Optimistic planning for near-optimal control of nonlinear systems with hybrid inputs

Ioana Lal, Constantin Morărescu, Jamal Daafouz, Lucian Buşoniu

***Abstract*— We propose an optimistic planning, branch-and-bound algorithm for nonlinear optimal control problems in which there is a continuous and a discrete action (input). The dynamics and rewards (negative costs) must be Lipschitz but can otherwise be general, as long as certain boundedness conditions are satisfied by the continuous action, reward, and Lipschitz constant of the dynamics. We investigate the structure of the space of hybrid-input sequences, and based on this structure we propose an optimistic selection rule for the subset with the largest upper bound on the value, and a way to select the largest-impact action for further refinement. Together, these fully define the algorithm, which we call OPHIS: optimistic planning for hybrid-input systems. A near-optimality bound is provided together with empirical results in two nonlinear problems where the algorithm is applied in receding horizon.**

I. Introduction

We consider optimal control of nonlinear systems in which the inputs (control actions) consist of a continuous component and a discrete one; we refer to such systems as hybrid-input. These systems can be encountered in several fields, such as robotics [2], [7], industrial plants for multiple tanks systems [9], [12], [13], hydraulic systems [8] or the automotive industry, for joint control of engine power and the transmission gear [15], [6]. A number of techniques were used for solving this type of applications, such as Branch and Bound [2], switching control [7], or MPC [13], [8], [9], [12]. Compared to these methods, our solution can handle problems with more general dynamics and cost functions, while focusing on infinite-horizon optimal control, rather than finite-horizon. In addition, a near-optimality bound is provided, along with a fully-specified, directly implementable algorithm (which is not always the case with some proposed approaches).

Specifically, we consider hybrid-input systems for which the dynamics can be generally nonlinear and the cost functions arbitrary, as long as they are Lipschitz with respect to the state and continuous action. For such systems, we propose a method called OPHIS: Optimistic Planning for Hybrid-Input Systems, which like MPC produces at each given state an open-loop sequence, and is meant to be applied in receding horizon. The continuous action must be scalar, a restriction that can be relaxed at extra computational cost. OPHIS creates a partition of the set of hybrid-input sequences iteratively, by choosing for refinement at each iteration an optimistic set that maximizes an upper bound on the value. For this set, a dimension is chosen for refinement, together with the type of split (continuous or discrete).

We provide a near-optimality bound for the algorithm, which can be computed once it has terminated. We exemplify OPHIS in simulations for two systems. For a relatively simple two-tanks system, detailed simulations are given that study the effect of tuning parameters. Then, a more complex problem is shown, a two-link robot arm with one link actuated and the second that can only be influenced by a holding brake. In both cases, the algorithm succeeds in controlling the system.

OPHIS essentially combines optimistic planning for deterministic, discrete-input systems (OPD) [5], [11] and OP for continuous-input systems (OPC) [1]. The key technical challenge is that the structure of the hybrid space of solutions is significantly more complicated than either for OPC or OPD, and consequently so are the refinement rules that we must come up with to explore it in the novel algorithm. OPHIS in fact specializes to OPD when the continuous action is removed, and to OPC when the discrete action is removed.

Compared to nonlinear MPC, which typically focuses on finite-horizon problems, here the objective is infinite-horizon. For instance, in [12], the horizon used in simulations is firstly 1, and then 3. Moreover, algorithmically, MPC methods usually rely on derivatives, while our method only requires Lipschitz continuity and thus, at the cost of more computation, is more general with respect to dynamics and cost functions. Some MPC versions require the linearization of the model around several operating points [9], [8], while OPHIS uses the nonlinear model directly. Branch and bound is used in [2], in combination with sparse direct collocation. However, no near-optimality analysis is provided for this solution.

Another key difference between usual hybrid-input control approaches and OPHIS is that the former primarily concentrate on stability, such as in [7], where a switching control strategy is employed, whereas here we aim for near-optimality. In effect, by using discounting and imposing a joint condition on the discount factor and Lipschitz constant of the dynamics, the system dynamics are instead *required* to satisfy a certain contractiveness property. Promising *guarantees* of stability have been obtained both for the exactly optimal solution of general discounted optimal control [11] and for discrete-input optimistic planning [4]. However, the behavior of OPHIS (and of OPC) when refining continuous

I. Lal and L. Buşoniu are with the Automation Department, Technical University of Cluj-Napoca, Romania. C. Morărescu and J. Daafouz are with Université de Lorraine, CRAN, UMR 7039 and CNRS, CRAN, UMR 7039, Nancy, France. Email addresses: ioanalal04@gmail.com, constantin.morarescu@univ-lorraine.fr, jamal.daafouz@univ-lorraine.fr, lucian@busoniu.net.
This work was been financially supported from by the Romanian National Authority for Scientific Research, CNCS-UEFISCDI, SeaClear support project number PN-III-P3-3.6-H2020-2020-0060.

actions is much more intricate, and analyzing its stability falls outside the scope of this paper.

Next, Section II formalizes the problem and Section III discusses the background on OPC and OPD. OPHIS is described in Section IV, while Section V provides the near-optimality bound. Finally, Section VI considers the two applications with the corresponding simulation results, and VII gives the conclusions of this paper.

II. Problem statement

We consider an optimal control problem for a hybrid-input, nonlinear system $x_{k+1} = f(x_k, u_k)$, with $x \in X \subseteq \mathbb{R}^m$ and $u \in U$ consisting of a continuous action and a discrete one:

$$u_k = \begin{bmatrix} c_k & d_k \end{bmatrix}^T \tag{1}$$

where $c_k \in \mathbb{R}$ and $d_k \in \{0, 1, ..., p\}$, $p \in \mathbb{N}$. We define a reward function $\rho : X \times U \to \mathbb{R}$, that takes as input a state-action pair (x_k, u_k): $r_{k+1} = \rho(x_k, u_k)$. Starting from an initial state x_0, we define an infinitely-long sequence of actions $\mathbf{u}_\infty = (u_0, u_1, ...)$ and its infinite-horizon discounted value:

$$v(\mathbf{u}_\infty) = \sum_{k=0}^{\infty} \gamma^k \rho(x_k, u_k) \tag{2}$$

where $\gamma \in (0, 1)$ is the discount factor (note that $\gamma = 1$ is excluded). We aim in principle to find the optimal value $v^* := sup_{\mathbf{u}_\infty} v(\mathbf{u}_\infty)$ and a sequence that achieves it.

We make the following assumptions:

Assumption 1. *We have (i) $r \in [0, 1]$ and (ii) $c \in [0, 1]$.*

Together with discounting, the bounded rewards ensure boundedness of the sequence values, a necessary condition for our planning algorithms and the analysis. The bounded continuous action is needed because we will numerically refine that interval. Note that now $U = ([0, 1] \times \{0, 1, ..., p\})$. For both the reward and the continuous action, the unit interval is used only for convenience and can be easily obtained by scaling other intervals. The restriction to scalar continuous actions can be relaxed, but at significant extra computational cost for the extended algorithm; see [1], supplementary material for such an extension in OPC.

Bounded costs are typical in AI methods for optimal control, such as reinforcement learning [14]. Boundedness could e.g. follow from physical limitations in the system, or could otherwise be achieved by saturating an a priori unbounded reward function, which changes the optimal solution but is often sufficient.

Assumption 2. *(i) Both the dynamics and the rewards are Lipschitz with respect to the state and the continuous action, i.e., $\exists L_f, L_p$ s.t. $\forall x, x' \in X$ and $c, c' \in [0, 1]$:*

$$\begin{aligned} \|f(x, [c, d]^T) - f(x', [c', d]^T)\| &\leq L_f(\|x - x'\| + |c - c'|) \\ |\rho(x, [c, d]^T) - \rho(x', [c', d]^T)| &\leq L_\rho(\|x - x'\| + |c - c'|) \end{aligned} \tag{3}$$

(ii) The following inequality is satisfied:

$$\gamma L_f < 1 \tag{4}$$

It should be noted that Lipschitz continuity is only imposed w.r.t. the continuous component c of the action; whereas the variation w.r.t. d can be arbitrary. This is a useful feature because switches often induce discontinuities in the system. Furthermore, whereas typical derivative-based MPC techniques [10] require differentiability, our condition (i) allows the dynamics and rewards to be nondifferentiable as long as they are still Lipschitz. This helps to model effects like saturations, actuator deadzones, etc. Condition (ii) means that the dynamics need not be strictly contractive on their own, but should become so when combined with a shrink rate equal to the discount factor γ. In principle, this condition is not required, and the algorithm will work without it; however, without this condition the contribution of dimensions k in (5) can become unbounded as h grows, which in practice would mean that the algorithm never finds a long-horizon solution. Finally, note that in either of the inequalities from (3), different Lipschitz constants might apply to the dependencies in x and c, in which case we simply take the maximum among these constants.

Lemma 3. *For any two sequences $\mathbf{u}_\infty, \mathbf{u}'_\infty \in U^\infty$, we have:*

$$|v(\mathbf{u}_\infty) - v(\mathbf{u}'_\infty)| \leq L_\rho \sum_{k=0}^{h-1} |c_k - c'_k| \gamma^k \frac{1 - (\gamma L_f)^{h-k}}{1 - \gamma L_f} + \frac{\gamma^h}{1 - \gamma} \tag{5}$$

where h is equal to the first step k at which $d_k \neq d'_k$.

Proof: Consider the two sequences $\mathbf{u}_\infty$ and $\mathbf{u}'_\infty$, and h defined as above. Denote by $\mathbf{u}_h$ and $\mathbf{u}'_h$ the subsequences of actions up until dimension $h - 1$, including this dimension. Then:

$$\begin{aligned} |v(\mathbf{u}_h) - v(\mathbf{u}'_h)| &= |\sum_{k=0}^{h-1} \gamma^k r_{k+1} - \sum_{k=0}^{h-1} \gamma^k r'_{k+1}| \\ &= |\sum_{k=0}^{h-1} \gamma^k (r_{k+1} - r'_{k+1})| \\ &\leq \sum_{k=0}^{h-1} \gamma^k |r_{k+1} - r'_{k+1}| \\ &\leq L_\rho \sum_{k=0}^{h-1} \gamma^k (\|x_k - x'_k\| + |c_k - c'_k|) \end{aligned} \tag{6}$$

Then, from the first part of equation (3), we get:

$$\begin{aligned} \|x_k - x'_k\| &= \|f(x_{k-1}, [c_{k-1}, d_{k-1}]^T) \\ &\quad - f(x'_{k-1}, [c'_{k-1}, d_{k-1}]^T)\| \\ &\leq L_f(\|x_{k-1} - x'_{k-1}\| + |c_{k-1} - c'_{k-1}|) \\ &\leq L_f\big(L_f(\|x_{k-2} - x'_{k-2}\| + |c_{k-2} - c'_{k-2}|) \\ &\quad + |c_{k-1} - c'_{k-1}|\big) \\ &\leq \dots \\ &\leq \sum_{i=1}^{k} L_f^i(|c_{k-i} - c'_{k-i}|) + \|x_0 - x'_0\| \\ &= \sum_{i=1}^{k} L_f^i(|c_{k-i} - c'_{k-i}|) \end{aligned} \tag{7}$$

For the last equality, we used the fact that the state sequences start from the same initial state, and so, $x_0 = x'_0$. Then, $\|x_k - x'_k\| + |c_k - c'_k| \leq \sum_{i=0}^{k} L_f^i(|c_{k-i} - c'_{k-i}|)$. Replacing this in (6), we have:

$$\begin{aligned}
|v(\mathbf{u}_h) - v(\mathbf{u}'_h)| &\leq L_\rho \sum_{k=0}^{h-1} \gamma^k \left(\sum_{i=0}^{k} L_f^i(|c_{k-i} - c'_{k-i}|) \right) \\
&= L_\rho(|c_0 - c'_0|(\gamma^0 + \gamma^1 L_f^1 + \ldots + \gamma^{h-1} L_f^{h-1}) \\
&\quad + |c_1 - c'_1|(\gamma^1 + \gamma^2 L_f^1 + \ldots + \gamma^{h-1} L_f^{h-2}) \\
&\quad + \ldots + |c_{h-1} - c'_{h-1}|(\gamma^{h-1} L_f^0)) \\
&= L_\rho \sum_{k=0}^{h-1} |c_k - c'_k| \gamma^k \frac{1 - (\gamma L_f)^{h-k}}{1 - \gamma L_f}
\end{aligned} \tag{8}$$

Starting with dimension $k = h$, the discrete actions differ, so we have a maximum difference of 1 between the rewards at each dimension. Therefore:

$$\begin{aligned}
&|v(\mathbf{u}_\infty) - v(\mathbf{u}'_\infty)| \\
&\quad \leq L_\rho \sum_{k=0}^{h-1} |c_k - c'_k| \gamma^k \frac{1 - (\gamma L_f)^{h-k}}{1 - \gamma L_f} + \sum_{k=h}^{\infty} 1 * \gamma^k \\
&\quad = L_\rho \sum_{k=0}^{h-1} |c_k - c'_k| \gamma^k \frac{1 - (\gamma L_f)^{h-k}}{1 - \gamma L_f} + \frac{\gamma^h}{1 - \gamma} \qquad \square
\end{aligned} \tag{9}$$

While this property is conservative in the sense that it "only" exploits Lipschitz continuity, it is helpful in that it drives our entire algorithm: actions will be prioritized for refinement according to their importance in (5). In future work we aim to exploit stability properties [11], [4] in order to achieve tighter bounds. Note that the right-hand side of (5) is a semimetric on the space of action sequences, in which the first component describes the impact of differences in continuous actions, and the second for the discrete actions. These two components are fundamentally different from each other. When there is no continuous action, the summation disappears and the formula simplifies to $\frac{\gamma^h}{1-\gamma}$, the OPD metric. Conversely, eliminating the discrete action is equivalent to taking $h \to \infty$, so we get $\frac{L_\rho}{1-\gamma L_f} \sum_{k=0}^{\infty} |c_k - c'_k| \gamma^k$, the OPC metric.

III. Background

Our algorithm specializes to OPD when there is no continuous action, and to OPC when there is no discrete action. Therefore, it will be important to understand these basic algorithms first.

A. Optimistic planning for continuous actions

OPC aims to find an infinitely-long sequence of continuous actions c_k that maximize the objective function v, without discrete component d. The full explanation is given in [1], and here we will provide a short description of the algorithm. OPC refines a collection of infinitely-dimensional (hyper)boxes of the form $(\mu_0 \times \cdots \times \mu_{K-1} \times [0,1] \times [0,1] \cdots)$ where μ_k is the interval of actions at step k, and K is the first unrefined dimension; from there on, all the intervals are full, $[0,1]$. OPC starts with the full set of sequences for $K = 0$, and iteratively refines it by selecting at each iteration an optimistic set with the largest upper bound on the value, and splitting it into M equal pieces along a well-chosen dimension. We return to give the precise formula for the upper bound and the choice of dimension after we have introduced OPHIS, as it will be easier to understand at that point. Here we only note that for each box, OPC needs to simulate the system with the sequence at the center of the set, up until step $K - 1$, and store the resulting state and reward trajectories. At the end, OPC returns such a center sequence that maximizes the discounted value along these K steps.

B. Optimistic planning for discrete actions

OPD [5] is an algorithm used for systems with discrete inputs. It works by building a tree, starting from a root node that corresponds to the entire set of possible actions $(0, \ldots, p)^\infty$. Then, at each step, an optimistic leaf node is chosen for expansion, by maximizing an upper bound that, like for OPC, we clarify later. Each node is expanded by making its next discrete action definite. For instance the root node will have $p+1$ children, one for each value of d_0, and the remaining actions remain free. Expanding any of these level-1 nodes makes the action d_1 definite with $p+1$ children at level 2, and so on. OPD returns a sequence on the tree with maximal sum of discounted rewards for the definite actions.

IV. Optimistic Planning for Hybrid-Input Systems

We next derive a novel algorithm that can search for sequences of hybrid inputs. The main idea is to iteratively split an optimistic set of inputs (like in OPC and OPD), but refining either the discrete or the continuous action. The key technical novelty compared to OPC and OPD is that continuous- and discrete-action refinements must be interspersed, in a way that is dictated by the intricate geometry of the space of hybrid-input sequences.

A set is represented by an interval μ for each continuous action and a discrete action set σ for each refined step k. Let us define it as such:

$$\mathbb{S}_i = \prod_{k=0}^{\infty} (\mu_{i,k}, \sigma_{i,k}) \tag{10}$$

where by the product of sets we mean the repeated application of the cross-product $\times$, and notation (μ, σ) means a set in which $c \in \mu$ and $d \in \sigma$. For clarity, from now on we will refer to the set per step k $(\mu_{i,k}, \sigma_{i,k})$ as a *pair*, and the infinite-horizon $\mathbb{S}_i$ as a *set*.

In addition, a set also has two characteristics: D_i and C_i, representing the number of refined discrete and continuous dimensions, respectively. For all $k \geq C_i$, $\mu_{i,k} = [0,1]$. For all $k < D_i$, $\sigma_{i,k} = d_{i,k}$, a single, definite value, and for all $k \geq D_i$, $\sigma_{i,k} = \{0, 1, ..., p\}$. A sequence of actions corresponding to each set would then be $(u_{i,0}, u_{i,1}, u_{i,2}, ...)$, where

$$u_{i,k} = \left[c_{i,k},\, d_{i,k}\right]^T \tag{11}$$

Here, $c_{i,k} \in \mu_{i,k}$ and $d_{i,k} \in \sigma_{i,k}$. A continuous split can be done along any dimension $k \leq C_i$, by dividing the interval $\mu_{i,k}$ into M equal pieces and thus generating M new sets. A discrete split is always done for dimension $k = D_i$, by adding $p + 1$ new sets that make discrete action d_k definite, one set for each discrete possibility. Note that the ways of splitting continuously and discretely, respectively, are fundamentally different. We provide examples splits of each type below.

To decide which set to split, let us first consider reward $r_{i,k+1} = \rho(x_{i,k}, u_{i,k})$, where with a slight abuse of notation we now refer by $c_{i,k}$ to the specific action that is at the center of interval $\mu_{i,k}$. Define then the sample value of a set i:

$$v(i) = \sum_{k=0}^{D_i-1} \gamma^k r_{i,k+1} \tag{12}$$

Each continuous interval $\mu_{i,k}$ has a certain length $a_{i,k}$. For $k \geq C_i$, $a_{i,k} = 1$. For each set, we define its diameter in the semimetric of (5):

$$\delta(i) = L_\rho \sum_{k=0}^{D_i-1} a_{i,k}\gamma^k \frac{1-(\gamma L_f)^{D_i-k}}{1-\gamma L_f} + \frac{\gamma^{D_i}}{1-\gamma} \tag{13}$$

Given the sample value of a set i and its diameter, we have the upper bound:

$$B(i) = v(i) + \delta(i) \tag{14}$$

so that $v(\mathbf{u}_\infty) \leq B(i), \forall \mathbf{u}_\infty \in \mathbb{S}_i$. This inequality is shown as follows. By the definition of set $\mathbb{S}_i$, for all $k < D_i$ we have $d_k = \sigma_{i,k}$ and $c_k \in \mu_{i,k}$, so $|c_k - c_{i,k}| \leq a_{i,k}$. All the quantities without subscript i refer to the sequence $\mathbf{u}_\infty$. Thus:

$$\begin{aligned}\sum_{k=0}^{\infty} \gamma^k r_{k+1} &= \sum_{k=0}^{D_i-1} \gamma^k r_{k+1} + \sum_{k=D_i}^{\infty} \gamma^k r_{k+1} \\ &\leq \left[v(i) + L_\rho \sum_{k=0}^{D_i-1} a_{i,k}\gamma^k \frac{1-(\gamma L_f)^{D_i-k}}{1-\gamma L_f} \right] + \frac{\gamma^{D_i}}{1-\gamma} \\ &= v(i) + \delta(i) = B(i)\end{aligned} \tag{15}$$

where the bound on the first summation (in square brackets) follows as in the proof of Lemma 3, and the bound on the second summation holds because all rewards are at most 1.

The algorithm works iteratively, by selecting for refinement at each iteration an optimistic set that maximizes the upper bound:

$$i^\dagger = \arg\max_{i \in \mathbb{A}} B(i) \tag{16}$$

where $\mathbb{A}$ is the collection of all sets.

In order to select whether we have a continuous or discrete split, and along which dimension, we look at the contribution of each dimension k up to $D_{i^\dagger} - 1$ to the diameter, as well as at the contribution $\frac{\gamma^{D_{i^\dagger}}}{1-\gamma}$ of the first unrefined dimension. Whichever contribution is the greatest among those in (13) dictates where we split. Thus:

$$k^\dagger = \arg\max_{k \in \{0,1,\ldots,D_{i^\dagger}\}} \{ L_\rho a_{i^\dagger,k} \gamma^k \frac{1-(\gamma L_f)^{D_{i^\dagger}-k}}{1-\gamma L_f} \} \tag{17}$$

If $L_\rho a_{i^\dagger,k^\dagger} \gamma^{k^\dagger} \frac{1-(\gamma L_f)^{D_{i^\dagger}-k^\dagger}}{1-\gamma L_f} \leq \frac{\gamma^{D_{i^\dagger}}}{1-\gamma}$, we have a discrete split, at dimension $D_{i^\dagger}$. Otherwise, we have a continuous split, along dimension $\min(k^\dagger, C_{i^\dagger})$. Recall that for dimensions k between $C_{i^\dagger}$ and $D_{i^\dagger} - 1$ the size a of the interval is 1. Further, note that by this rule, we always have $D \geq C$ for any set.

An example is given in Figures 1 and 2, for a continuous refinement and a discrete one, respectively. In both cases, we start from a set i, which already had 6 discrete refined dimensions ($D_i = 6$), and 5 continuous discretized dimensions ($C_i = 5$). The set is $\mathbb{S}_i = \{([4/9, 5/9], 1), ([1/3, 2/3], 0), ([2/3, 1], 0), ([0, 1/3], 1), ([1/3, 2/3], 1), ([0, 1], 0), ([0, 1], \{0, 1, ..., p\})^\infty\}$. We take the center of the interval as the continuous action, and thus we will have the following sequence of actions $\{\begin{bmatrix}1/2\\1\end{bmatrix}, \begin{bmatrix}1/2\\0\end{bmatrix}, \begin{bmatrix}5/6\\0\end{bmatrix}, \begin{bmatrix}1/6\\1\end{bmatrix}, \begin{bmatrix}1/2\\1\end{bmatrix}, \begin{bmatrix}1/2\\0\end{bmatrix}\}$. Then, f and ρ are called at each step, for this sequence of actions from x_0, to determine both the sequence of states and rewards for set $\mathbb{S}_i$.

Figure 1 shows a continuous split, along dimension 1. In the figure, one can observe in black the parent set, from which $M = 3$ new sets are formed, having the same continuous intervals for all dimensions other than $k = 1$, and the same discrete actions. The number of refined continuous dimensions C remains 5 for all children sets and D remains 6. The resulting intervals at $k = 1$ are shown with different colors and styles. At the top of the figure, we have added, symbolically, a trajectory that represents the states and the rewards. These are the same until step $k = 1$ (the refined dimension), and differ afterwards, based on the fact that the continuous actions at step 1 will be $7/18$, $1/2$ and $11/18$, respectively. The middle set will have the same continuous action and trajectories as the parent, so these trajectories can be reused.

Figure 2 shows what a discrete split looks like, starting from the same parent set. In the case of a discrete split, we always refine dimension D_i, in this case 6. One can see the new added children with colors and different line styles. They inherit the continuous intervals from the parent, as well as the previous discrete actions. Again, the sample values are the same for the children sets, until dimension $k = 6$, and then differ, based on the new discrete action.

We have mentioned before that our algorithm specializes to OPC when there is no discrete action, and to OPD when there is no continuous action. OPC chooses a set to expand based on an upper bound given by $b(i) = v(i) + \delta(i) := \sum_{k=0}^{C_i-1} \gamma^k r_{i,k+1} + \max(\frac{L_\rho}{1-\gamma L_f}, 1) \sum_{k=0}^{\infty} \gamma^k a_{i,k}$. The maximum is there to cover with a unified formula the contribution of discretized and undiscretized (unsimulated) dimensions [1]. A dimension to refine is chosen in OPC as $k^\dagger = \arg\max_{k=0,\ldots,C_i} \gamma^k a_{i^\dagger,k}$, without comparing to the discrete-action contribution because there is none. Note that the OPC diameter is differently structured only for convenience reasons, and in fact a tighter diameter can be

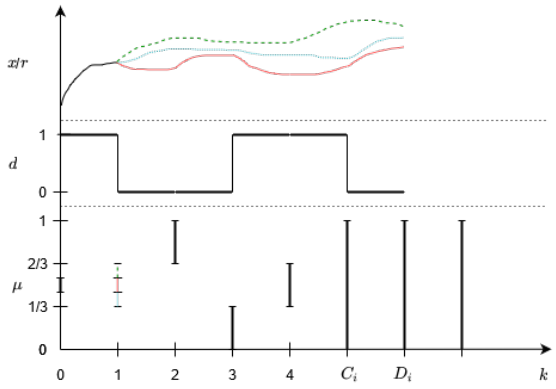

Fig. 1. Example of continuous split: top - states or rewards trajectories, middle - discrete actions, bottom - continuous intervals

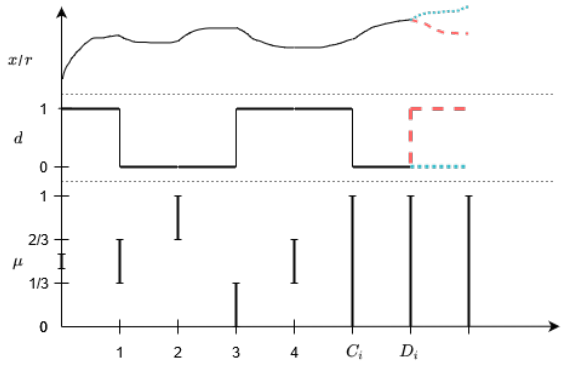

Fig. 2. Example of discrete split: top - states or rewards trajectories, middle - discrete actions, bottom - continuous intervals

written: $\delta(i) = L_\rho \sum_{k=0}^{C_i-1} a_{i,k}\gamma^k \frac{1-(\gamma L_f)^{C_i-k}}{1-\gamma L_f} + \frac{\gamma^{C_i}}{1-\gamma}$. This follows in the same way as the diameter of OPHIS, except that now instead of stopping at D_i, we stop at C_i because we have not discretized actions further so their rewards are unknown.

For OPD, only the optimistic set is chosen for expansion, based on an upper bound given as $b(i) = v(i) + \frac{\gamma^{D_i}}{1-\gamma}$; we do not need to take into account any continuous-action deviations from the center sequence.

So that we are able to reuse the center sequence at a continuous split, we impose for the remainder of the paper that M is odd (this will also help in the analysis).

Next, a pseudocode of the algorithm is given. We must pass the model of the system, as well as the initial state. The algorithm outputs a near-optimal sequence of actions.

We discuss now the algorithm inputs that are selected by the user. The budget should, of course, be taken as large as computationally feasible to get as close as possible to the optimal solution. For M we suggest to take it 3, the smallest feasible odd value, since that enables center sequence reuse without costing too much computation at each continuous refinement. Regarding now the Lipschitz constants L_f and L_ρ, while in principle they are given by the problem, in practice they may be difficult to compute (especially L_f), or computing them might give overly conservative values that work poorly across most of the state-action space. Thus we suggest treating them as tuning parameters. Similarly, γ may be fixed by the problem objective, but if it is not, then it can be treated as a tuning parameter that should not be very far from 1; larger γ will promote looking for longer-horizon solutions at the expense of refining less the continuous actions; while smaller γ will refine more the continuous actions at the expense of the horizon.

Algorithm 1: Optimistic Planning for Hybrid-Input Systems

Input: state x_0, model f, ρ, split factor M, discrete set $\{0, 1, ..., p\}$, budget n, Lipschitz constants L_f and L_ρ, discount factor γ

initialize collection of sets $\mathbb{A}$ with $\mathbb{S}_0$
while *budget still available* **do**
  select set $i^\dagger = \arg\max_{i\in\mathbb{A}} B(i)$;
  select dimension with max contribution for continuous actions $k^\dagger = \arg\max_{k\in\{0,1,...,D_{i^\dagger}\}} \{L_\rho a_{i^\dagger,k}\gamma^k \frac{1-(\gamma L_f)^{D_{i^\dagger}-k}}{1-\gamma L_f}\}$;
  if $L_\rho a_{i^\dagger,k^\dagger}\gamma^{k^\dagger} \frac{1-(\gamma L_f)^{D_{i^\dagger}-k^\dagger}}{1-\gamma L_f} \leq \frac{\gamma^{D_{i^\dagger}}}{1-\gamma}$ **then**
    /*split discretely*/
    create $p+1$ children sets from $i^\dagger$;
    children sets inherit continuous intervals and discrete actions up to dimension $D_{i^\dagger}-1$;
    create one child set for each d - this action is added for dimension $D_{i^\dagger}$;
    all children will have $D = D_{i^\dagger}+1$ and $C = C_{i^\dagger}$;
  else /*split continuously*/
    expand set $i^\dagger$ along $k^\dagger$ by creating its M children sets;
    children sets inherit continuous intervals and discrete actions up to dimension $D_{i^\dagger}-1$;
    interval at step $k^\dagger$ is refined by splitting into M equal parts;
    all children will have $D = D_{i^\dagger}$ and $C = C_{i^\dagger}$ if $k^\dagger \neq C_{i^\dagger}$, or $C = C_{i^\dagger}+1$ if $k^\dagger = C_{i^\dagger}$;

Output: sequence $\hat{u}$ of set $i^* = \arg\max_{i\in\mathbb{A}} v(i)$

At each call, the algorithm computes an open-loop, hybrid-input sequence. However, in practice the algorithm should be applied in receding horizon. This means that at each step, we run the algorithm starting from the current state, we apply the first action of the returned sequence to the system, reaching the next state, where we reapply the algorithm, and so on. Thus, overall, a closed-loop control scheme is obtained.

In order to obtain a measure of the required computation, we need to look to the number of calls made to the dynamics f and the reward function ρ. Of course other operations matter as well in the computation time. However, when the dynamics are nonlinear, the numerical integration is usually expensive, making the number of calls to f dominant. If we

have a discrete expansion, we make $p+1$ calls to simulate the new discrete step with each of the $p+1$ discrete actions and continuous action 0.5. When we have a continuous split of set $i^\dagger$ at dimension $k^\dagger$, we need $(M-1)(D_{i^\dagger}-k^\dagger)$ calls to simulate the $M-1$ sequences (except the center one, which is reused) from step $k^\dagger$ to step $D_{i^\dagger}$.

V. A posteriori analysis of near-optimality

We provide a bound on the near-optimality of the sequence returned that is explicitly available for use *a posteriori*, once the algorithm has run.

Proposition 4. *The sequence i^* returned by the algorithm satisfies:*

$$v^* - v(i^*) \leq \delta_{\min}$$

where $\delta_{\min}$ is the smallest diameter among all the sets expanded by the algorithm.

Proof: Consider any set i^+ expanded at some iteration. We have $B(i^+) \geq v^*$, for the following two reasons. First, as the sets currently considered by the algorithm form a partition of the set of solutions, one of them (let us call it i_{opt}) contains the optimal solution with value v^*. Thus, $B(i_{\text{opt}}) \geq v^*$. Second, set i^+ is optimistic, hence it has the largest upper bound among all sets in the collection at that iteration, and in particular $B(i^+) \geq B(i_{\text{opt}})$.

Moreover, each split produces at least one child set i_c with $v(i_c) \geq v(i^+)$: a discrete split adds new, positive rewards to the end of the center sequence; and at a continuous split the center sequence is inherited by the middle child from the parent. This means that in the final collection of sets, there is at least one set i_l that is a descendant of i^+ for which $v(i_l) \geq v(i^+)$. But $v(i^*) \geq v(i_l)$ by the selection rule of i^*, so $v(i^*) \geq v(i^+)$.

Overall, $v(i^+) \leq v(i^*) \leq v^* \leq B(i^+)$, thus $v^* - v(i^*) \leq B(i^+) - v(i^+) = \delta(i^+)$, and since this is true at any iteration, the inequality is also satisfied with $\delta_{\min}$. □

Note we can easily compute lower and upper bounds on the optimal value: $v^* \in [v(i^*), B^*]$, where B^* is the *smallest* upper bound of any set expanded. Such bounds are popular in hybrid systems, where they are called certification bounds [3].

Such properties are standard for OP algorithms; both OPD and OPC ensure similar bounds. The property of increasing set values along continuous splits is not trivial, especially for OPHIS where we need to handle discrete and continuous splits at the same time.

In future work, we also expect to obtain an *a priori* convergence rate of $\delta_{\min}$ to 0 with increasing budget n. This requires a deep understanding of the shape of the sets given by the expansion rules, and of the amount of sets expanded.

VI. Simulation results

This section discusses the results of applying this algorithm for 2 hybrid-input problems. First, a system of two cascaded tanks is considered, and then a two-link robot arm example is presented.

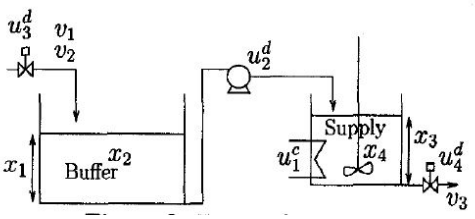

Fig. 3. Tanks system, taken from [13]

A. Cascaded tanks

We begin by discussing the case of two cascaded tanks, a buffer tank and a supply tank. The setup can be seen in Figure 3. The model is taken from [13] and its details are given next. The state is $x = [x_1, x_2, x_3, x_4]^T$, where x_1 and x_3 are the buffer and supply levels and x_2 and x_4 are the buffer and supply temperatures. The continuous action c is u_1^c and represents the heater's power, while the discrete action d is u_2^d and represents whether the pump is closed, half-open, and open. Discrete signals, u_3^d and u_4^d, for the inlet and outlet valves, are always 1 (open, because it is undesirable to have inlet and outlet valves closed), so we do not consider them as inputs. Moreover, v_1, v_2, v_3 represent the inflow, the inflow temperature and the outflow, respectively, and are considered constant. Setting all these constant values was also done in [13], where the parameters of the model are also given. Here, we present only the dynamics of the system:

$$\begin{aligned} \dot{x}_1 &= \tfrac{1}{A_b}(v_1 u_3^d - \alpha u_2^d) \\ \dot{x}_2 &= \tfrac{1}{A_b x_1}(-x_2 v_1 u_3^d + v_1 v_2 u_3^d) \\ \dot{x}_3 &= \tfrac{1}{A_s}(\alpha u_2^d - v_3 u_4^d) \\ \dot{x}_4 &= \tfrac{1}{A_s x_3}((x_2 - x_4)\alpha u_2^d + \tfrac{u_1^c}{c_l \rho_l}) \end{aligned} \tag{18}$$

The goal is to control the temperature in the supply tank, by bringing it to a desired value. Neither of the tanks should become empty or overflow. The setpoint for the supply tank temperature is $x_{40} = 22°\text{C}$. The operating points for the variables are: $x_{10} = 7\text{m}$, $x_{20} = 18°\text{C}$, $x_{30} = 1,5\text{m}$, $u_{10} = 280\text{kW}$, $u_{20} = 1\text{m}^3/\text{min}$, $u_{30} = 1\,(\text{open})$, $u_{40} = 1\,(\text{open})$, $v_{10} = 1(\text{stage } 1)$, $v_{20} = 18°\text{C}$, $v_{30} = 1\text{m}^3/\text{min}$. Control is performed in discrete time with a sampling period $\Delta = 0.25\text{min}$. Recall that the state variables, as well as u_1 and u_2 will change values, while the remaining variables are considered constants and equal to their operating point values. We have mentioned in Section II that c should be bounded and in interval $[0, 1]$. In the algorithm, $c = 0$ means that the heater is off, while $c = 1$ represents $2u_{10}$, maximum heating. The discrete action d can take 3 values: 0 for pump closed, 1 for pump half open, and 2 for pump open. These are the same conditions for the simulations as used in [13].

Simulations were run in a receding horizon manner. The integration was done using the Euler method, in one step per sampling period, so that $x_{k+1} = x_k + \Delta \dot{x}_k$. The reward function used was:

$$\rho(x_k, u_k) = (5 - (x_{4,k+1} - x_{4,0})^2)/5; \tag{19}$$

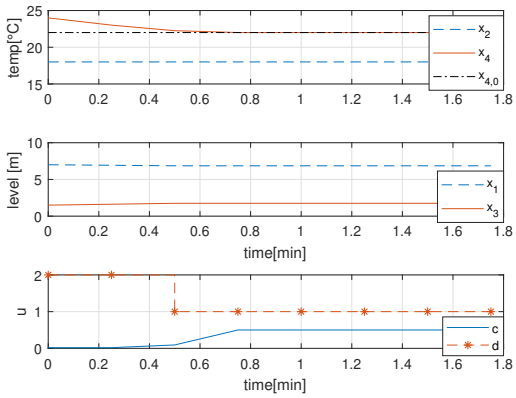

Fig. 4. Evolution in time of the states and actions for the two-tanks system

where $x_{4,k+1}$ is the fourth state resulting at the next step. Note that [13] used undiscounted cost, while we have a discount factor in (2); to compare the approaches, we will therefore report the undiscounted costs obtained by our method, even though this method works with discounting. Any unbalance in this comparison is in favor of the baseline.

The initial state was $x_0 = [7, 18, 1.5, 24]$. For all the experiments, we keep $\gamma = 0.8$ and a continuous interval is always split in $M = 3$ intervals. Initially we selected the budget to be 100, and $L_f = L_\rho = 1.2$. These Lipschitz constants were tuned experimentally, for the reasons explained in Section IV, see also Figure 6 for the tuning results; for simplicity we kept the two constants equal. We ran the simulations for 2 minutes of simulated time. The results can be seen in Figure 4, where we can observe the states and the two actions, respectively. The first subplot shows the temperature in the two tanks. The temperature in the buffer tank (blue dashed line) remains the same, as there is no disturbance. The temperature in the supply tank (red, continuous line) decreases to reach its nominal value (black, dash-dotted line). The second plot in the figure represents the levels in the tanks, with red continuous for the supply tank, and blue dashed line for the buffer tank. There has been no overflowing or emptying of the tanks. The third subplot shows in blue continuous line the values of the continuous action c, with $u_1 = 2 \cdot c \cdot u_{10}$, and in red, the values of the discrete action d at each step. As you can see, the temperature reaches the setpoint value in 0.75 minutes. Compared to the classical nonlinear MPC method in [13], we obtain the same results. So in this relatively simple problem, we recover the performance from the literature with a small budget, which provides confidence in the algorithm. Note that even though our bound (5) is conservative, we recover the classical MPC performance, while having a more general algorithm.

Next, we change some of the parameters, to see their influence on the performance and execution time of the algorithm. First, we are interested in tuning the budget. Figure 5 shows first the undiscounted cost as a function of the budget and then the average time for the algorithm

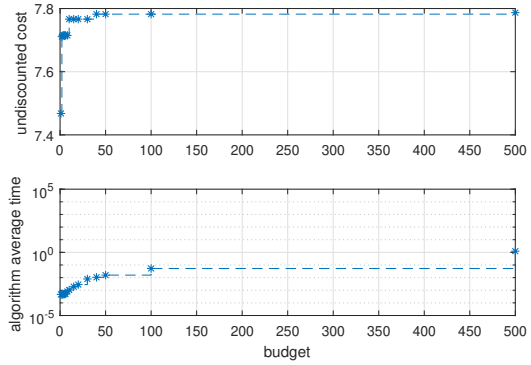

Fig. 5. Undiscounted cost and average execution time as a function of budget; For the bottom subplot, the Y axis is logarithmic

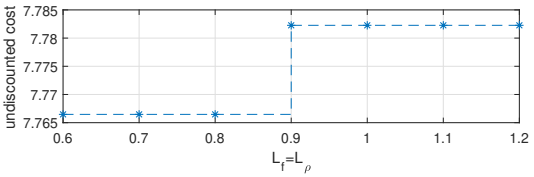

Fig. 6. Undiscounted cost as a function of Lipschitz values

to run once, in seconds. When we increase the budget, the time of course increases, but so does the performance of the algorithm. However, in this simple example of the two cascaded tanks, even a small budget is sufficient in order to have a good performance. The timing is good enough for this problem, as even with 1000 budget, it takes about 8 seconds to run the algorithm once, which is roughly half of the sampling time $\Delta = 0.25\text{min} = 15\text{s}$.

We now tune the Lipschitz constants, keeping them equal to one another. The budget remained 100. Figure 6 shows again the undiscounted return for different Lipschitz values. The performance in terms of cost is not very sensitive to the Lipschitz value (note the vertical scale).

B. Two-link robot arm

This subsection presents the simulations run for a robot arm with 2 joints, one controlled joint and one which can only have a brake set [2]. Figure 7 shows its kinematic structure. The model of this robot is given in [7] and is not presented here, as it would take too much space. The state vector is represented by the two angles and the angular velocities:

$$\mathbf{x} = [\theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2] \tag{20}$$

The continuous control action is the torque τ_1, corresponding to the first joint, and the discrete action is represented by the braking torque τ_b. The dynamics are derived using Euler-Lagrange and are given in [7]. The parameters used are the same as in [2]. In addition, we take the following values for the parameters that are not given in the cited papers: the angle $\alpha = -\pi/12$, τ_b will either take the value 0 or 1 and the maximum value of τ_1 will be 20, which will be rescaled to 1 for continuous action c. The numerical integration is done

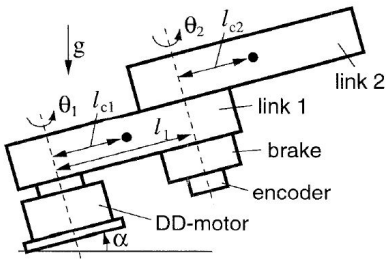

Fig. 7. Two-link robot arm, taken from [7]

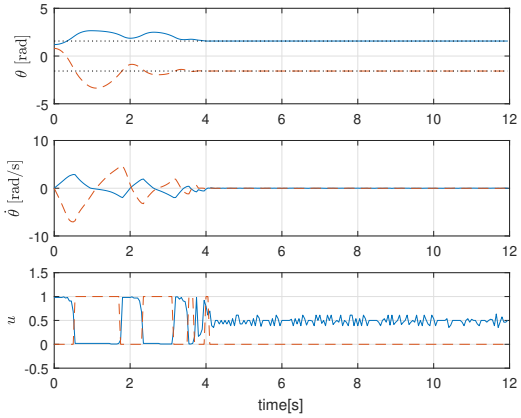

Fig. 8. Evolution in time of the states and actions for the two-link robot arm

using the Euler method, for 5 integration steps per control sampling time, which is $\Delta = 0.05$s. The goal is to get the state to a desired setpoint $\mathbf{x}_f$. The reward function is:

$$\rho(\mathbf{x}_k, u_k) = (10 - |x_{1_{k+1}} - x_{1_f}| - |x_{3_{k+1}} - x_{3_f}|)/10 \quad (21)$$

where x_{k+1} is the state at the next step. We took a nondifferentiable reward to showcase that the algorithm is resilient to this.

The following values were used for the algorithm: $M = 3$, budget of 1000, $L_f = L_\rho = 1.2$, $\gamma = 0.8$. Thus, only the budget is changed from the previous problem, without retuning anything else. Again, a receding horizon simulation is done, for 12 seconds. The starting position is $\mathbf{x}_0 = [1.2, 0, 0.8, 0]$. The desired final state is $\mathbf{x_f} = [\pi/2, 0, -\pi/2, 0]$. The results can be seen in Figure 8, which shows the evolution of the states and actions in time. The top subplot presents the angles, and the middle one the angular velocities. With blue continuous lines we can see the states corresponding to the first, actuated, joint. The red, dashed lines represent the states of the second joint, which can only be influenced by a holding brake. As one can observe, both angles reach their desired setpoints, which are represented in black dotted lines. Also, the final velocities are oscillating around 0, and the brake is not set. The last subplot presents the evolution in time of the 2 actions: with blue continuous line c and with red dashed line d.

A comparison with [2] is unfortunately not possible, since, as stated above, there are several missing parameters (α, τ_b and the maximum of τ_1), which have a great influence on the model, according to other simulations that we have run.

VII. Conclusions

The Optimistic Planning for Hybrid-Input Systems algorithm is proposed for systems with both continuous and discrete actions. It is successful for two examples, a simple two-tanks system and a two-link robot arm.

In the future, we plan to extend the analysis of the algorithm so as to include convergence rate guarantees, as explained in Section V. Also, we aim to derive a so-called simultaneous version of the algorithm, which does not require to know the Lipschitz constants.

References

[1] L. Buşoniu, E. Páll, and R. Munos, "Continuous-action planning for discounted infinite-horizon nonlinear optimal control with Lipschitz values," *Automatica*, vol. 92, pp. 100–108, 2018.

[2] M. Buss, M. Glocker, M. Hardt, O. Von Stryk, R. Bulirsch, and G. Schmidt, "Nonlinear hybrid dynamical systems: modeling, optimal control, and applications," in *Modelling, Analysis, and Design of Hybrid Systems*. Springer, 2002, pp. 311–335.

[3] J. C. Geromel and R. H. Korogui, "H2 robust filter design with performance certificate via convex programming," *Automatica*, vol. 44, no. 4, pp. 937–948, 2008.

[4] M. Granzotto, R. Postoyan, L. Buşoniu, D. Nešić, and J. Daafouz, "Optimistic planning for the near-optimal control of nonlinear switched discrete-time systems with stability guarantees," in *2019 IEEE 58th Conference on Decision and Control (CDC)*. IEEE, 2019, pp. 3405–3410.

[5] J.-F. Hren and R. Munos, "Optimistic planning of deterministic systems," in *European Workshop on Reinforcement Learning*. Springer, 2008, pp. 151–164.

[6] J. Lygeros, C. Tomlin, and S. Sastry, "Hybrid systems: modeling, analysis and control," *Electronic Research Laboratory, University of California, Berkeley, CA, Tech. Rep. UCB/ERL M*, vol. 99, 2008.

[7] J. Mareczek, M. Buss, and G. Schmidt, "Robust global stabilization of the underactuated 2-DOF manipulator R2D1," in *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No. 98CH36146)*, vol. 3. IEEE, 1998, pp. 2640–2645.

[8] N. N. Nandola and S. Bhartiya, "A multiple model approach for predictive control of nonlinear hybrid systems," *Journal of process control*, vol. 18, no. 2, pp. 131–148, 2008.

[9] N. N. Nandola and K. Puttannaiah, "Modeling and predictive control of nonlinear hybrid systems using disaggregation of variables-A convex formulation," in *2013 European Control Conference (ECC)*. IEEE, 2013, pp. 2681–2686.

[10] L. G. J. Pannek and L. Grüne, "Nonlinear Model Predictive Control: Theory and Algorithms," in *Nonlinear Model Predictive Control*. Springer, 2011, pp. 2267–2274.

[11] R. Postoyan, L. Buşoniu, D. Nešić, and J. Daafouz, "Stability analysis of discrete-time infinite-horizon optimal control with discounted cost," *IEEE Transactions on Automatic Control*, vol. 62, no. 6, pp. 2736–2749, 2016.

[12] M. Sarailoo, Z. Rahmani, and B. Rezaie, "A novel model predictive control scheme based on bees algorithm in a class of nonlinear systems: Application to a three tank system," *Neurocomputing*, vol. 152, pp. 294–304, 2015.

[13] O. Slupphaug, J. Vada, and B. A. Foss, "MPC in systems with continuous and discrete control inputs," in *Proceedings of the 1997 American Control Conference (Cat. No. 97CH36041)*, vol. 5. IEEE, 1997, pp. 3495–3499.

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

[15] A. J. Van Der Schaft and J. M. Schumacher, *An introduction to hybrid dynamical systems*. Springer London, 2000, vol. 251.