Hierarchical Reinforcement Learning for Event-triggered Control

Applying the concepts introduced in Sec. 4 is not sufficient for successfully learning ETC strategies. This is due to the special nature of the ETC problem. When using the algorithm in a standard, periodic control setting, both options coincide to sampling a continuous action from the respective intra option policy and applying it to the system. On the contrary, in ETC, choosing option 0, i.e., the ZOH, directly fixes the action and, thus, introduces a great difference between the capabilities of the options. Simply put, in ETC, the two options cannot compensate for each other as only option 1 is capable of changing the action applied to the system. To account for this difference, we next propose several modifications to the original algorithm, mainly focused on stabilizing the learning process and increasing exploration on the options level.

Due to the different capabilities of the two options in ETC, learning the policy over options is very sensitive. However, in the original PPOC framework, the authors propose to only use PPO for updating the intra option policy, while the policy over options is refined using vanilla policy gradient (9). In contrast, we propose to also use the PPO algorithm for updating the policy over options for two reasons. First, the learning process is more sensitive due to the limited action of the ZOH in case of no communication. Second, as the two policies influence and affect each other, as can be seen in (6), restricting the update process for both of the policies enhances the overall performance and stabilizes the learning process. Let θ_μ denote the parameters of the policy over options, subject to optimization, and $\mu' = \mu(\theta_\mu)$ the new policy, while $\mu = \mu(\theta_{\mu_{old}})$ represents the old one. Using PPO to update the policy over options results in

$$\begin{aligned} L(\theta_\mu) &= R(\mu') - R(\mu) = Q(x, o(\mu')) - Q(x, o(\mu)) \\ &= r(x, o(\mu')) + \gamma V^\mu(x') - V^\mu(x) \\ &= A^\mu(x, u^{\mu'}) \approx \frac{\mu'(o|x)}{\mu(o|x)} A^\mu(x, o^\mu) \, , \end{aligned} \quad (17)$$

where V is the value function and defined as $V(x) = \sum_{\tilde{o} \in \mathcal{O}} \mu(\tilde{o}|x) Q(x, \tilde{o})$. Thus, exploiting the PPO algorithm, the gradient is given by

$$\begin{aligned} \frac{\partial L(\theta_\mu)}{\partial \theta_\mu} = E[\frac{\partial}{\partial \theta_\mu} \min[\frac{\mu'(o|x)}{\mu(o|x)} A^\mu(x, o^\mu), \\ \text{clip}(\frac{\mu'(o|x)}{\mu(o|x)}, 1 - \epsilon, 1 + \epsilon) A^\mu(x, o^\mu)]] \, . \end{aligned} \quad (18)$$

For calculating the advantage function in this case, there are two possibilities:

(1) $A^\mu(x, o^\mu) = Q(x, o) - \max_{\tilde{o} \in \mathcal{O}} Q(x, \tilde{o})$;

(2) $A^\mu(x, o^\mu) = Q(x, o) - \sum_{\tilde{o} \in \mathcal{O}} \mu_{\text{old}}(\tilde{o}|x) Q(x, \tilde{o})$.

The first possibility represents a greedy approach as the baseline is the maximum Q-value possible, whereas the second one represents the expected Q-value under the old parameters. In preliminary experiments, the first version performed better and is, therefore, used herein.

The discrepancy in the capabilities of the options also affects the exploration process. While any continuous control input can be applied in the case of communication (i.e., option 1), option 0 is extremely limited. The algorithm might tend to only choose option 1 as, especially at the beginning of the learning process, the ZOH will likely result in no improvements of the reward. Hence, we introduce an entropy term in the optimization algorithm. The entropy scheduling ensures that there is enough exploration for the policy over options, which handles the communication decision. The gradient for the policy over options, therefore, equates to

$$
\frac{\partial L(\theta_{\mu'})}{\partial \theta_{\mu'}} = \frac{\partial}{\partial \theta_{\mu'}} \mathbb{E}[\min[\frac{\mu'(o|x)}{\mu(o|x)} A^\mu(x, o), \text{clip}(\frac{\mu'(o|x)}{\mu(o|x)},
$$
$$
1 - \epsilon, 1 + \epsilon) A^\mu(x, o)] + \tau \log(\mu'(o|x)) \mu'(o|x)],
$$

$$(19)$$

where τ represents the entropy regularization coefficient. Over time, the entropy regularization is reduced as enough exploration has been conducted and, therefore, a rather exploitative behavior is preferred.

Considering the optimization objectives of the original PPOC algorithm reveals that the gradients of the policy over options, as well as the termination function, effectively optimize the same objective (see (9), (10)). The policy over options is supposed to choose the option with the highest Q-value, and the termination function should terminate the current option when another option has a higher estimated Q-value. As this is the same goal, we remove the termination function (i.e., $\beta(x) \equiv 1$). In this new setting, one can also interpret termination as the policy over options deciding to choose another option. This modification simplifies the learning process, as only two policies have to be optimized. It further clearly establishes the link to the ETC problem formulation. The discrete decision of the policy over options coincides with the triggering law, while the intra option policy represents the control law.

The resulting algorithm for learning the hierarchical control policy is presented in Alg. 1. We use NN policies to represent the policy over options, the intra option policy, as well as to approximate the Q-function. Details on the network structures are provided in App. A.

Due to the ZOH, we have to include the last control action applied to the system in the state to keep the properties of a Markov decision process. Therefore, we define $\tilde{x}[k] = (x[k], u_{\text{ac}}[k-1])^{\text{T}}$.

Algorithm 1 Hierarchical RL for ETC

1: Initialize Clipping ϵ and Entropy Regularization τ
2: Inititalize Q-Network $Q(x, o)$
3: Initialize Policy over options network $\mu(o|x)$
4: Initialize Intra option policy network $\pi_o(u|x)$
5: **for** number of epochs **do**
6:     $Q' \leftarrow Q$
7:     $\mu' \leftarrow \mu$
8:     $\pi'_o \leftarrow \pi_o$
9:     Sample (x, o, u)-Transitions using current μ, π
10:    Use GAE ([35]) to calculate $A^\pi(x, o, u)$
11:    **for** number of optimizer iterations **do**
12:        **for** number of options, $o = 0, 1, 2, ...$ **do**
13:            Sample batch
14:            $A^\mu(x, o) = Q'(x, o) - \max_{\tilde{o}} Q'(x, \tilde{o})$
15:            $L_1(\theta_{\mu'}) = \mathbb{E}[\min[\frac{\mu'(o|x)}{\mu(o|x)} A^\mu(x, o), \text{clip}(\frac{\mu'(o|x)}{\mu(o|x)},$
                  $1 - \epsilon, 1 + \epsilon) A^\mu(x, o)] + \tau \log(\mu'(o|x)) \mu'(o|x)]$
16:            $L_2(\theta_{\pi'}) = \mathbb{E}[\min[\frac{\pi'_o(u|x)}{\pi_o(u|x)} A^\pi(x, o, u), \text{clip}(\frac{\pi'_o(u|x)}{\pi_o(u|x)},$
                  $1 - \epsilon, 1 + \epsilon) A^\pi(x, o, u)]]$
17:            $L_3(\theta_{Q'}) = \mathbb{E}[(Q' - (Q(x, o) + A^\pi(x, o, u)))^2]$
18:            $\theta_{\mu'} \leftarrow \theta_{\mu'} + \alpha_{\theta_\mu} \frac{\partial L_1(\theta_{\mu'})}{\partial \theta_{\mu'}}$
19:            $\theta_{\pi'} \leftarrow \theta_{\pi'} + \alpha_{\theta_{\pi'}} \frac{\partial L_2(\theta_{\pi'})}{\partial \theta_{\pi'}}$
20:            $\theta_{Q'} \leftarrow \theta_{Q'} - \alpha_{\theta_{Q'}} \frac{\partial L_3(\theta_{Q'})}{\partial \theta_{Q'}}$
21: /* One potential implementation of the entropy scheduling is shown below
22:        **if** epoch % 1000==0 **then**
23:            $\tau = \tau/10$

---

6   Results in Simulation Environments

To showcase the versatility of the presented algorithm, we apply it to low-dimensional and linear as well as high-dimensional and nonlinear systems. We first present results for the OpenAI Gym [36] Pendulum environment. This simple environment also allows us to compare the algorithm's performance to classical ETC approaches. Next, we show its behavior in challenging nonlinear and high-dimensional MuJoCo [37] environments.

For all simulation experiments, we use two options as shown in (16). The reward is given by (4), where, if not stated differently, $R_{\text{ctrl}}[k]$ is the unmodified reward provided by the respective environment. For all the experiments, we use the environments' original sampling rate of 20 Hz and their unmodified measurement functions. Thus, the state available to the learning agent is given by $\tilde{x}[k] = (g(x[k], w[k]), u_{\text{ac}}[k-1])^{\text{T}}$.

We train all of the agents for 5000 epochs and store the model that achieves the highest reward. Each epoch consists of sampling 2048 (x, o, u)-transitions. For each communication penalty λ, we start 10 training runs with different seeds and report the performance of the best models. The NN architectures are presented in App. A.

Training one model took around 30 to 40 hours on a single CPU. However, we did not parallelize the training process, which the algorithm would allow for. Thus, we expect that the training time can be reduced significantly. On a laptop with an Intel® Core™ i7-7700HQ CPU @ 2.80GHz and 24GB RAM, the evaluation of both of the policies (policy over options and intra option policy) takes on average 1.1 ms. The code that has been used to train the learning-based models and videos illustrating the results are available at ².

6.1 Pendulum Environment

In this rather simple environment, we consider the challenge of stabilizing the inverted pendulum on top, already starting in an upright position. We use the reward function's original parameters, except for increasing the penalization on the control input from 0.001 to 0.1 to prevent the controller from being too aggressive.

For this task, it is straightforward to linearize the system dynamics around the equilibrium. This allows us to compare the results of the presented algorithm to other well-known ETC approaches. In particular, we compare our algorithm to

- **LQR**:
  $u[k] = Kx[k] \Rightarrow \delta[k] = 1 \; \forall k$,
- **LQR random skip**:
  $u[k] = \begin{cases} Kx[k] \Rightarrow \delta[k] = 1, & \text{if } \nu > \xi \\ u[k-1] \Rightarrow \delta[k] = 0, & \text{otherwise} \end{cases}$,
- **state triggering 2 norm**:
  $u[k] = \begin{cases} Kx[k] \Rightarrow \delta[k] = 1, & \text{if } \|x[k]\|_2 > \xi \\ u[k-1] \Rightarrow \delta[k] = 0, & \text{otherwise} \end{cases}$,
- **output based triggering** [38]:
  $u[k] = \begin{cases} Kx[k] \Rightarrow \delta[k] = 1, & \text{if } \|K\hat{x}[k] - Kx[k]\|_2 > \xi\|Kx[k]\|_2 \\ u[k-1] \Rightarrow \delta[k] = 0, & \text{otherwise} \end{cases}$,
- **state diff triggering** [39]:
  $u[k] = \begin{cases} Kx[k] \Rightarrow \delta[k] = 1, & \text{if } \|\hat{x}[k] - x[k]\|_2 > \xi\|x[k]\|_2 \\ u[k-1] \Rightarrow \delta[k] = 0, & \text{otherwise} \end{cases}$,

where $\hat{x}[k]$ represents the state at the last triggering instance and ξ a threshold variable, adjusting the triggering condition. The random variable ν is uniformly sampled from the interval $[0, 1]$. The gain matrix K is chosen from an LQR design where the weights are identical to the parameters of the reward function.

In Fig. 3, we show the performance of the introduced event-triggering laws and our data-based RL algorithm. As can be seen, for the classical approaches, only communication savings up to about 80 % are possible, whereas our algorithm finds policies that can save up to 90 %. However, for intermediate communication savings, the classical methods usually outperform our learning-based
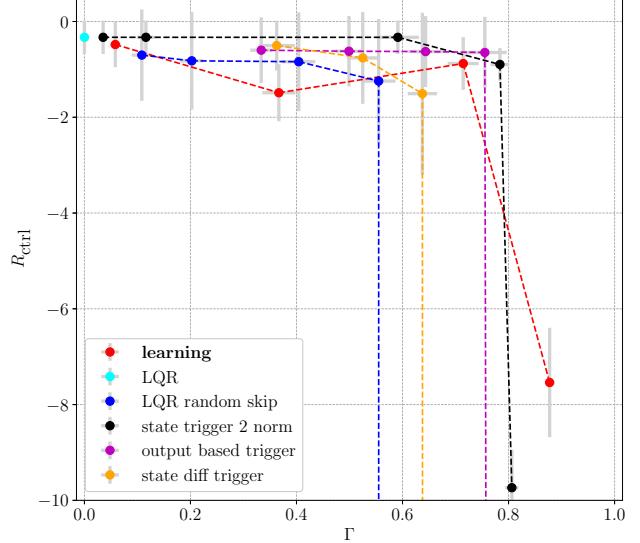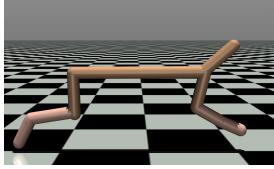
---

² https://sites.google.com/view/learn-event-triggered-control

Fig. 3. Illustration of the magnitude of the control reward R_{ctrl} versus the communication savings Γ for various ETC strategies and our learning approach for the task of stabilizing an inverted pendulum. The mean and standard deviation are obtained by performing 10 rollouts with each of the policies, and indicated by the grey lines. The vertical drops indicate when the policies become unstable, i.e., fail to stabilize the pendulum. For intermediate communication savings, all the approaches show similar performance. Considering the maximum savings possible, the learning approach outperforms the others.

approach. This might be caused by the algorithm getting stuck in local optima. Nevertheless, the rewards of the classical approaches and our algorithm are still in the same order of magnitude for the intermediate savings. This distinguishes the herein presented algorithm from the RL framework proposed in [19], which also achieved communication savings of up to 90 % in the pendulum environment, but was outperformed by orders of magnitudes for intermediate communication savings.
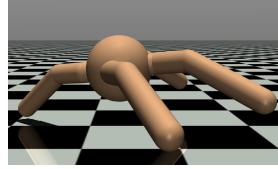
6.2 Results in high-dimensional, nonlinear Environments

We now focus on the nonlinear, high-dimensional MuJoCo Half-Cheetah (Fig. 4a) and Ant (Fig. 4b) environment, that cannot simply be linearized around certain equilibria points to yield linear system dynamics. In those challenging nonlinear environments, known approaches for ETC usually fail as the settings are too complex. To our knowledge, only [19] presents results in such environments. However, through separately learning control and communication.

In these environments, the reward R_{ctrl} is mainly made up of the distance covered, but additionally includes a cost on the control action and contact forces. The goal of the optimization is to move the robotic agent as far

(a) Half-Cheetah environment. Its observation space has 18 dimensions, and the action space 6.

(b) Ant environment. Its observation space has 111 dimensions, and the action space 8.

Fig. 4. The high-dimensional MuJoCo environments in which we train the learning agents.

as possible in the available time, given input and environmental constraints. Apart from the reward, we will also analyze the distance d that the robotic agents cover during a rollout. We slightly modify the Ant environment by eliminating the restriction that limits the jump height of the center of mass of the Ant.

6.2.1 Half-Cheetah Environment

The red graph in Fig. 5 illustrates the performance of policies trained using the presented algorithm. By increasing the value of λ, higher communication savings Γ can be achieved while, on the other hand, the covered distance decreases. In this Half-Cheetah environment, we are capable of learning policies that can achieve up to 80 % communication savings. In the figure, we showcase the performance of the nine best, out of ten rollouts with each policy. This is due to the fact that some of the policies exhibit one rollout where the Cheetah is flipped onto its back, resulting in a significantly decreased distance covered.

Interestingly, the communication savings have an impact on the gait, as shown in the accompanying video. More communication savings result in a rather jumpy policy; when the Cheetah is in the air, there is no need to communicate. One can see that for a low penalty on communication, the Cheetah's feet stay rather close to the ground, which also results in slightly faster progress. As shown in the red graph of Fig. 5, the standard deviation of the distance traveled is small, indicating that all the policies are robust and reliable. The training stability, i.e., the percentage of training runs that result in moving the Cheetah forward, is dependent on the communication penalization λ. The percentage of successful training runs typically decreases as the value of λ is increased, as the option of never communicating becomes more and more attractive. Therefore, for high values of λ, the training runs naturally converge to policies that do not communicate at all. This results in the Cheetah standing still at its initial position.

We also compare our algorithm's results with a baseline policy trained using the PPO algorithm. This baseline PPO policy only learns the control policy, while the communication strategy is fixed to always communicate. One possibility to achieve communication savings using this baseline policy is to randomly skip communication with a predefined probability. As can be seen in the blue graph in Fig. 5 and the corresponding video, this rapidly decreases the performance. This emphasizes that it is crucial to optimize the control and communication strategy jointly. The models trained using our algorithm outperform the baselines. They are considerably more resource-efficient while the Cheetah still covers at least the same distance.

As an additional benchmark, we implemented a modified version of the proposed algorithm that optimizes the control and communication strategy separately, in an alternating fashion. The performance of these agents is visualized through the black and purple graph in Fig. 5. To obtain the agents corresponding to the black and purple line, we switch between solely optimizing control and solely optimizing communication, every 25 respectively 100 epochs. As can be seen, the proposed joint optimization approach outperforms the separate optimization method. Considering low communication savings, the difference in performance is small. However, for high communication savings, it is significant. This highlights that especially for more difficult tasks, where it is crucial to finely adjust the two policies to each other, the joint optimization approach is superior. The experiments also illustrate that a higher frequency of alternation (black graph) results in better performance. Further increasing the frequency of alternation naturally converges to the joint optimization approach.

We additionally investigate the significance of the modifications detailed in Sec. 5.2. For this, we try to learn an event-triggered controller using the original PPOC implementation (see Sec. 4). As shown in Fig. 5, for $\lambda = 0.0$, the PPOC algorithm finds a solution that saves a minimal amount of communication. However, for higher values of λ, e.g., 1.0, the PPOC algorithm always results in a policy that never communicates, and therefore, the Cheetah does not progress at all. Thus, unlike our algorithm, the original PPOC algorithm is incapable of arriving at event-triggered controllers that reduce communication significantly, while still moving the Cheetah forward. This is probably due to the greedy optimization of the PPOC algorithm. Further insights are provided in App. B.

6.2.2 Ant Environment

When deploying the algorithm to the even higher-dimensional Ant environment, we obtain similar results. Fig. 6 shows the distance covered by the Ant versus the communication savings Γ for models trained using the proposed algorithm. We again show the performance of the nine best, out of ten rollouts for each policy since some policies exhibit one rollout where the Ant
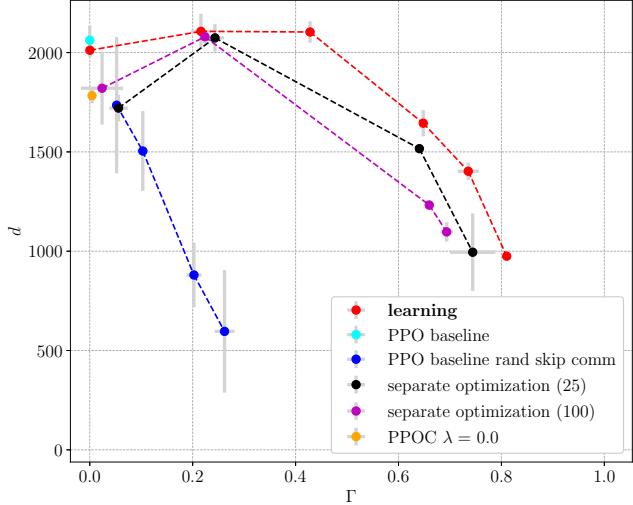
Fig. 5. Illustration of the performance of various agents. Depicted is the distance traveled by the Half-Cheetah d versus the communication savings Γ. The shown mean and standard deviation are calculated from the nine best, out of ten rollouts, and illustrated in grey. The red line connects the results obtained from agents, trained using our algorithm. The blue line combines the rollouts of a PPO baseline policy, which saves communication by randomly skipping communication. The purple and black lines connect the results obtained from agents, trained using a modified version of our proposed algorithm. I.e., control and communication are optimized separately, in an alternating fashion. For the black line, the optimization is switched every 25 epochs; for the purple one, every 100 epochs. The herein presented joint optimization approach outperforms the others in terms of performance at intermediate communication savings as well as maximum savings possible.

is flipped onto its back, resulting in a significantly decreased distance covered. As can be seen from the plot, the most resource-aware control policies are capable of saving up to 70 % of communication. The figure also illustrates that performance degrades if the communication savings increase. The policy saving 70 % exhibits the largest standard deviation. Therefore, it is the least stable, but also the most resource-efficient policy. The corresponding video from rollouts with different penalizations underlines those findings and aligns with the results for the Half-Cheetah. When more communication is possible, the Ant's feet are kept rather close to the ground, which results in fast progress, and more reliability as the chances of flipping are minimized. When the penalization on communication is progressively increased, the gait changes towards a rather jumpy behavior. This allows for the most significant communication savings, as when the feet are in the air, no communication is needed. Nevertheless, compared to the results for the Cheetah, the changes in the gait behavior are less obvious. Considering the training process, again, if the resource constraints become very restrictive, the learning process becomes more difficult, and never communicating also becomes a local optimum.
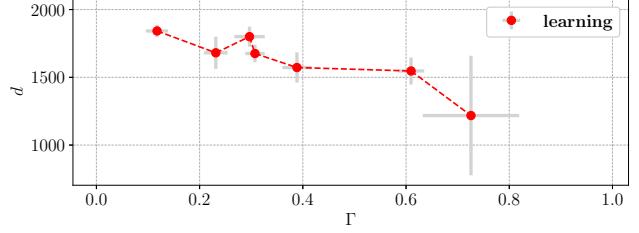

Fig. 6. Illustration of the agents' performance, trained using our proposed algorithm. Depicted is the distance covered by the Ant d versus the communication savings. The shown mean and standard deviation are calculated from the nine best, out of ten rollouts, and illustrated in grey. At maximum, 70 % of communication can be saved.

Compared to [19], where the authors also try to learn an ETC strategy from data for the Ant environment, the proposed joint optimization achieves even higher communication savings with up to 70 %, whereas [19] only reported savings of up to 60 % for their separate optimization approach.

7 Results on Hardware

While many deep RL methods have shown good performance in simulation environments, there are only few examples where the learned policies are actually deployed on real hardware. This is mainly due to the sample inefficiency of those algorithms or insufficient simulation to reality transfer. In this section, we present the results of using our learning algorithm on real hardware, namely, the Apollo robot shown in Fig. 7.

In contrast to the previous simulation experiments, which provide perfect communication and no delays, we now consider real experimental conditions with noise, delays, and potential packet losses. The experimental setup is as follows: the sensory information, i.e., Apollo's joint configuration, is communicated to the learning agent running on a computer. The agent then decides whether or not to communicate, and, depending on that decision, eventually sends new control commands to the robot. We use a base sampling time of 20 Hz. Further details on the setup can be found in App. C.

7.1 Problem Definition and Setup

To demonstrate the feasibility of the introduced framework, we employ a real-time position controller for the end-effector of Apollo's right arm with 6 DoF. The controller that is to be learned operates in the cartesian space. The goal is to reach a desired position as accurately as possible under resource constraints, i.e., trying to communicate as efficiently as possible. The learning agent's output is a desired reference velocity in the task space of the end-effector, $u[k] = v_{\text{ref}}[k] = (v_x[k],\ v_y[k],\ v_z[k])^{\text{T}}$.

As the input to the robot has to be with respect to the individual joints, while the learning algorithm's output is defined in task space, the procedure is as follows. The learning algorithm outputs velocities in task space, which are then mapped to the corresponding commands in joint space through inverse kinematics. We use the Pinocchio library [40] for this step. The process of computing the target velocity in joint space is iterative and affected by the current robot configuration. It is not guaranteed that the desired velocity can be reached accurately. Once the velocities in joint space are computed, they are applied until the next instance of communication. In the beginning, the desired velocity is reached quite accurately. Considering a longer horizon, it is obvious that constant joint velocities do not coincide with a constant velocity in task space. Therefore, over time, the discrepancy between the desired and the actual velocity increases as there is no recomputation until the next instance of communication.

Since the learning agent only operates in the task space, the state which is fed to the agent is $\tilde{x}[k] = (g(x[k], w[k]), \ u_{ac}[k-1])^\mathrm{T} = (\dot{x}_{ef}[k], \ x_{ef}[k], \ x_{ref}[k] - x_{ef}[k], \ u_{ac}[k-1])^\mathrm{T}$, where x_{ef} denotes end-effector position and x_{ref} denotes the desired reference position that is to be reached, in cartesian coordinates. Thus, no joint information is available to the learning agent, which makes the setting partially observable.

In the settings in Sec. 6.2, the steady-state solution is applying a sequence of control inputs to achieve a constant movement of the agent. However, for this task, once the end-effector is close to the reference, it is desirable to apply a zero action to hold its position. As it is unlikely that a NN policy outputs exactly zero, we exploit our approach's hierarchical nature and simply define a third option (o_2) that corresponds to setting the control input to zero. Hence, for the hardware experiments, unlike presented in (16), we consider three options:

$$u_{ac}[k] = \begin{cases} u_{ac}[k-1], & \text{if } \delta[k]=0, \ o=o_0 \\ u_{ag}[k] = \pi_{o_1}(u|x), & \text{if } \delta[k]=1, \ o=o_1 \\ u_{ag}[k] = \begin{pmatrix} 0 & 0 & 0 \end{pmatrix}^\mathrm{T}, & \text{if } \delta[k]=1, \ o=o_2 \ . \end{cases}$$
(20)

To obtain the results presented in the next sections, we first train the learning algorithm in simulation using the simulation laboratory (SL) framework [41]. Then, we deploy the learned policies on the real robot. For the transfer from simulation to reality, no additional adjustments are performed.

7.2 Dynamic Reference Position Tracking

At first, we describe the performance of resource-aware agents trained for reaching a dynamic reference position. I.e., putting the end-effector close to a cup (cf. Fig. 7)
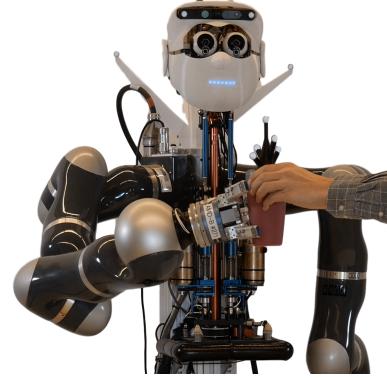

Fig. 7. Illustration of the hardware experiment where the goal is to place Apollo's hand close to the cup. The cup's position is estimated using the Vicon system.

such that it could be grabbed. The cup's position is estimated using the Vicon system and included in the sensory information.

During training, we choose a random initial position of the robot arm and a random reference position (representing the cup) inside the interval $([-0.2, 0.5], \ [0.55, 0.9], \ [-0.15, 0.45])^\mathrm{T}$m, for each trajectory. At each simulated timestep, the reference position is reset to another randomly sampled point inside this interval with probability 1%. This procedure should already account for the fact that during the evaluation on the real system, the reference will change dynamically.

The policies, whose results are presented in the following, have been trained for 2250 epochs in simulation with a communication penalty of $\lambda = 0.1$. To incentivize reaching the final position more accurately, we added an inverse term to the reward function R, which is thus given by $R = \sum_{k=0}^{N} \gamma^k (R_{ctrl}[k] + R_{comm}[k]) = \sum_{k=0}^{N} \gamma^k (-3(0.01\|u[k]\|_2^2 + 10\|x_{ref}[k] - x_{ef}[k]\|_2^2 + 0.01\|\dot{x}_{ef}[k]\|_2^2 + \lambda\delta[k]) + \frac{0.05}{\|x_{ref}[k] - x_{ef}[k]\|_2^2})$. The parameters have been obtained empirically and reflect a standard cost function with an increased emphasis on reaching the target while ensuring that the overall cost stays in the same order of magnitude as for the previous experiments. We chose a factor of 0.05 for the inverse term as we view 5 cm as close enough to the goal position.

In Fig. 8, we depict an exemplary rollout of the model while tracking the reference signal provided by the Vicon system. It is striking that 85% of communication can be saved while still reaching the dynamic desired reference reliably. As can be seen, the reference signal is sometimes slightly disturbed. However, although only trained in simulation without such nonidealities, the learning agent is robust to these disturbances. Thus, we conclude that we did not overfit and that our problem formulation and implementation results in a stable and robust simu-
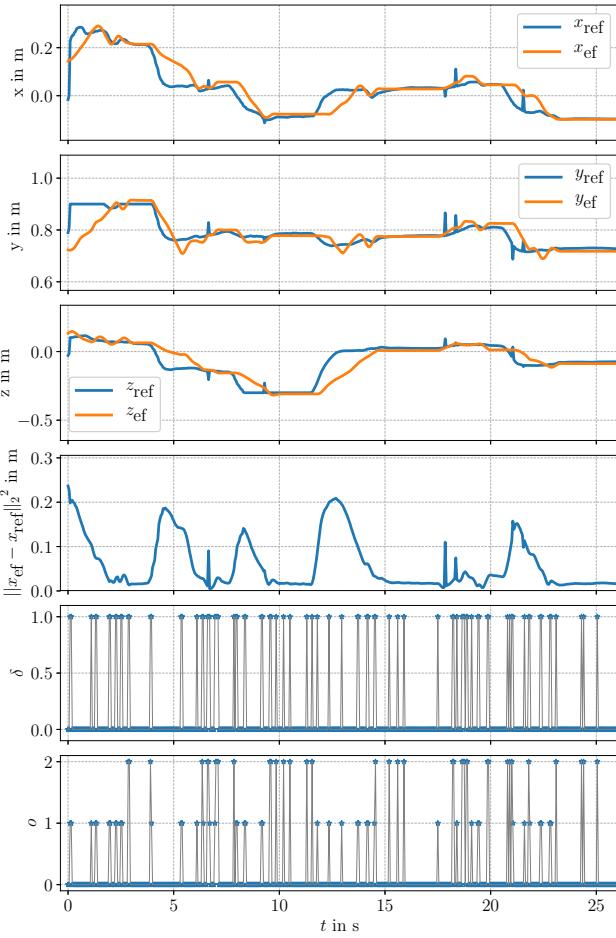
11

Fig. 8. Exemplary rollout of a policy trained for the task of reaching a dynamic reference position with Apollo's right hand, see Fig. 7. The reference (i.e., the cup's position) is provided by the Vicon system. During the rollout, the policy saves 85% of communication, while the reference is still tracked reliably. This performance is consistent among various runs.

lation to reality transfer. The corresponding video illustrates the responsiveness of the final policy and confirms the low tracking error. The learned controller drives the hand close enough to the cup such that it can be grabbed.

7.3 Obstacle Avoidance

To further illustrate our learning algorithm's capabilities, we increase the difficulty and demonstrate the resulting performance for the inherently more complex, nonlinear task of reaching a desired end-effector position in the face of an obstacle, as illustrated in Fig. 1. Now, it is not sufficient to simply drive the end-effector in the direction of the reference. Instead, multiple changes of direction are necessary to avoid the obstacle.

The state available to the agent is the same as introduced in Sec. 7.1. Thus, the learning agent is only aware of the

end-effector position x_{ef} in task space. The algorithm's goal is to reach the desired reference position x_{ref} without hitting the obstacle with the end-effector. We assume a static obstacle position, as otherwise, this information would need to be provided to the agent. The reference position is now defined in software and not retrieved via the Vicon system.

We adapt the previously presented training procedure as follows. The starting and reference positions are sampled from the intervals $([0.45, 0.55], [0.7, 0.9], [0.0, 0.2])^{\mathrm{T}}\mathrm{m}$, and $([-0.2, -0.1], [0.7, 0.9], [-0.1, 0.1])^{\mathrm{T}}\mathrm{m}$, respectively. This way, they are separated, with the obstacle in between at $([-0.05, 0.35], [0.5, 1.0], [-1.0, 0.1])^{\mathrm{T}}\mathrm{m}$. Moreover, during training only, before applying the control action to the system, we predict the next position of the end-effector using integrator dynamics, i.e., $x_{\mathrm{pred,ef}}[k+1] = x_{\mathrm{ef}}[k] + \Delta T u_{\mathrm{ac}}[k]$. If the next predicted position $x_{\mathrm{pred,ef}}[k+1]$ lies inside the obstacle, we apply $u_{\mathrm{ac}}[k] = (0, 0, 0)^{\mathrm{T}}$ to the system, and additionally penalize this state/action combination in the reward function. Thus, the reward is now given by $R = \sum_{k=0}^{N} \gamma^k (R_{\mathrm{ctrl}}[k] + R_{\mathrm{comm}}[k] + R_{\mathrm{obst}}[k]) = \sum_{k=0}^{N} \gamma^k (-3(0.01\|u[k]\|_2^2 + 10\|x_{\mathrm{ref}}[k] - x_{\mathrm{ef}}[k]\|_2^2 + 0.01\|\dot{x}_{\mathrm{ef}}[k]\|_2^2 + \lambda\delta[k]) + \frac{0.05}{\|x_{\mathrm{ref}}[k] - x_{\mathrm{ef}}[k]\|_2^2} - \zeta[k])$, where

$$\zeta[k] = \begin{cases} 0, & \text{if } x_{\mathrm{pred,ef}}[k+1] \text{ not inside obstacle area} \\ 5, & \text{otherwise.} \end{cases}$$

For obtaining the results presented in Fig. 9, we first pretrain the policy for 2250 epochs in simulation and then evaluate it on the real hardware.

As shown in Fig. 9, Fig. 1, and the associated video, the reference position is reached reliably without hitting the obstacle, while still 92 % of communication can be saved. This behavior is consistent among different starting and reference positions. However, compared to the previously presented reference tracking, without the obstacle present, the reference position is not reached as accurately. The reason for this behavior is that the presence of the obstacle limits the freedom of the arm and the policy, resulting in less accuracy. The fact that the reference position is now defined in software and not noisy could explain why more communication can be saved compared to the previously presented dynamic cup reaching scenario (Sec. 7.2).

8 Towards Stability of Event-triggered Control

In the previous sections, we tackle the ETC problem formulation using model-free, deep RL. The results of our learning algorithm are NN policies consisting of many parameters. In this setting, it is usually difficult to provide stability guarantees. Yet, since the learned policies are also envisioned to be used in real-world scenarios, such guarantees are essential. In the following,
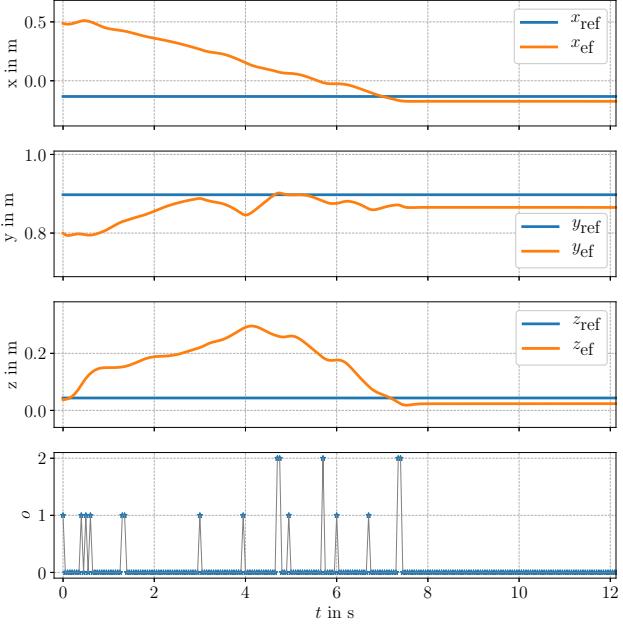
Fig. 9. Exemplary obstacle avoidance trajectory, where a fixed reference position is to be reached in the face of an obstacle, cf. Fig. 1. During the rollout, the policy saves 92 % of communication, while the reference is reached reliably without hitting the obstacle. This performance is consistent among various runs.

we present an approach toward checking the stability of learned NN policies. As a starting point, in this work, we restrict to NNs parametrized with ReLUs, and known, linear system dynamics.

8.1   Stability Verification of Neural Network Policies

The following stability analysis is based on the Marabou framework proposed in [15,42], which allows to check for properties of deep NNs. The Marabou framework is an expansion of the Simplex method [43]. Simplex tries to find a valid assignment to a linear program. It either returns an admissible value for all the free variables involved that result in satisfying the query, or returns that the query is not satisfiable. As piecewise linear activation functions can be interpreted as case dependent linear constraints, the authors expand the algorithm such that the same satisfiability queries, as for the Simplex method, can be posed with respect to NNs.

We exploit this framework to come up with stability guarantees for trained policies. For our considerations, the only limitation is that exclusively piecewise linear activation functions, i.e., ReLU activations can be used within the networks. Nevertheless, by combining the algorithm with assumptions on the system dynamics, it is possible to provide stability guarantees through output range analysis for the learned NN policies. Further, in case the policy does not fulfill the stability requirements
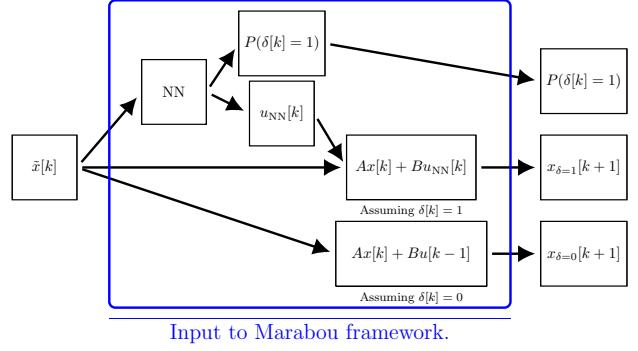

Input to Marabou framework.

Fig. 10. Schematic of the pipeline used for the stability verification procedure. The parts inside the blue box can all be parametrized using ReLU units. Therefore, the Marabou framework can check whether the next state is inside the same set as the initial state. This is the basis for our stability analysis.

straight away, the framework can be used to develop a retraining procedure to refine the NNs.

8.2   Stability Analysis and Retraining of Event-triggered Control Policies

We define stability as finding a positive invariant set in the state space: once inside this set, when applying the NN policy, the next state is guaranteed to also lie within this set.

Definition 1 (Stability)
The system f is considered stable under the NN control policy $u_{NN}[k] = h_{NN}(\tilde{x}[k])$ if there exists a region $\mathcal{M}$, such that
$x[k] \in \mathcal{M} \Rightarrow x[k+1] = f(x[k], h_{NN}(\tilde{x}[k])) \in \mathcal{M} \; \forall x[k] \in \mathcal{M}.$

While the RL algorithm presented in Alg. 1 is model-free and can be used for systems with linear or nonlinear dynamics, the stability check relies on the underlying system exhibiting known, linear dynamics. The algorithm then checks whether the control policy is stable, according to Def. 1. Linear system dynamics can also be represented using ReLUs. Thus, algorithmically, we can design one NN that takes as the input the current state $x[k]$ and outputs the next state $x[k+1]$, cf. Fig. 10. With this network given, we can exploit the Marabou framework [42] to check for the desired properties.

8.2.1   Stability Analysis of the Policies

In the event-triggered setting, the next state depends on the communication decision. Therefore, effectively two next states have to be calculated, as shown in Fig. 10. To make use of the deterministic Marabou framework, we need to eliminate the stochasticity of the communication decision. Hence, we always choose to communicate in case the probability is larger than or equal to 50 %.
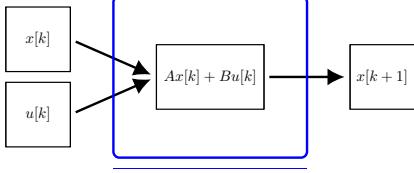
Input to Marabou framework.

Fig. 11. Schematic of the pipeline that is used to find an admissible input $u[k]$ for an unstable point $x[k]$. The parts inside the blue box can all be parametrized using ReLU units.

The resulting algorithm that checks for the stability of a region $\mathcal{M}$ is presented in Alg. 2. As the input to the stability verification framework (see Fig. 10) is given by $\tilde{x}[k]$, we additionally define $\mathcal{S} = (\mathcal{M}, \ \mathcal{L})^{\mathrm{T}} = (\mathcal{M}, \ [-u_{\lim}, u_{\lim}])^{\mathrm{T}}$, which combines the stable region with the input range. Adding the input range is necessary as it covers the potential range of the previously applied control actions, which are reapplied in case the ZOH is selected. If the algorithm returns an empty set of points, we know that given the region $\mathcal{M}$, the input range, and the system dynamics, the ETC policy is stable. Otherwise, the algorithm directly outputs exemplary unstable points $\tilde{x}[k]$. There are two possibilities, why a point $\tilde{x}[k]$ is unstable under the ETC policy. Either the policy erroneously decides to skip communication, and the reapplication of the previous control input results in the next state outside the invariant set. Or, in case of communication, the policy chooses an unstable control input.

Algorithm 2 Check for Stability in ET setting ($\mathcal{S}$)

1: points=[]
2: Marabou query: $\tilde{x}[k] \in \mathcal{S}$, $P(\delta[k] = 1) \geq 0.5 \Rightarrow x_{\delta=1}[k+1] \in \mathbb{R}^n \setminus \mathcal{M}$
3: **if** Valid assignment is found: **then** points.append($\tilde{x}[k]$)
4: Marabou query: $\tilde{x}[k] \in \mathcal{S}$, $P(\delta[k] = 1) < 0.5 \Rightarrow x_{\delta=0}[k+1] \in \mathbb{R}^n \setminus \mathcal{M}$
5: **if** Valid assignment is found: **then** points.append($\tilde{x}[k]$)
6: **return** points

8.2.2 Retraining Neural Network Policies

If Alg. 2 does not indicate stability, i.e., returns a non-empty set of points, it is possible to refine the ETC policy such that it fulfills the invariance property using Alg. 3.

At the core of this algorithm is the *FindValidInputET* function (ll. 1-3). In case the stability verification algorithm (Alg. 2) returns an unstable point, we feed it into the schematic presented in Fig. 11 and Marabou returns a value for the input $u[k]$ such that the next state $x[k+1]$ also lies inside the invariant set.

In case a point $\tilde{x}[k]$ is unstable using the current policy, both parts of the event-triggered controller have to be adapted. On the one hand, it is crucial to ensure that an appropriate control action is chosen. On the other hand,

we have to ensure that the policy decides to communicate. This supervised retraining for unstable points can be seen in ll. 34-38 of Alg. 3. We define that at those points, the probability of communication should be set to 60 %. Note that any choice above 50 % is admissible.

When using the previously described mechanism, all the refined policies exhibit significantly reduced communication savings as any unstable configuration can only be resolved using communication. Thus, we found it beneficial to sample additional points using Sobol sequences [44] to cover the whole state space as thoroughly as possible (l. 20 of Alg. 3). We then add the condition to the retraining procedure that communication must be saved whenever possible (ll. 9-14 and 30-32). This can also be interpreted as trying to be maximally resource-efficient.

Combining the individual parts results in Alg. 3. If the algorithm terminates, we know that the ETC law is stable for the defined region $\mathcal{M}$, the input range, and the assumed system dynamics.

8.3 Simulation Example

In the following section, we demonstrate the previously presented algorithm. The task is to obtain a stable event-triggered controller for stabilizing the inverted pendulum on top in the OpenAI Gym Pendulum environment. To obtain the linear dynamics required for the algorithm, we linearize the nonlinear pendulum dynamics around the upper equilibrium point $\theta = 0°$ and $\dot{\theta} = 0°/\mathrm{s}$. The conversion from continuous to discrete-time is done using the matrix exponential method. We use the exact same configuration as in Sec. 6.1. In line with Definition 1, we define the safe state space ($\mathcal{M}$) to capture ranges of $\theta \in [-2.5°, 2.5°]$ and $\dot{\theta} \in [-5°/\mathrm{s}, 5°/\mathrm{s}]$. Usually, the state of the pendulum environment that is fed into the NNs is given by $x[k] = (\cos(\theta[k]), \ \sin(\theta[k]), \ \dot{\theta}[k])^{\mathrm{T}}$. However, for the stability verification algorithm, we need to express the NN policies' output in terms of the system variables, i.e., $\theta[k]$ and $\dot{\theta}[k]$. Thus, we can neither use the nonlinear sine nor the cosine function and linearize the state of the environment around the equilibrium, which results in $x[k] \approx (1.0, \ \theta[k], \ \dot{\theta}[k])^{\mathrm{T}}$.

The results of running the retraining procedure (Alg. 3) are shown in Fig. 12. We initialize the procedure with the linearized dynamics and a policy that has been trained for 500 epochs using our proposed training algorithm (Alg. 1). As can be seen in Fig. 12a, this policy is not stable yet. Running the retraining procedure for 4 epochs results in a guaranteed stable event-triggered controller that also successfully stabilizes the nonlinear system around the equilibrium point (see Fig. 12b). Although we only trained for an angular region between $-2.5°$ and $2.5°$, the policy also keeps the pendulum upright when starting outside of this interval, as shown

Algorithm 3 Refine Policy ET
___

1: **function** FINDVALIDINPUTET($\tilde{x}[k], \mathcal{M}$)
2:   Marabou query: find $u[k]$, s.t. $x[k], u[k] \Rightarrow x[k+1] \in \mathcal{M}$
3:   **return** $u[k]$

4: **function** CHECKPOINTET($\tilde{x}[k], \mathcal{M}$)
5:   points=[]
6:   Marabou query: $\tilde{x}[k] \Rightarrow x[k+1] \in \mathbb{R}^n \setminus \mathcal{M}$
7:   **if** Valid assignment is found: **then** points.append($\tilde{x}[k]$)
8:   **return** points

9: **function** COMMSAVINGPOSSIBLE($\tilde{x}[k], \mathcal{M}$)
10:   points=[]
11: /* Check whether next state is also stable without communicating.
12:   Marabou query: $\tilde{x}[k] \Rightarrow x_{\delta=0}[k+1] \in \mathcal{M}$ for $P(\delta[k]=1) \geq 0.5$
13:   **if** Valid assignment is found: **then** points.append($\tilde{x}[k]$)
14:   **return** points
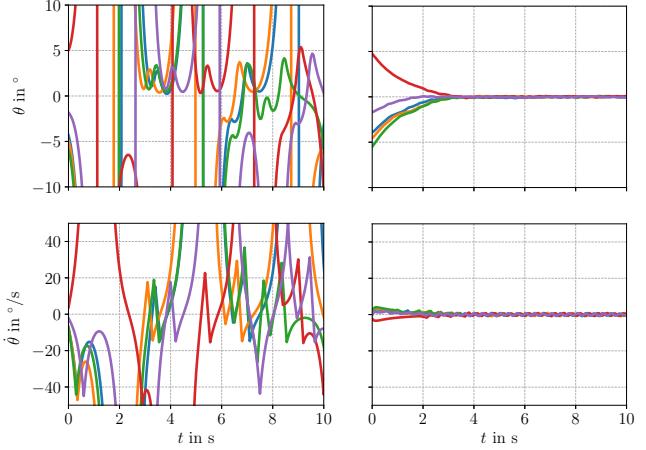
15: Choose region $\mathcal{M}$
16: Define region $\mathcal{S} = (\mathcal{M}, \ [-u_{\lim}, u_{\lim}])^{\mathrm{T}}$
17: points$_{\mathrm{crit}}$ = Check for Stability ($\mathcal{S}$), see Alg. 2
18: **while** not points$_{\mathrm{crit}}$ is EMPTY **do**
19: /* Use Sobol sequences to generate additional points
20:   points$_{\mathrm{sobol}}$ = Use sobol sequence to sample from $\mathcal{S}$
21:   points$_{\mathrm{commsav}}$ = []
22:   **for** $\tilde{x}[i]$ in points$_{\mathrm{sobol}}$ **do**
23:     points$_{\mathrm{crit}}$.append(CHECKPOINTET($\tilde{x}[i], \mathcal{M}$))
24:     points$_{\mathrm{commsav}}$.append(COMMSAVINGPOSSIBLE($\tilde{x}[i], \mathcal{M}$))
25: /* Calculate admissible input for all the unstable points.
26:   u_{crit} = []
27:   **for** $\tilde{x}[i]$ in points$_{\mathrm{crit}}$ **do**
28:     u_{crit}.append(FINDVALIDINPUTET($\tilde{x}[i], \mathcal{M}$))
29: /* Supervised retraining to enforce communication savings
30:   **for** number of optimizer epochs **do**
31:     $L_2(\theta_2) = (P(\delta[k]=1|\text{points}_{\mathrm{commsav}}, \theta_2) - 0.4)^2$
32:     $\theta_2 = \theta_2 - \alpha_{\theta_2} \frac{\partial L_2}{\partial \theta_2}$
33: /* Supervised retraining of the NN policy for critical points
34:   **for** number of optimizer epochs **do**
35:     $L_1(\theta_1) = (u_{\mathrm{NN}}(\text{points}_{\mathrm{crit}}, \theta_1) - u_{\mathrm{crit}})^2$
36:     $L_2(\theta_2) = (P(\delta[k]=1|\text{points}_{\mathrm{crit}}, \theta_2) - 0.6)^2$
37:     $\theta_1 = \theta_1 - \alpha_{\theta_1} \frac{\partial L_1}{\partial \theta_1}$
38:     $\theta_2 = \theta_2 - \alpha_{\theta_2} \frac{\partial L_2}{\partial \theta_2}$
39:   points$_{\mathrm{crit}}$ = Check for Stability ($\mathcal{S}$), see Alg. 2
___

(a) Exemplary rollouts for trying to stabilize a pendulum with an event-triggered policy, which has been trained for 500 optimization epochs. This policy does not succeed in keeping the pendulum upright.

(b) Exemplary rollouts for stabilizing a pendulum after retraining the policy from Fig. 12a with Alg. 3 for 4 epochs. This policy is guaranteed to stabilize the pendulum on top.

Fig. 12. Effect of the retraining procedure presented in Alg. 3. In the plots, each color represents a different rollout. While the initial policy is unstable (Fig. 12a), after refining this policy for 4 iterations using Alg. 3, a guaranteed stable controller is obtained which successfully stabilizes the pendulum and still saves about 70 % of communication (Fig. 12b).
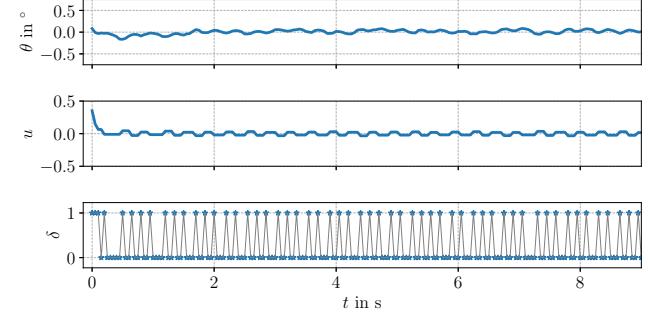

Fig. 13. Illustration of an exemplary rollout of the same policy as shown in Fig. 12b.

in Fig. 12b. Over 10 randomly started runs, the policy saves around 70 % of communication. An exemplary rollout of this very resource-efficient and guaranteed stable control policy is presented in Fig. 13.

9 Conclusion

In this paper, we propose a model-free hierarchical RL algorithm capable of jointly learning event-triggered control policies from scratch. Without any modification, the algorithm can be applied to linear and nonlinear, low- and high-dimensional systems. In those high-dimensional environments, communication savings of up to 80 % can be reported. To the best of our knowledge, our algorithm is the first that can obtain

event-triggered policies for such environments through joint optimization.

The algorithm is also successfully deployed on real hardware, i.e., the Apollo robot. We provide a demonstration for resource-efficient setpoint tracking and obstacle avoidance while saving around 85 % and 90 % of communication, respectively. These results imply that the presented algorithm scales to partially observable settings, to using more than 2 options, and to imperfect communication settings with potential delays.

Moreover, we show a novel algorithm for evaluating the stability of linear systems controlled by NN policies. In

case the learned event-triggered policy initially does not yield the desired stability guarantee, we propose a re-training procedure for refining the previously unstable policy. Scaling those ideas to nonlinear environments, as well as higher-dimensional systems through handling the increased computational complexity, is subject to ongoing research.

Acknowledgements

References

[1] Joo P Hespanha, Payam Naghshtabrizi, and Yonggang Xu. A survey of recent results in networked control systems. *Proceedings of the IEEE*, 95(1):138–162, 2007.

[2] Jan Lunze. *Control Theory of Digitally Networked Dynamic Systems*, volume 1. Springer, 2014.

[3] X. Zhang, Q. Han, and X. Yu. Survey on recent advances in networked control systems. *IEEE Transactions on Industrial Informatics*, 12(5):1740–1752, 2016.

[4] WPMH Heemels, Karl Henrik Johansson, and Paulo Tabuada. An introduction to event-triggered and self-triggered control. In *IEEE Conference on Decision and Control (CDC)*, pages 3270–3285, 2012.

[5] Marek Miskowicz. *Event-based control and signal processing*. CRC press, 2018.

[6] Lars Grüne, Sandra Hirche, Oliver Junge, Péter Koltai, Daniel Lehmann, Jan Lunze, Adam Molin, Rudolf Sailer, Manuela Sigurani, Christian Stöcker, et al. Event-based control. In *Control Theory of Digitally Networked Dynamic Systems*, pages 169–261. Springer, 2014.

[7] Michael Lemmon. Event-triggered feedback in control, estimation, and optimization. In *Networked Control Systems*, pages 293–358. Springer, 2010.

[8] Sebastian Trimpe and Raffaello D'Andrea. An experimental demonstration of a distributed and event-based state estimation algorithm. *IFAC World Congress*, 44(1):8811–8818, 2011.

[9] José Araújo, Manuel Mazo, Adolfo Anta, Paulo Tabuada, and Karl H Johansson. System architectures, protocols and algorithms for aperiodic wireless control systems. *IEEE Transactions on Industrial Informatics*, 10(1):175–184, 2013.

[10] Victor S Dolk, Jeroen Ploeg, and WP Maurice H Heemels. Event-triggered control for string-stable vehicle platooning. *IEEE Transactions on Intelligent Transportation Systems*, 18(12):3486–3500, 2017.

[11] Burak Demirel, Arunselvan Ramaswamy, Daniel E Quevedo, and Holger Karl. DeepCAS: A deep reinforcement learning algorithm for control-aware scheduling. *IEEE Control Systems Letters*, 2(4):737–742, 2018.

[12] Kyriakos G Vamvoudakis and Henrique Ferraz. Model-free event-triggered control algorithm for continuous-time linear systems with optimal performance. *Automatica*, 87:412–420, 2018.

[13] Chithrupa Ramesh, Henrik Sandberg, Lei Bao, and Karl Henrik Johansson. On the dual effect in state-based scheduling of networked control systems. In *American Control Conference*, pages 2216–2221. IEEE, 2011.

[14] Richard S Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.

[15] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117. Springer, 2017.

[16] Xiangnan Zhong, Zhen Ni, Haibo He, Xin Xu, and Dongbin Zhao. Event-triggered reinforcement learning approach for unknown nonlinear continuous-time system. In *International Joint Conference on Neural Networks (IJCNN)*, pages 3677–3684. IEEE, 2014.

[17] Xiong Yang, Haibo He, and Derong Liu. Event-triggered optimal neuro-controller design with reinforcement learning for unknown nonlinear systems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2017.

[18] Avimanyu Sahoo, Hao Xu, and Sarangapani Jagannathan. Neural network-based event-triggered state feedback control of nonlinear continuous-time systems. *IEEE Transactions on Neural Networks and Learning Systems*, 27(3):497–509, 2015.

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

[20] Kazumune Hashimoto, Yuichi Yoshimura, and Toshimitsu Ushio. Learning self-triggered controllers with Gaussian processes. *arXiv preprint arXiv:1909.00178*, 2019.

[21] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[22] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338, 2016.

[23] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[24] Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21(4):682–697, 2008.

[25] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[26] Warwick Masson, Pravesh Ranchod, and George Konidaris. Reinforcement learning with parameterized actions. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[27] Matthew Hausknecht and Peter Stone. Deep reinforcement learning in parameterized action space. In *International Conference on Learning Representations (ICLR)*, May 2016.

[28] Michael Neunert, Abbas Abdolmaleki, Markus Wulfmeier, Thomas Lampe, Jost Tobias Springenberg, Roland Hafner, Francesco Romano, Jonas Buchli, Nicolas Heess, and Martin Riedmiller. Continuous-discrete reinforcement learning for hybrid control in robotics. *arXiv preprint arXiv:2001.00449*, 2020.

[29] Richard S Sutton. TD models: Modeling the world at a mixture of time scales. In *Machine Learning Proceedings 1995*, pages 531–539. Elsevier.

[30] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[31] Fabio Bonassi, Enrico Terzi, Marcello Farina, and Riccardo Scattolini. LSTM Neural Networks: Input to state stability and probabilistic safety verification. *Learning for Dynamics and Control*, 2020.

[32] Benjamin Karg and Sergio Lucia. Stability and feasibility of neural network-based controllers via output range analysis. *arXiv preprint arXiv:2004.00521*, 2020.

[33] Martin Klissarov, Pierre-Luc Bacon, Jean Harb, and Doina Precup. Learnings options end-to-end for continuous action tasks. *Hierarchical Reinforcement Learning Workshop (NIPS)*, 2017.

[34] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[35] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[36] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym, 2016.

[37] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5026–5033, 2012.

[38] M. C. F. Donkers and W. P. M. H. Heemels. Output-based event-triggered control with guaranteed $\mathcal{L}_\infty$-gain and improved and decentralized event-triggering. *IEEE Transactions on Automatic Control*, 57(6):1362–1376, 2012.

[39] Paulo Tabuada. Event-triggered real-time scheduling of stabilizing control tasks. *IEEE Transactions on Automatic Control*, 52(9):1680–1685, 2007.

[40] Justin Carpentier, Guilhem Saurel, Gabriele Buondonno, Joseph Mirabel, Florent Lamiraux, Olivier Stasse, and Nicolas Mansard. The Pinocchio C++ library: A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives. In *IEEE/SICE International Symposium on System Integration (SII)*, pages 614–619, 2019.

[41] Stefan Schaal. The SL simulation and real-time control software package. Technical report, Los Angeles, CA, 2009.

[42] Guy Katz, Derek A Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, et al. The Marabou framework for verification and analysis of deep neural networks. In *International Conference on Computer Aided Verification*, pages 443–452. Springer, 2019.

[43] George Dantzig. *Linear Programming and Extensions*. Princeton University Press, 1963.

[44] Ilya M Sobol. Uniformly distributed sequences with an additional uniform property. *USSR Computational Mathematics and Mathematical Physics*, 16(5):236–242, 1976.

[45] Daniel Kappler, Franziska Meier, Jan Issac, Jim Mainprice, Cristina G. Cifuentes, Manuel Wüthrich, Vincent Berenz, Stefan Schaal, Nathan Ratliff, and Jeanette Bohg. Real-time perception meets reactive motion generation. *IEEE Robotics and Automation Letters*, 3(3):1864–1871, 2018.

A Parametrization of the Proposed Algorithm

In this section, we provide insights on how we implement the major components needed for the proposed learning algorithm (Alg. 1).

As explained in Sec. 5.2, our implementation is based on three main components, the policy over options $\mu(o|\tilde{x})$, the intra option policy $\pi(u|\tilde{x}, o)$, and the Q-function $Q(\tilde{x}, u)$. Fig. A.1, Fig. A.2, and Fig. A.3 show the standard implementations of the respective components for the case of using 2 options. As done in standard RL, we normalize the input before it is passed to the networks and clip the output of the intra option policy to reflect the input constraints of the physical system. Fig. A.2 illustrates that for the case of performing the ZOH, i.e., option 0, the intra option policy does not have to be evaluated. Thus, computational resources can be saved in the forward, as well as the backward pass. For implementing the NN estimator of the Q-function (see Fig. A.3), we decided to split the estimates for option 0 and option 1 already before the first hidden layer. The reason for this choice is that as the two options are very different, we also expect very different Q-values for the two options, although being in the same state $\tilde{x}$. This is because option 0 is limited to the ZOH, while option 1 can basically apply any action, depending on the intra option policy. For the same reasons, we arrive at the design choice for the policy over options (see Fig. A.1).

For the stability verification and retraining procedure, we apply the following modifications to the previously presented implementation. Instead of using the hyperbolic tangent (TanH) activation function, we apply the ReLU activation. The number of hidden neurons is decreased from 64 to 32. This is due to the fact that at the core of the verification algorithm, we run a modified version of the Simplex algorithm, which simply runs faster if less neurons are used. Further, instead of using the softmax activation function for the policy over options, we calculate $Z = \zeta_0 - \zeta_1$. If $Z > 0$, this corresponds to performing the ZOH and choosing option 0, otherwise we use option 1. That way, we achieve deterministic behavior of the policy over options and avoid using the softmax activation function, which is incompatible with the verification framework. Moreover, this deterministic decision is compatible with the stochastic case, as $Z = 0$ is equal to $\mu(o = 0|\tilde{x}) = \mu(o = 1|\tilde{x}) = 50\,\%$, and $Z > 0$ corresponds to $\mu(o = 0|\tilde{x}) > \mu(o = 1|\tilde{x})$.

B Learning Hierarchical, Periodic Control

Using the proposed algorithm for ETC is one possibility. However, it is also possible to use it as a normal, hierarchical control algorithm where the two options represent two different NN policies that we can sample from. The original PPOC [33] algorithm has also been devel-
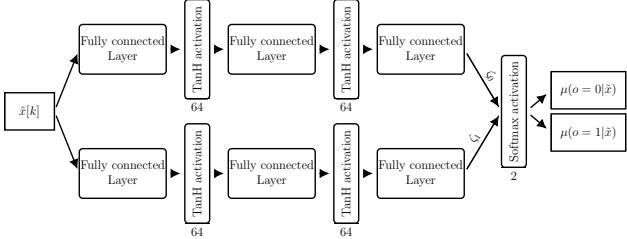
Fig. A.1. Illustration of the parametrization of the policy over options. In this specific implementation, each hidden layer consists of 64 neurons, and the TanH and softmax activation functions are used. The variables ζ_0 and ζ_1 represent intermediate values.
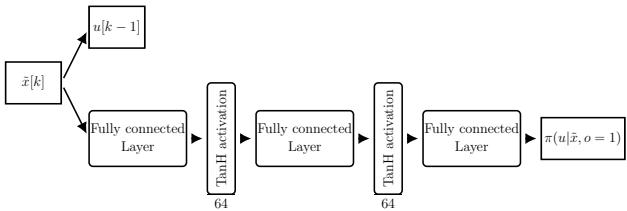

Fig. A.2. Illustration of the parametrization of the intra option policy. In this specific implementation, each hidden layer consists of 64 neurons, and the TanH activation function is used.
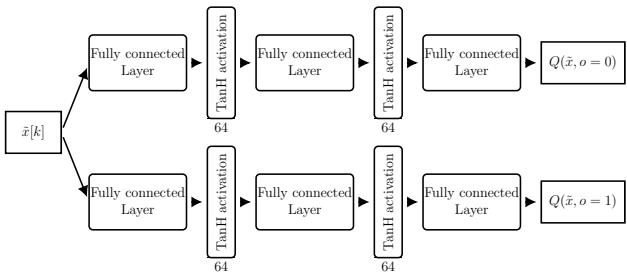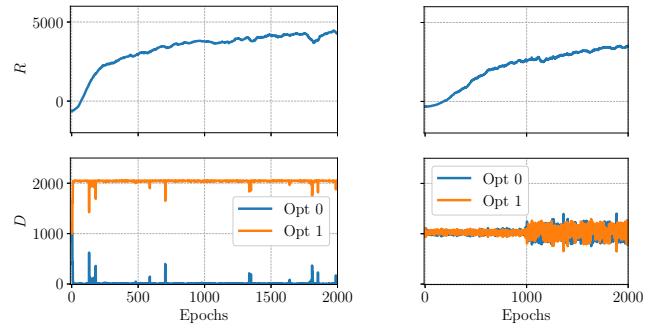

Fig. A.3. Illustration of the parametrization of the Q-function estimator. In this specific implementation, each hidden layer consists of 64 neurons, and the TanH activation function is used.

oped for this case of standard periodic control without any event-triggering involved.

Considering this setting, Fig. B.1 exemplarily shows the difference in the learning progress between using the PPOC and the proposed algorithm for the Cheetah environment. As shown in Fig. B.1a, the PPOC algorithm quickly collapses to essentially only using one of the options as the other one is almost never executed. In contrast, as illustrated in Fig. B.1b, our algorithm ensures that both of the options are used. This is due to the entropy scheduling that prevents the policy over options from being too greedy and also due to the PPO updates. Considering the reward, this might slightly slow down the learning process but is an essential property of the algorithm, which is important when it comes to ETC. In ETC, the smooth learning process allows to

(a) Training progress, using the PPOC algorithm [33], in the setting of periodic control, without any event-trigger.

(b) Training progress, using our algorithm, in the setting of periodic control, without any event-trigger.

Fig. B.1. Comparing the training progress for two exemplary policies, one using the PPOC and the other one using our algorithm for the setting of periodic control in the Half-Cheetah environment. Each epoch consists of 2048 (x, u)-transitions. The lower plot illustrates how many of those transitions D are conducted using option 0 or option 1.

arrive at policies with intermediate communication savings. On the contrary, we think that the greedy optimization of the PPOC algorithm is the reason why its event-triggered implementation always collapses to one of the extreme cases, either saving no communication at all or never communicating.

C Deatils on the Robot Experiments

For running the robot experiments, three main components are needed: The Apollo robot executing the control actions; the learning agent, running on a computer, using Ubuntu 14.04 together with a Xenomai kernel that sends the control commands to the robot; and the sensors monitoring the robot, which are Apollo's internal sensors and a Vicon camera system. Using the Xenomai kernel is essential, as this allows us to check whether timing constraints are violated. The Apollo robot is equipped with two KUKA LBR4+ robotic arms. Each of the arms consists of 6 joints. All the experiments presented in this work only use the right arm where at the end, a Barrett Hand is mounted onto the arm. Further details on Apollo can be found in [45]. For controlling and simulating the robot, we use the simulation laboratory (SL) framework [41]. As the SL package is programmed in C, but our learning algorithms are implemented in Python and using Tensorflow, we use a shared memory to exchange information between the Python-based learning pipeline and the C Code, which takes care of the actual robot control.