Near-optimal strategies for nonlinear networked control systems using optimistic planning

Lucian Buşoniu Romain Postoyan Jamal Daafouz

Abstract— **We consider the scenario where a controller communicates with a general nonlinear plant via a network, and must optimize a performance index. The problem is modeled in discrete time and the admissible control inputs are constrained to belong to a finite set. Exploiting a recent optimistic planning algorithm from the artificial intelligence field, we propose two control strategies that take into account communication constraints induced by the use of the network. Both resulting algorithms have guaranteed near-optimality. In the first strategy, input sequences are transmitted to the plant at a fixed period, and we show bounded computation. In the second strategy, the algorithm decides the next transmission instant according to the last state measurement (leading to a self-triggered policy), working within a fixed computation budget. For this case, we guarantee long transmission intervals. Examples and simulation experiments are provided throughout the paper to illustrate the results.**

I. Introduction

In a variety of applications, controllers are implemented over networks in order to reduce installation costs and to facilitate maintenance, leading to *networked control systems* (NCS). The control law therefore has to share the communication bandwidth with other tasks. This constraint cannot be ignored in general as it may have a serious impact on the system performance. Various methodologies for NCS have been developed over these last decades. Two main approaches are distinguished based on whether the transmissions are defined by a clock, see e.g. [1], [2], or are triggered depending on the state of the plant, in which case we talk of *event-* or *self-triggered control*, see e.g. [3]–[8]. However, most of these works focus on stabilization or estimation problems, while few address optimal control, and then mostly for linear systems, e.g. [9]–[17]. One interesting exception is [18], where model-predictive control is used to address nonlinear systems with quadratic costs, focusing on stability. Thus the general problem of optimal control in nonlinear NCS remains largely open.

In this paper, we propose an online approach for the near-optimal control of nonlinear NCS with general costs. We focus on the challenge of reducing network usage, without considering other effects such as delays, packet drop-outs, etc. We borrow from the artificial intelligence community a recent *optimistic planning* (OP) algorithm [19], which works in discrete time and for very general optimal control problems, having any nonlinear system dynamics. The quality of state transitions is measured by bounded rewards and the objective is to maximize the cumulative reward. At each execution, the algorithm explores possible sequences of actions (inputs) from the current state, using their predicted cumulative rewards to guide the search. Thus, OP can be seen as model-predictive control, while also being based on insights from bandit theory [20], optimization, and classical planning (heuristic search). Several OP algorithms have been introduced, e.g. [21]–[24], and they have shown good performance in problems from control [24], medicine [23], and artificial intelligence, where they were the first to achieve expert-level play in the game of Go [25]. In the sequel, we use the method of [19] and call it simply 'OP'.

Crucially for our approach, from the analysis of OP [19] it turns out that it returns a *long sequence* of actions that has *near-optimal* performance. Thus, rather than sending only the first action in the sequence and then rerunning the algorithm, as in [19], we choose to send a longer subsequence. This simple idea allows us to reduce the need for communication (and also computation) while guaranteeing near-optimality.

We propose two strategies. In the first, communication between the plant and the controller is set to occur at a fixed period which is freely selected. We then investigate the resulting near-optimality and the induced computational complexity. The second strategy enforces a fixed computation budget at every OP execution, and within this budget generates a sequence of actions from the last measured plant state. As a result, the communication interval adapts to the state, leading to a self-triggered policy, e.g. [5], [6], [8]. We then investigate how sequence length and near-optimality vary with the computation budget. Both strategies allow sending only an initial part of the sequences found, spanning a spectrum from the original method [19] which only applies the first action, to applying the complete sequences. Interestingly, shorter subsequences may do better in some problems, but worse in others, and we provide results and insight about this phenomenon.

OP requires a finite set of actions, so its performance is limited by discretization accuracy. Nevertheless, as will be illustrated in an example, the loss due to discretization is often manageable. Further, the limitation is not fundamental and continuous-action optimistic methods exist, e.g. [24], although their analysis is not as well developed. On the other hand, discrete actions may even be preferable due to their benefits in NCS. Indeed, the size of communication packets can be reduced by encoding the discrete actions by their index, and actuator saturation can be dealt with by simply discretizing within the operating ranges. Other authors have shown interest and theoretical guarantees for coarsely-discretized control [26].

The authors are with the Université de Lorraine, CRAN, UMR 7039 and the CNRS, CRAN, UMR 7039, France ({lucian.busoniu, romain.postoyan, jamal.daafouz}@univ-lorraine.fr). L. Buşoniu is also associated with the Automation Department, Technical University of Cluj-Napoca, Romania. J. Daafouz is also with the IUF.

Next, Section II describes the optimal control problem and OP with its guarantees. In Section III, we introduce and analyze our approach. Examples are used throughout the paper to clarify important concepts, and Section IV shows simulation experiments for the networked control of a nonlinear robot arm. Section V concludes the paper.

II. Background

A. Optimal control problem

Consider an optimal control problem for a deterministic, discrete-time nonlinear system

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

with state $x \in X$ and action $u \in U$, where X and U are arbitrary, possibly multidimensional spaces (although we restrict U to a finite set below). Each transition from x_k to x_{k+1} as a result of u_k is associated with a reward $r_{k+1} = \rho(x_k, u_k)$, and the goal is to find for each state x a sequence of actions $\boldsymbol{u}_\infty = (u_0, u_1, \dots) \in U^\infty$ that maximizes the infinite-horizon discounted return (also called the *value*)

$$V^{\boldsymbol{u}_\infty}(x) = \sum_{k=0}^{\infty} \gamma^k r_{k+1} = \sum_{k=0}^{\infty} \gamma^k \rho(x_k, u_k) \tag{2}$$

where $x_0 = x, x_{k+1} = f(x_k, u_k)$ for $k \geq 0$. The discount factor $\gamma \in [0, 1)$ ensures the boundedness of the return, see [15] for another method that uses discounting in NCS. Elements X, U, f, ρ, and γ together form a deterministic type of *Markov decision process* (MDP). The optimal (maximal) value function, defined as[1] $V^*(x) = \sup_{\boldsymbol{u}_\infty} V^{\boldsymbol{u}_\infty}(x)$, always exists and is unique [27]. Note that this optimal control formulation is nonstandard, since typically state feedback control policies $h(x)$ are used. While state feedback policies are sufficient to achieve the optimal values, the action-sequence formulation helps to more easily understand our approach, and does not lose generality.

Assumption 1: The action space is discrete (or discretized), $U = \{u^1, \dots, u^M\}$. Rewards are bounded in $[0, 1]$.

To ensure bounded rewards in practice, a first solution is to directly saturate the reward function. This changes the optimal solution, but may be sufficient if the changes do not affect interesting state regions. On the other hand, the physical limitations of the system may be meaningfully modeled by saturating the states in the dynamics. If the limits are known a reward bound follows. Finally, in most cases a bounded reward function can be normalized to $[0, 1]$ without changing the optimal solution.[2] Next, we exemplify the framework with a classical LQR problem.

Example 1: Discounted LQR. Consider the problem of optimally stabilizing a DC motor. Discretizing in time a first-principles model with the zero-order-hold method and $T_\mathrm{s} = 0.01\,\mathrm{s}$, we obtain the dynamics

$$f(x, u) = Ax + Bu,\ A \approx \begin{bmatrix} 1 & 0.0095 \\ 0 & 0.9100 \end{bmatrix},\ B \approx \begin{bmatrix} 0.0084 \\ 1.6618 \end{bmatrix}$$

where $x_1 = \alpha$ is the shaft angle, $x_2 = \dot{\alpha}$ the angular velocity, and u the voltage. The goal is stabilizing the system around $x = 0$, and is described by the reward function:

$$\rho(x, u) = -x^\top Q x - u^\top R u,\ Q = \mathrm{diag}(5, 0.001),\ R = 0.01$$

with discount factor $\gamma = 0.9$. Because the dynamics are linear and the rewards are quadratic, the optimal state feedback policy is linear[3] $h(x) = L^\top x$ where $L \approx [-5.60, -0.28]^\top$. Figure 1, left shows a trajectory controlled with this policy.

We investigate the effects of the required assumptions. First, as discussed above, the states and actions are restricted using saturation to $\alpha \in [-\pi, \pi]\,\mathrm{rad}$, $\dot{\alpha} \in [-15\pi, 15\pi]\,\mathrm{rad/s}$, $u \in [-30, 30]\,\mathrm{V}$. Note that saturation effectively makes the dynamics nonlinear, but this nonlinearity does not manifest itself over the variable ranges in the figures we exemplify. The rewards can then easily be rescaled into $[0, 1]$. The actions are discretized into the set $U = \{-10, -3, 0, 3, 10\}$. Figure 1, right shows a trajectory controlled by a policy that is near-optimal for this discretization. Of course, discretization reduces performance (e.g. it introduces steady-state error), but not greatly, which can also be seen in the small difference between the returns V obtained along the two trajectories: 9.082 with continuous actions, 9.067 with discrete actions.[4] □

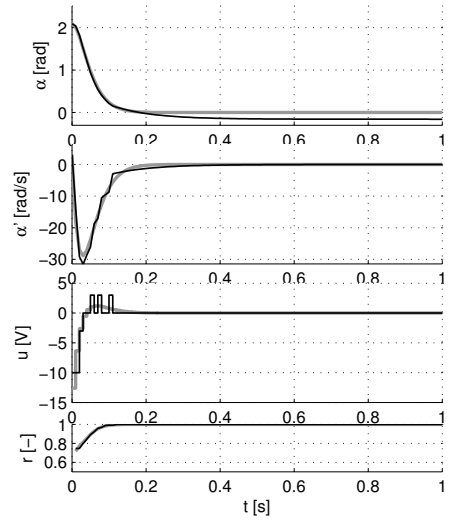

Fig. 1. Controlled trajectories from $x_0 = [2\pi/3, \pi]^\top$, with continuous actions (gray) and discretized actions (black).

[1] In optimal control, often a cost J is minimized rather than a *return* or *value* V being maximized. The two formulations are equivalent.

[2] One exception are tasks with absorbing states, which once reached cannot be escaped and always provide zero rewards, for any action. Such states can be used to represent e.g. "goal achieved" and "failure" situations.

[3] The gains are computed with an extension of the LQR solution to the discounted case, $L = -\gamma(\gamma B^\top Y B + R)^{-1} B^\top Y A$, where Y is the stabilizing solution of the Riccati equation: $Y = A^\top [\gamma Y - \gamma^2 Y B (\gamma B^\top Y B + R)^{-1} B^\top] A + Q$, see [27], Ch. 3.

[4] The discrete-action solution is computed with approximate value iteration using a fine state discretization, specifically fuzzy Q-iteration [28], a consistent algorithm i.e. one that obtains small errors for fine discretization. The consistency proof in [28] also gives more insight into how solution quality relates to the state and action discretization resolutions.

B. Optimistic planning

To introduce the algorithm, in this section we focus on a particular state x where it must be applied, and by convention set the current time to 0, so that $x_0 = x$. Of course, the procedure works at any time step.

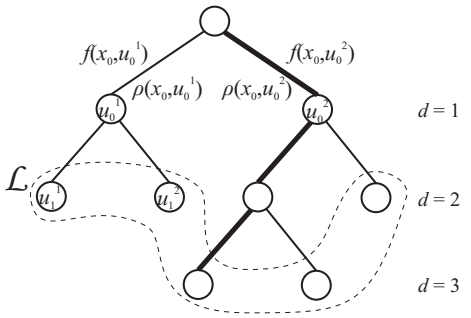

Fig. 2. Illustration of an OP tree $\mathcal{T}$. Nodes are labeled by actions, arcs represent transitions and are labeled by the rewards and next states resulting by applying the corresponding action. Subscripts are depths, superscripts index the M possible actions/transitions from a node (here, $M = 2$). The leaves $\mathcal{L}$ are enclosed in a dashed line, while the thick path highlights an action sequence.

Optimistic planning (OP) [19] explores a tree representation of the possible action sequences from the current state, as illustrated in Figure 2. OP starts with an unlabeled root node, and iteratively expands T nodes. Expanding a node adds new children nodes corresponding to all the M actions $u^1, \dots, u^M$. Each node at some depth d is reached via a unique path through the tree, and can thus be uniquely associated to the sequence of actions $\boldsymbol{u}_d = (u_0, u_1, \dots, u_{d-1})$ on this path. In what follows, we will work interchangeably with sequences and paths, keeping this equivalence in mind.

For a sequence $\boldsymbol{u}_d$, we define three quantities:

$$\ell_x(\boldsymbol{u}_d) = \sum_{d'=0}^{d-1} \gamma^{d'} \rho(x_{d'}, u_{d'}), \quad b_x(\boldsymbol{u}_d) = \ell_x(\boldsymbol{u}_d) + \frac{\gamma^d}{1-\gamma}$$
$$v_x(\boldsymbol{u}_d) = \ell_x(\boldsymbol{u}_d) + \gamma^d V^*(x_d) \tag{3}$$

where the states are generated with the action sequence $\boldsymbol{u}_d$, like in (2). The quantity $\ell_x(\boldsymbol{u}_d)$ provides a lower bound on the value of any infinite sequence that starts with $\boldsymbol{u}_d$, while $b_x(\boldsymbol{u}_d)$ is an upper bound (this is because all the rewards at depths larger than d are in $[0, 1]$). The value $v_x(\boldsymbol{u}_d)$ is obtained by continuing optimally after $\boldsymbol{u}_d$. Subscript x indicates that the three quantities depend on the state $x = x_0$ where OP is applied.

We denote the set of sequences corresponding to leaves by $\mathcal{L}$. OP *optimistically* explores the space of action sequences, by always expanding further a most promising leaf sequence: one with the largest upper bound $b_x(\boldsymbol{u})$. After a certain number of expansions, a sequence that maximizes the lower bound $\ell_x(\boldsymbol{u})$ among the leaves is returned, intuitively seen as a safe choice. Algorithm 1 summarizes the entire procedure, where the function $\Delta(\cdot)$ gives the depth of a tree. We allow the algorithm to terminate either after a given number of expansions, or after a given depth has been explored (i.e. at least a node has been expanded at that depth). The b_x and ℓ_x values can be efficiently maintained on the tree. Note that OP is related to the classical A* search algorithm, and can also be seen as a branch-and-bound optimization over sequences.

Algorithm 1 Optimistic planning for deterministic systems

Input: state x, budget n or depth d (set the other to ∞)
1: initialize tree: $\mathcal{T} \leftarrow \{\text{root}\}$, $i = 0$
2: **repeat**
3: find optimistic sequence: $\boldsymbol{u}^\dagger \in \arg\max_{\boldsymbol{u} \in \mathcal{L}} b_x(\boldsymbol{u})$
4: add children $u_j, j = 1, \dots, M$ to the node of $\boldsymbol{u}^\dagger$
5: $i \leftarrow i + 1$
6: **until** $i = n$ or $\Delta(\mathcal{T}) = d + 1$
7: $n \leftarrow i$; $d \leftarrow \Delta(\mathcal{T}) - 1$
Output: $\boldsymbol{u}^* \in \arg\max_{\boldsymbol{u} \in \mathcal{L}} \ell_x(\boldsymbol{u})$

C. Theoretical guarantees

To characterize the complexity of finding the optimal sequence from a given state x, we define $\kappa(x)$ as the asymptotic branching factor of the near-optimal subtree: $\mathcal{T}^*(x) = \left\{ \boldsymbol{u}_d \;\middle|\; d \geq 0, V^*(x) - v_x(\boldsymbol{u}_d) \leq \gamma^d/(1-\gamma) \right\}$. Intuitively, $\mathcal{T}^*(x)$ contains the sequences for which it is impossible to tell, from their rewards up to d, whether or not they are part of an optimal solution, because their near-optimality is smaller than the amount of reward $\gamma^d/(1-\gamma)$ they might accumulate below depth d. In general, we say that a sequence $\boldsymbol{u}_d$ is ε-optimal when $V^*(x) - v_x(\boldsymbol{u}_d) \leq \varepsilon$.

We provide the guarantees about OP in a form that brings out its useful properties in NCS. Intuitively, part (i) of the upcoming theorem shows that OP returns a long and near-optimal sequence, and parts (ii), (iii) show that sequence length and near-optimality are closely related to the computation budget, via the branching factor $\kappa(x)$. These results have either been obtained by [19] or are simple extensions of the results proven there.

Theorem 2: OP has the following properties:[5]

(i) After developing a tree $\mathcal{T}$, OP returns a sequence $\boldsymbol{u}_d$ of length[6] $d = \Delta(\mathcal{T}) - 1$, which is also $\frac{\gamma^d}{1-\gamma}$-optimal.
(ii) When OP is called in x with a large enough d: • If $\kappa(x) > 1$ it will require a number of expansions $n(x) = O(\kappa(x)^d)$. • If $\kappa(x) = 1$, $n(x) = O(d)$.
(iii) When OP is called in x with a large enough n: • If $\kappa(x) > 1$ it will reach a depth of $d(x) = \Omega(\frac{\log n}{\log \kappa(x)})$, and $\varepsilon(x) = O(n^{-\frac{\log 1/\gamma}{\log \kappa(x)}})$. • If $\kappa(x) = 1$, $d(x) = \Omega(n)$ and $\varepsilon(x) = O(\gamma^{c(x)n})$, where $c(x)$ is a constant. □

Since $\kappa(x)$ is generally unknown, the computation requirements or near-optimality of OP cannot be determined in advance. However, Theorem 2 provides confidence that the algorithm automatically adapts to the complexity of the problem, described by $\kappa(x)$. The smaller $\kappa(x)$, the more easily near-optimal sequences can be distinguished, and the

[5]The notations $g = O(h)$ and $g = \Omega(h)$ mean that g asymptotically grows, respectively, at most as fast / at least as fast as h. Note also that $d(x) = \Omega(\log n / \log \kappa(x))$ will be used to emphasize the link of $d(x)$ with $\kappa(x)$, even though $\Omega(\log n / \log \kappa(x))$ is equivalent to $\Omega(\log n)$.

[6]OP may also return a sequence of maximal length $\Delta(\mathcal{T})$; to maintain uniformity of notation, in that case the last action is removed, although this is not necessary in practice.

better OP does. In particular, the best case is $\kappa(x) = 1$, obtained e.g. when a single sequence always obtains rewards of 1, and all the other rewards on the tree are 0. In this case the algorithm must only develop this sequence. In the worst case, $\kappa(x) = M$ (the number of discrete actions); this happens when all the sequences have the same value, in which case the algorithm must explore the complete tree uniformly, expanding nodes in order of their depth.

III. OP FOR NETWORKED CONTROL SYSTEMS

A. Setting

In this paper we consider a networked-control setting, in which actuation and state signals are exchanged over a network that must be efficiently utilized. To this end, the controller should only communicate with the plant when needed. OP is well equipped to handle this case, since it guarantees that it will return *long sequences* of actions that have *near-optimal* performance.

We envision the following setup. The sequence of transmission instants is denoted by k_i, $i \in \{0, 1, 2, \ldots\}$, and it will either be fixed by the user or defined by the controller itself. At each k_i, the controller receives the state's measurement and generates a sequence of control actions which is sent as a single packet to the actuators' buffer, like in [29]. The actuators then apply the k'-th component of the sequence to the plant at step $k_i + k'$, until the full sequence has been used. Afterwards, the new state's measurement is sent to the controller and the procedure is repeated. In this way, the communication cost is reduced, since the channel is only used at intervals equal to the lengths of the action sequences.

B. Algorithms

Algorithm 1 and Theorem 2 suggest two ways in which OP could be exploited for NCS. The first possibility is to impose a desired planning depth d at every controller execution step, and then send to the plant either the full sequence or an initial subsequence thereof. Denoting the length of the sent (sub)sequence by $d' \leq d$, this means the communication between the controller and the plant is set to occur at fixed period d', i.e. the transmission instants k_i are multiples of d'. Applying OP in this way is novel. Since the controller execution interval is fixed, this first strategy is called Clock-triggered OP (COP); it is summarized in Algorithm 2.

Algorithm 2 COP: Clock-triggered optimistic planning

Input: initial state x_0, target depth d, subsequence length d'

$k \leftarrow 0$
loop
  measure current state x_k
  apply OP(x_k, d), obtaining a sequence $\boldsymbol{u}_d$
  send initial subsequence $\boldsymbol{u}_{d'}$ to plant
  $k \leftarrow k + d'$, wait d' steps
end loop

The second possibility is to impose the computation budget n, like in the classical application of OP, and let the algorithm find the longest sequence it can within this budget. Then, different from classical OP which sends just one action, we send again either the whole sequence or a subsequence. Since the returned sequence length, and so the communication interval, will vary depending on the planning complexity at the current state, the algorithm is called Self-Triggered OP (STOP); it is summarized as Algorithm 3. To allow sending subsequences, the algorithm is parameterized by the fraction $\alpha \in (0, 1]$, so that if a sequence of length d is returned by OP, only the first $\lceil \alpha d \rceil$ actions are actually sent and applied. Here, $\lceil \cdot \rceil$ denotes the ceiling operator.

Algorithm 3 STOP: Self-triggered optimistic planning

Input: initial state x_0, budget n, subsequence fraction α

$k \leftarrow 0$
loop
  measure current state x_k
  apply OP(x_k, n), obtaining a sequence $\boldsymbol{u}_{d(x)}$
  send initial subsequence $\boldsymbol{u}_{\lceil \alpha d(x) \rceil}$ to plant
  $k \leftarrow k + \lceil \alpha d(x) \rceil$, wait $\lceil \alpha d(x) \rceil$ steps
end loop

It should be emphasized that the upcoming analysis of these methods is performed under the assumption that the model is correct. Of course, in practice model errors or disturbances appear, which means the sequences cannot be too long and the loop must be closed fairly often. Even assuming correct models, some nontrivial relationships arise between the performance of shorter and longer sequences, as detailed in the next section.

C. Analysis

We first consider the near-optimality and complexity of COP.

Theorem 3: COP is $\frac{\gamma^d}{1-\gamma}$-optimal. Furthermore, at every state x where it is called, COP requires: • $n = O(\kappa(x)^d)$ computation if $\kappa(x) > 1$; • $n = O(d)$ computation if $\kappa(x) = 1$ (with $\kappa(x)$ the branching factor from Section II-C). □

Thus, the quality of the solution grows with the imposed sequence length d, and the computation requirements to reach this length are bounded and characterized using $\kappa(x)$. Specifically, computation grows exponentially in d, with base $\kappa(x)$ – unless $\kappa(x) = 1$, in which case it grows linearly in d. Next, we move on to STOP.

Theorem 4: For large computational budget n, the near-optimality of STOP is: • $O(n^{-\frac{\log 1/\gamma}{\log \kappa(x_0)}})$ if $\kappa(x_0) > 1$, and • $O(\gamma^{c(x_0)n})$ if $\kappa(x_0) = 1$. Furthermore, at every state x where it is called, STOP finds a sequence of length: • $d(x) = \Omega(\frac{\log n}{\log \kappa(x)})$ if $\kappa(x) > 1$, and • $d(x) = \Omega(n)$ if $\kappa(x) = 1$. □

In this case, the performance guarantee depends only on the planning difficulty at the initial state x_0: it is polynomial in n when $\kappa(x_0) > 1$, and exponential (better) when it is $\kappa(x_0) = 1$. The sequence length grows fast, in a way that is characterized using $\kappa(x)$, and which basically 'inverts' the relationship between computation and length in COP.

It is essential to note that the subsequence length (represented by d' in COP and α in STOP) does not affect the near-optimality guarantee: there is no loss, whether the loop is closed sooner or later. This is the main novelty in

Theorems 3 and 4, while the other results follow easily from Theorem 2. Of course, this does not mean that the same performance is obtained. In fact, applying shorter sequences may achieve better or worse performance, depending on the problem. The following result characterizes this behavior, in a general way that applies to both COP and STOP.

Theorem 5: Consider OP returns a sequence $\boldsymbol{u}_d$. Define the sequence $(\boldsymbol{u}_{d'}, \boldsymbol{u}_{d_1})$, formed by joining a subsequence $\boldsymbol{u}_{d'}$ of $\boldsymbol{u}_d$, with the new sequence $\boldsymbol{u}_{d_1}$ obtained by replanning after $\boldsymbol{u}_{d'}$ (see Figure 3). Define similarly $(\boldsymbol{u}_{d''}, \boldsymbol{u}_{d_2})$, where $d'' > d'$. Recall the value v of a sequence is the return obtained by applying it and then acting optimally. Then:

$$v(\boldsymbol{u}_{d'}, \boldsymbol{u}_{d_1}) \geq v(\boldsymbol{u}_{d''}, \boldsymbol{u}_{d_2}) - \frac{\gamma^{d'+d_1}}{1-\gamma}$$

Furthermore, if OP runs with the same budget or target depth to compute $\boldsymbol{u}_d$, $\boldsymbol{u}_{d_1}$, and $\boldsymbol{u}_{d_2}$, then the bound is tight: there exist problems where it holds with equality. □

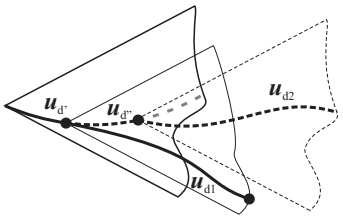

Fig. 3. Shorter versus longer subsequences.

The theorem says that applying a shorter sequence and then replanning may lose some performance, but not too much: the maximum loss is given by the accuracy of the *entire composite sequence* $(\boldsymbol{u}_{d'}, \boldsymbol{u}_{d_1})$, i.e., $\frac{\gamma^{d'+d_1}}{1-\gamma}$. This result is counterintuitive – one might expect shorter sequences to be strictly better, since they allow reconsidering action choices sooner. Nevertheless, since the bound is tight, nothing stronger can be said in general. The following examples provide more insight into this issue, using COP as it allows to directly control the (sub)sequence length.

Example 2: Shorter sequences can perform better. Consider an MDP with state space $\{1, 2, \ldots, 5\}$, two actions $-1, 1$ ("left" and "right"), and additive dynamics $x_+ = \max(1, \min(5, x+u))$. The rewards obtained upon reaching each of the five states are, respectively, $0.8, 0.7, 0.5, 0.8, 0$, and the discount factor is 0.8, see Figure 4.

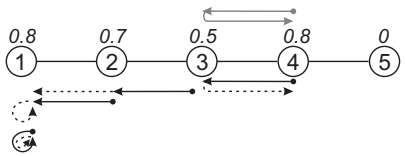

Fig. 4. A five-state MDP and two COP solutions. States are shown in circles, and rewards in italics above them. The solution from $x_0 = 4$ with $d' = d = 2$ is shown in gray on top of the figure, while the one for $d' = 1$, $d = 2$ is shown in black on the bottom. Solutions are shown as sequences of actions, where the bullets mark the states in which planning is run, and unapplied sequence tails are shown in dashed lines.

When applied from $x_0 = 4$ with $d' = d = 2$, COP exploits the rewards of states 4 and 5 and cycles between these states forever, obtaining a return of 3.17. When d' is decreased to 1 however, the algorithm has a chance to replan in $x_1 = 3$, and with the same horizon $d = 2$ this allows it to detect the larger rewards to the left. It eventually reaches state 1 and remains there, achieving the optimal return of 3.62. □

Example 3: Longer sequences can perform better. A similar MDP is taken but now with states $\{1, 2, \ldots, 7\}$ and the rewards shown in Figure 5. The discount factor is the same.

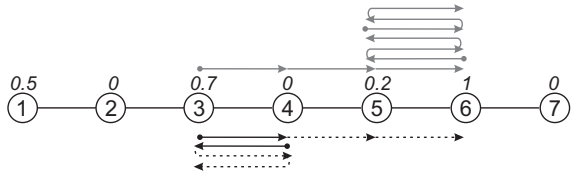

Fig. 5. A seven-state MDP and two COP solutions, for $d' = d = 3$ (top, gray) and $d' = 1$, $d = 3$ (bottom, black).

Now, when applied from $x_0 = 3$ with $d' = d = 3$, COP discovers the large reward in state 6 and controls towards this state, cycling afterwards between 5 and 6 for a return of 2.22. When $d' = 1$ however, replanning from state 4 misleads the algorithm into a shorter-horizon cycle that focuses on the reward 0.7, achieving only a suboptimal return of 1.56. □

IV. Results for a nonlinear robot arm

To illustrate our approach, we apply STOP to stabilize a nonlinear, two-link robot arm operating in a horizontal plane. The state variables are the angles and angular velocities of the two links, $x_i = [\theta_{i,1}, \dot{\theta}_{i,1}, \theta_{i,2}, \dot{\theta}_{i,2}]$, and the actions are the torques of the motors actuating the links: $u_i = [\tau_{i,1}, \tau_{i,2}]$. The model is standard so we omit the details and just note that the sampling time is $T_\mathrm{s} = 0.05\,\mathrm{s}$, and that time discretization is performed with the fourth-order Runge-Kutta method; the other parameters can be found in [28]. The goal of stabilizing in the zero state is modeled by a quadratic reward similar to that in Example 1, with weights $Q = \mathrm{diag}[1, 0.05, 1, 0.05]$ and $R = \mathrm{diag}[0.01, 0.01]$. The discount factor is $\gamma = 0.95$, and the discretized action set is $\{-1.5, 0, 1.5\} \times \{-1, 0, 1\}$. STOP is applied with $n = 1000$ and $\alpha = 0.4$ from the initial state $x_0 = [\pi, 0, \pi, 0]^\top$, and the results are shown in Figure 6. A good performance is achieved – subject to the limitations of the action discretization, due to which adjustments have to be made close to the equilibrium state. Note the variation of the sequence lengths with the complexity of the planning problem at different states: in particular, at states further away from the equilibrium sequences are shorter due to higher complexity, whereas at states closer to the equilibrium the complexity decreases and the sequences are longer.

V. Conclusions

Two novel methods have been introduced for the optimal networked control of nonlinear systems. They both rely on properties of optimistic planning (OP) and guarantee near-optimal performance. Clock-triggered OP repeatedly sends action sequences of a fixed length, and bounds the

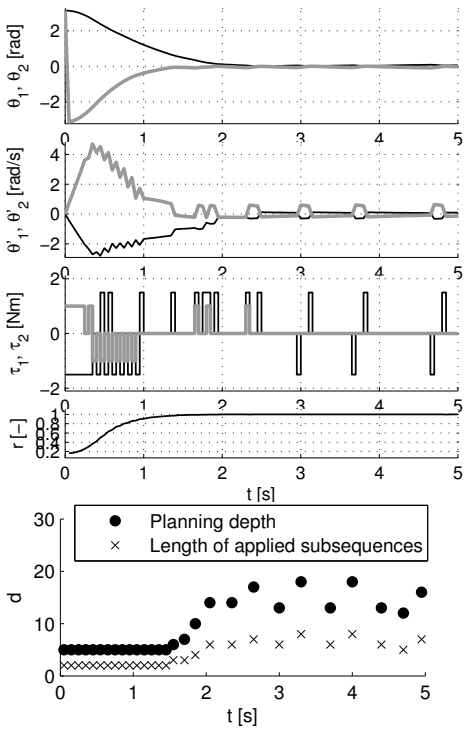

Fig. 6. Results of STOP for robot arm stabilization: trajectory (top), planning depths and lengths of the applied subsequences (bottom).

computation required to find them. Self-triggered OP works with fixed computation, and adapts the sequence length – and thus the communication interval – to the current state, guaranteeing long sequences. Interestingly, applying shorter subsequences instead of the full sequences may work better or worse, depending on the problem.

The most important topic for future work is reducing the dependence on a fully known, deterministic model. The first step towards this goal will be to exploit planning algorithms for stochastic systems, in order to handle certain classes of random disturbances. Furthermore, similar ideas to those presented here can be used to apply continuous-action planning to networked control systems.

REFERENCES

[1] M. Branicky, S. Phillips, and W. Zhang, "Scheduling and feedback co-design for networked control systems," in *CDC (IEEE Conference on Decision and Control) Las Vegas, U.S.A.*, 2002, pp. 1211–1217.

[2] J. Hespanha, P. Naghshtabrizi, and Y. Xu, "A survey of recent results in networked control systems," *IEEE Special Issue on Technology of Networked Control Systems*, vol. 95, no. 1, pp. 138–162, 2007.

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

[4] W. Heemels, J. Sandee, and P. van den Bosch, "Analysis of event-driven controllers for linear systems," *International Journal of Control*, vol. 81, no. 4, pp. 571–590, 2009.

[5] A. Anta and P. Tabuada, "To sample or not to sample: self-triggered control for nonlinear systems," *IEEE Transactions on Automatic Control*, vol. 55, no. 9, pp. 2030–2042, 2010.

[6] X. Wang and M. Lemmon, "Self-triggered feedback control systems with finite-gain $\mathcal{L}_2$ stability," *IEEE Transactions on Automatic Control*, vol. 45, pp. 452–467, 2009.

[7] T. Henningsson, E. Johannesson, and A. Cervin, "Sporadic event-based control of first-order linear stochastic systems," *Automatica*, vol. 44, pp. 2890–2895, 2008.

[8] M. Velasco, J. Fuertes, and P. Marti, "The self triggered task model for real-time control systems," *24th IEEE Real-Time Systems Symposium*, pp. 67–70, 2003.

[9] A. Molin and S. Hirche, "On LQG joint optimal scheduling and control under communication constraints," in *CDC (IEEE Conference on Decision and Control) Shangai: China*, 2009.

[10] T. Henningsson and A. Cervin, "Scheduling of event-triggered controllers on a shared network," in *CDC (IEEE Conference on Decision and Control) Cancun, Mexico*, 2008.

[11] R. Blind and F. Allgöwer, "On the optimal sending rate for networked control systems with a shared communication medium," in *CDC / ECC (IEEE Conference on Decision and Control and European Control Conference) Orlando, U.S.A., Orlando: U.S.A.*, 2011.

[12] ——, "Analysis of Networked Event-Based Control with a Shared Communication Medium: Part I - Pure ALOHA," in *14^{th} IFAC World Congress, Milan, Italy*, 2011.

[13] A. Molin and S. Hirche, "On the optimal design of decentralized event-triggered controllers for large-scale systems with contention-based communication," in *CDC / ECC (IEEE Conference on Decision and Control and European Control Conference) Orlando, U.S.A., Orlando: U.S.A.*, 2011.

[14] M. Rabi, K. Johansson, and M. Johansson, "Optimal stopping event-triggered sensing and actuation," in *CDC (IEEE Conference on Decision and Control) Cancun, Mexico*, 2008, pp. 3607–3612.

[15] D. Antunes, W. Heemels, and P. Tabuada, "Dynamic programming formulation of periodic event-triggered control: Performance guarantees and co-design," in *IEEE Conference on Decision and Control, Hawai: U.S.A.*, 2012, pp. 7212–7217.

[16] J. B. Berglind, T. Gommans, and W. Heemels, "Self-triggered MPC for constrained linear systems and quadratic costs," in *IFAC Nonlinear Model Predictive Control Conference, Noordwijkerhout: The Netherlands*, 2012, pp. 342–348.

[17] E. Henriksson, D. Quevedo, H. Sandberg, and K. Johansson, "Self-triggered model predictive control for network scheduling and control," in *IFAC Symposium on Advanced Control of Chemical Processes, Singapore*, 2012, pp. 432–438.

[18] A. Eqtami, D. Dimarogonas, and K. Kyriakopoulos, "Novel event-triggered strategies for model predictive controllers," in *IEEE Conference on Decision and Control and European Control Conference (CDC-ECC), Orlando: U.S.A.*, 2011, pp. 3392–3397.

[19] J.-F. Hren and R. Munos, "Optimistic planning of deterministic systems," in *Proceedings 8th European Workshop on Reinforcement Learning (EWRL-08)*, Villeneuve d'Ascq, France, 30 June – 3 July 2008, pp. 151–164.

[20] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine Learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

[21] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *Proceedings 17th European Conference on Machine Learning (ECML-06)*, Berlin, Germany, 18–22 September 2006, pp. 282–293.

[22] S. Bubeck and R. Munos, "Open loop optimistic planning," in *Proceedings 23rd Annual Conference on Learning Theory (COLT-10)*, Haifa, Israel, 27–29 June 2010, pp. 477–489.

[23] L. Buşoniu, R. Munos, B. De Schutter, and R. Babuška, "Optimistic planning for sparsely stochastic systems," in *Proceedings 2011 IEEE International Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL-11)*, Paris, France, 11–15 April 2011, pp. 48–55.

[24] C. Mansley, A. Weinstein, and M. L. Littman, "Sample-based planning for continuous action Markov decision processes," in *Proceedings 21st International Conference on Automated Planning and Scheduling*, Freiburg, Germany, 11–16 June 2011, pp. 335–338.

[25] S. Gelly, Y. Wang, R. Munos, and O. Teytaud, "Modification of UCT with patterns in Monte-Carlo Go," INRIA, Tech. Rep., 2006.

[26] C. De Persis and P. Frasca, "Robust self-triggered coordination with ternary controllers," *arXiv: 1205.6917*, 2012.

[27] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, 3rd ed. Athena Scientific, 2007, vol. 2.

[28] L. Buşoniu, D. Ernst, B. De Schutter, and R. Babuška, "Approximate dynamic programming with a fuzzy parameterization," *Automatica*, vol. 46, no. 5, pp. 804–814, 2010.

[29] A. Chaillet and A. Bicchi, "Delay compensation in packet-switching networked controlled systems," in *CDC (IEEE Conference on Decision and Control), Cancun, Mexico*, 2008, pp. 3620–3625.