Optimistic planning for control of hybrid-input nonlinear systems

Ioana Lal, Irinel-Constantin Morarescu, Jamal Daafouz, Lucian Buşoniu

Optimistic planning for control of hybrid-input nonlinear systems [1]

Ioana Lal [a] Irinel-Constantin Morărescu [b] Jamal Daafouz [b] Lucian Buşoniu [a]

[a] *Technical University of Cluj-Napoca, Romania*

[b] *Université de Lorraine, CRAN, UMR 7039 and CNRS, CRAN, UMR 7039, Nancy, France*

Abstract

We propose two branch-and-bound, optimistic planning algorithms for discrete-time 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 start by investigating the structure of the space of hybrid-input sequences. Based on this structure, we propose for the first algorithm an optimistic selection rule that picks for refinement (branching) the subset with the largest upper bound on the value. At the price of a higher budget, the second method reduces the reliance on the Lipschitz constant, by refining all sets that are potentially optimistic. This effectively means that the Lipschitz constant is automatically optimized. The way to select the largest-impact action along which to refine the sets is the same for both algorithms, and still depends on the Lipschitz constant. We provide convergence rate guarantees for both methods, which link the computational budget to the near-optimality of the action sequences returned, in a way that depends on a problem complexity measure. We also give empirical results for a nonlinear problem, where the algorithms are applied in receding horizon, and depending on the budget either one or the other algorithm works better.

Key words: Optimal control; planning; hybrid-input nonlinear systems; near-optimality analysis

1 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 [**?**], [**?**], industrial multiple-tanks systems [**?**], [**?**], [**?**], hydraulic systems [**?**], the automotive industry, for joint control of engine power and the transmission gear [**?**], [**?**], etc. A number of techniques were used for solving this type of problems, such as branch-and-bound approaches [**?**], switching control [**?**], or model-predictive control (MPC) [**?**], [**?**], [**?**], [**?**]. Compared to these methods, our approach can handle problems with more general dynamics and cost functions, while focusing on infinite-horizon optimal control, rather than finite-horizon. In addition, near-optimality bounds are provided, along with two fully-specified, directly implementable algorithms (which is not always the case with other approaches). These bounds are then used for proving convergence rates for both algorithms.

Specifically, we consider hybrid-input systems for which the dynamics can be generally nonlinear and the cost functions arbitrary, as long as both are Lipschitz with respect to the state and continuous action. This is not a greatly restrictive property, since usual dynamics and cost functions satisfy the constraint. The continuous action must be scalar, a restriction that can be relaxed at extra computational cost. For such systems, we propose two methods, called OPHIS and SOPHIS: Optimistic Planning for Hybrid-Input Systems, and Simultaneous OPHIS. Both algorithms are a nonstandard flavor of MPC, produce at each given state an open-loop sequence, and are meant to be applied in receding horizon. The two algorithms also belong to the optimistic planning (OP) class of algorithms. OP approaches explore the space of infinitely long sequences of actions, and re-

[1] Corresponding and shared first authors I. Lal and L. Buşoniu. Email addresses: ioanalal04@gmail.com, constantin.morarescu@univ-lorraine.fr, jamal.daafouz@univ-lorraine.fr, lucian@busoniu.net. This work was financially supported from EU's H2020 SeaClear project No 871295; and by the project 38 PFE in the frame of the programme PDI-PFE-CDI 2021. The work of I.C. Morărescu and J. Daafouz was funded by the ANR under grants HANDY ANR-18-CE40-0010 and NICETWEET ANR-20-CE48-0009. Computation resources were provided by the CLOUDUT Project, cofunded by the European Fund of Regional Development through the Competitiveness Operational Programme 2014-2020, contract no. 235/2020.

fine regions in which optimal solutions may be located.

OPHIS creates a partition of the set of hybrid-input sequences iteratively, by choosing for refinement at each iteration an optimistic set, i.e. one that maximizes an upper bound on the value. For any set that is chosen for refinement, a dimension (time step) is also chosen, together with the type of split (continuous or discrete).

OPHIS depends on the Lipschitz constant in two places: set selection and dimension selection. In practice, the Lipschitz constant is difficult to estimate: a too large value makes the algorithm slow, and a too small value invalidates it. Therefore, we introduce our second algorithm, SOPHIS, which removes the dependence on the Lipschitz constant *in set selection*, by refining several sets per iteration: any that may be optimistic regardless of the value of the Lipschitz constant. Although dimension selection still has to depend on the constant, we expect that set selection has a larger impact, and this intuition is confirmed by experiments in Section 6: SOPHIS is less sensitive to the value of the Lipschitz constant.

Regarding experimental performance, both algorithms have their use: SOPHIS works better for larger budgets, whereas for smaller ones, the OPHIS approach of focusing this limited budget on one value of the Lipschitz constant still pays off. Analytically, the convergence rate of SOPHIS (when properly tuned) is almost as good as that of OPHIS. However, this is not the full story: the fact that SOPHIS expands sets for all possible Lipschitz constant values means in effect that the rates are those for the best possible value of the constant; in effect, SOPHIS automatically optimizes the Lipschitz constant for the set selection component.

The first method, OPHIS, has already been described in our preliminary conference paper [**?**], where the focus was on deriving the method and empirically validating it in two separate problems. In addition, only an a-posteriori guarantee was given for OPHIS. In this paper we introduce an extension to this algorithm, namely SOPHIS. Furthermore, we provide proof of convergence rates to the global infinite-horizon discounted optimum for both methods. We define a complexity measure of any problem to which (S)OPHIS is applied, and prove that the algorithms converge faster for smaller complexity measures. In particular, in the simplest type of problem, convergence is according to an exponential in a power of the computational budget. This is significant because it gives **strong guarantees of near-optimality, which to our knowledge, are not given for other hybrid-input methods** in the literature [**?**], [**?**], [**?**].

We exemplify OPHIS and SOPHIS in simulations. Both methods succeed in controlling a robot arm where one link is continuously controlled and the other can have a brake either applied or not. OPHIS wins for small budgets, but SOPHIS is better for large budgets and less sensitive to the Lipschitz constant value.

OPHIS essentially combines optimistic planning for deterministic, discrete-input systems (OPD) [**?**], [**?**] and OP for continuous-input systems (OPC) [**?**]. The new algorithm SOPHIS, which does not need the Lipschitz value in the set selection, is similar to the extension of OPC to SOPC in [**?**]. The key novel technical challenge here is that the structure of the hybrid space of solutions is significantly more complicated than for either OPC or OPD, and consequently so are the refinement rules that we must come up with to explore this space. OPHIS in fact specializes to OPD when the continuous action is removed, and to OPC when the discrete action is removed; and SOPHIS specializes to SOPC.

A branch and bound approach related to optimistic planning is used in [**?**], in combination with sparse direct collocation. However, no near-optimality analysis is provided there. A key difference between usual hybrid-input control approaches and (S)OPHIS is that the former primarily concentrate on stability, such as in [**?**], 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 [**?**] and for discrete-input optimistic planning [**?**]. However, the behavior of (S)OPHIS when refining continuous actions is much more intricate, and analyzing its stability is left for future work.

In the landscape of tree search methods [**?**], OP is a best-first method, since it aims to refine the optimistic set. Compared to Monte-Carlo tree search [**?**], which uses random sampling to make the search more efficient, the OP flavor that we use has the advantage of providing deterministic guarantees.

Next, Section 2 formalizes the problem and Section 3 discusses the background on OPC, SOPC, and OPD. OPHIS and SOPHIS are described in Section 4, while Section 5 provides the convergence rate analysis. Finally, Section 6 gives simulation results for a two-link robot arm, and Section 7 gives the conclusions of this paper.

2 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:

$$u_k = [c_k \ d_k]^T \tag{1}$$

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.

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

Together with discounting, the bounded rewards ensure boundedness of the sequence values. Bounded costs are typical in e.g. reinforcement learning for control [**?**] and they could follow either from physical limitations in the system, or from saturating an a priori unbounded reward function, which changes the optimal solution but may be sufficient. Note that now $U = ([0,1] \times \{0, 1, ..., p\})$. The restriction to scalar continuous actions can be relaxed, but at significant extra computational cost for the extended algorithm; see [**?**], supplementary material for such an extension in OPC. The growth would be exponential in the number of inputs.

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 (i) 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. Note also that 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 dead-zones, 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.

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

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

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

The proof of Lemma 3 can be found in [**?**]. The property in Lemma 3 drives our entire algorithm: actions will be prioritized for refinement according to their importance in (5). The two terms on the right-hand side of the inequality correspond to continuous and discrete actions, respectively, and are fundamentally different from each other. When there is no continuous action, the summation disappears and the formula simplifies to $\frac{\gamma^D}{1-\gamma}$, the OPD metric [**?**]. Conversely, eliminating the discrete action is equivalent to taking $D \to \infty$, so we get $\frac{L_\rho}{1-\gamma L_f} \sum_{k=0}^{\infty} |c_k - c'_k| \gamma^k$, the OPC metric [**?**].

3 Previous optimistic planning algorithms

OPHIS specializes to OPD when there is no continuous action, and to OPC when there is no discrete action. Under some conditions, SOPHIS also specializes to SOPC when there is no discrete action. Therefore, understanding these basic algorithms is important.

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 [**?**]; 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. 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. Finally, OPC returns such a center sequence that maximizes the discounted value along these K steps.

SOPC [**?**] is an extension of OPC, where all the sets that may have the largest upper bound for any Lipschitz value are expanded. Both the selection and splitting rules, through modifications, become independent from the Lipschitz constant.

Optimistic planning for discrete actions (OPD) [**?**] is an algorithm used for systems with discrete inputs. It works by building a tree, starting from a root node corresponding 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. 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 level-1 node 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.

4 Hybrid-input algorithms

In the sequel we derive two novel algorithms 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 unlike those algorithms, by refining either the discrete or the continuous action. For this, we look at the uncertainty on the value of the set, and choose the action with the largest impact on the uncertainty. The key technical novelty compared to OPC and OPD is that continuous- and discrete-action refinements must be alternated, 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:

$$\mathbb{S}_i = \prod_{k=0}^{\infty} (\mu_{i,k}, \sigma_{i,k}) \tag{6}$$

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*.

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 is then $(u_{i,0}, u_{i,1}, u_{i,2}, ...)$, where:

$$u_{i,k} = \Big[c_{i,k},\, d_{i,k}\Big]^T \tag{7}$$

and $c_{i,k} \in \mu_{i,k}$, $d_{i,k} \in \sigma_{i,k}$. 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 example splits of each type below.

4.1 OPHIS

In this subsection, we focus on the OPHIS algorithm.

In order 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:

$$v(i) = \sum_{k=0}^{D_i - 1} \gamma^k r_{i,k+1} \tag{8}$$

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):

$$\begin{gathered} \sup_{\mathbf{u}_\infty, \mathbf{u}'_\infty \in \mathbb{S}_i} |v(\mathbf{u}_\infty) - v(\mathbf{u}'_\infty)| \leq \delta(i) \\ \delta(i) = L_\rho \textstyle\sum_{k=0}^{D_i - 1} a_{i,k} \gamma^k \frac{1 - (\gamma L_f)^{D_i - k}}{1 - \gamma L_f} + \frac{\gamma^{D_i}}{1 - \gamma} \end{gathered} \tag{9}$$

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

$$B(i) = v(i) + \delta(i) \tag{10}$$

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:

$$\begin{aligned} \sum_{k=0}^{\infty} \gamma^k r_{k+1} &= \sum_{k=0}^{D_i - 1} \gamma^k r_{k+1} + \sum_{k=D_i}^{\infty} \gamma^k r_{k+1} \\ &\leq \left[v(i) + L_\rho \sum_{k=0}^{D_i - 1} a_{i,k} \gamma^k \frac{1 - (\gamma L_f)^{D_i - k}}{1 - \gamma L_f} \right] + \frac{\gamma^{D_i}}{1 - \gamma} \\ &= v(i) + \delta(i) = B(i) \end{aligned} \tag{11}$$

where the bound on the first summation (in square brackets) follows as in [**?**], and the bound on the second summation holds because all rewards are at most 1.

Denoting $\mathbb{A}$ as the collection of all sets, OPHIS iteratively selects for refinement at each iteration an optimistic set that maximizes the upper bound:

$$i^\dagger = \arg\max_{i \in \mathbb{A}} B(i) \tag{12}$$

In order to decide whether we have a continuous or discrete split of $\mathbb{S}_{i^\dagger}$, and along which dimension, we look at the contribution of each dimension k up to $D_{i^\dagger} - 1$ to the diameter (9), as well as at the contribution $\frac{\gamma^{D_{i^\dagger}}}{1-\gamma}$ of the first unrefined dimension. Whichever contribution is the greatest dictates where we split. Thus:

$$k^\dagger = \arg\max_{k \in \{0, 1, ..., D_{i^\dagger}\}} \{ L_\rho a_{i^\dagger, k} \gamma^k \frac{1 - (\gamma L_f)^{D_{i^\dagger} - k}}{1 - \gamma L_f} \} \tag{13}$$

and 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.

For compactness reasons, we shall denote the contribution of a dimension k in the continuous part of the diameter with $\lambda_k = L_\rho a_k \gamma^k \frac{1-(\gamma L_f)^{D-k}}{1-\gamma L_f}$.

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 $\mathbb{S}_i = \{([4/9, 5/9], 1) \times ([1/3, 2/3], 0) \times ([2/3, 1], 0) \times ([0, 1/3], 1) \times ([1/3, 2/3], 1) \times ([0, 1], 0) \times ([0, 1], \{0, 1, ..., p\})^\infty\}$. We take the center of the interval as the continuous action, and thus we will have the following sequence of actions $\big((1/2, 1), (1/2, 0), (5/6, 0), (1/6, 1), (1/2, 1), (1/2, 0)\big)$. 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 $\mathbb{S}_i$. In this example, there are 2 possible discrete actions, 0 and 1, and $M = 3$.

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,since 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. A discrete split implies always refining dimension D_i, in this case 6. One can see the newly added children with colors and different line styles. They inherit the continuous intervals from the parent, as well as the previous discrete actions. The sample values are the same for the children sets, until dimension $k = 6$, and then they differ, based on the new discrete action.

Overall, we create a tree structure, where each set is a node in the tree. Whenever a discrete split is done for set i, $p + 1$ children are added to the node representing set i in the tree. In the case of a continuous split, M children are added instead. An example of such a tree can be seen in Figure 3. Discrete splits are represented by blue, continuous lines and continuous splits by red dotted lines. As we can observe, for the root node, all continuous intervals are $[0, 1]$ and all discrete actions are

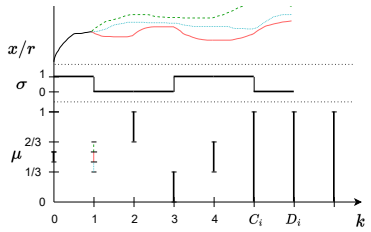

Fig. 1. Example of continuous split: top - states or rewards trajectories, middle - discrete actions, bottom - continuous intervals

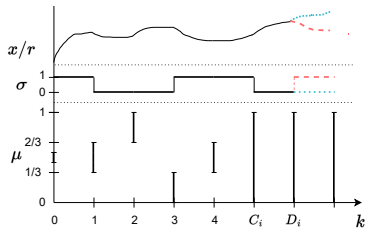

Fig. 2. Example of discrete split: top - states or rewards trajectories, middle - discrete actions, bottom - continuous intervals

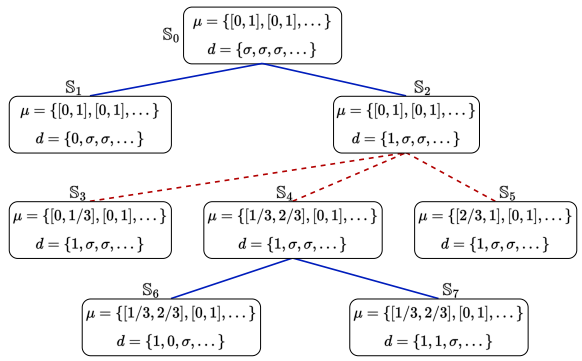

Fig. 3. Example of tree

undefined. Then, by a discrete split, we get two new sets, where the first discrete action is defined as 0 for set $\mathbb{S}_1$, and 1 for set $\mathbb{S}_2$, respectively. In the same way, 2 new sets are formed in Figure 2, by defining the discrete action at step $D_i = 6$. Then, set $\mathbb{S}_2$ is chosen for refinement, and a continuous refinement on dimension 0 is done. This adds sets $\mathbb{S}_3, \mathbb{S}_4, \mathbb{S}_5$, each with a third of interval for the first dimension. Again, this is similar with the continuous split done in Figure 1, where 3 new sets are added, each with the interval for the second dimension being a third of the interval of their parent set.

Note that even though the interval refinements are discrete at each step, asymptotically, after a large number of expansions, the algorithm can reach arbitrarily close to any continuous value; in other words, the set of applicable actions is dense in the unit interval.

We have mentioned before that OPHIS 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 [**?**]. A dimension to refine is chosen in OPC as $k^\dagger = \arg\max_{k=0,...,C_i} \gamma^k a_{i\dagger,k}$, 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 written:

$$\delta(i) = L_\rho \sum_{k=0}^{C_i-1} a_{i,k} \gamma^k \frac{1-(\gamma L_f)^{C_i-k}}{1-\gamma L_f} + \frac{\gamma^{C_i}}{1-\gamma} \qquad (14)$$

This follows in the same way as the diameter (9) 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, 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 method is given in Algorithm 4.1. We must pass the model of the system, as well as the initial state. The initial set $\mathbb{S}_0$ represents the root of the tree, with all the continuous intervals being $\mu = \{[0,1],[0,1],[0,1],...\}$ and the discrete actions not yet defined as certain values, but allowing the sets of all possible values $\{0,1,...,p\}$. This is because no refinement has been done yet; as such, we also have that $D_0 = C_0 = 0$. An important parameter is the budget n , which represents the number of simulations of the system dynamics that we are allowed to perform. The algorithm outputs a near-optimal sequence of actions.

4.2 SOPHIS

In this section, we discuss an extension of the OPHIS algorithm, in which we have a different set selection rule. To avoid assuming knowledge of the Lipschitz constant for this rule, we will expand all the sets that may be optimistic for any value of this constant. Note however that the Lipschitz constant must still satisfy Assumption 2. Recall that both methods create a tree of sets, and denote by H the depth in this tree. Quantity H represents the sum of continuous and discrete expansions done in order to reach a certain set. Since all sets have the same shape, diameters $\delta(i)$ are the same at a given depth, so the maximum-upper-bound set at that depth can only be a set with the largest value $v(i)$. Thus, at each depth

Algorithm 1 OPHIS

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 $\mathbb{S}_0$, $D_0 = 0, C_0 = 0$;
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 $k^\dagger = \arg\max_{k \in \{0,1,...,D_{i^\dagger}\}} \lambda_k$;
  if $\lambda_{k^\dagger} \leq \frac{\gamma^{D_{i^\dagger}}}{1-\gamma}$ **then** /*split discrete*/
    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 continuous*/
    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 $k^\dagger = C_{i^\dagger}$;
  end if
end while
output sequence $\hat{u}$ of set $i^* = \arg\max_{i \in \mathbb{A}} v(i)$

H that still has nodes unexpanded, we expand the set with the greatest v value among all sets at that depth.

To prevent expansions from continuing indefinitely we also configure a maximum depth $H_{\max}(n)$ up to which the expansions are allowed to continue when the budget spent so far is n. If $H_{\max}$ grows fast with budget n, the algorithm will favour deep searches. On the contrary, a slower growth with n allows us to do a search focused on breadth. More insight into choosing $H_{\max}$ will be provided in the analysis. Of course, expanding several sets per iteration comes at an extra cost. However, as we will show in the analysis, this does not have a great impact on the guarantees and in simulations performance is often improved.

The dimension selection rule for SOPHIS will remain the same as for OPHIS. This means that it will unfortunately still depend on the Lipschitz constant, and there is no way to avoid this.

As previously mentioned, the extension from OPHIS to SOPHIS is similar to the one from OPC to SOPC [**?**]. Both SOPHIS and SOPC refine several sets per iteration, based on the best value at each depth. Moreover, SOPHIS would specialize to SOPC if there was no discrete action and if SOPC were to use the tighter diameter (14) of Section 4.1. A pseudocode of SOPHIS is given

in Algorithm 4.2.

Algorithm 2 SOPHIS

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, $H_{\max}(n)$
initialize collection of sets $\mathbb{A}$ with $\mathbb{S}_0, D_0 = 0, C_0 = 0$;
while budget still available **do**
  $H =$ smallest depth with unexpanded nodes;
  if $H \geq H_{\max}(n)$ **then**
    stop and exit the loop;
  else
    while $H < H_{\max}(n)$ **do**
      select set $i^\dagger = \arg\max_{i \in \mathbb{A}} v(i)$;
      select dimension $k^\dagger = \arg\max_{k \in \{0,1,...,D_{i^\dagger}\}} \lambda_k$;
      if $\lambda_{k^\dagger} \leq \frac{\gamma^{D_{i^\dagger}}}{1-\gamma}$ **then**
        split discrete (see Algorithm 4.1);
      else
        split continuous (see Algorithm 4.1);
      end if
      $H = H + 1$;
    end while
  end if
end while
output sequence $\hat{u}$ of set $i^* = \arg\max_{i \in \mathbb{A}} v(i)$

4.3 Discussion of OPHIS and SOPHIS

The following remarks apply for both OPHIS and SOPHIS. We first discuss the algorithms 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. Moreover, as previously mentioned, we set M to be odd, such that we can reuse the middle sequence when we have a continuous split. This works because we always consider the middle of the interval $\mu_{i,k}$ as the continuous action. We suggest taking M to be 3, the smallest feasible odd value.

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; if it is not, 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.

Both algorithms should be applied in receding horizon, by running them at each step, applying the first action from the returned sequence, and then repeating the procedure. Overall, a closed-loop control scheme is obtained. Moreover, we assume a setting in which simulating the system dominates computation, so to obtain a measure of the required computation, we need to look at the number of calls made to the dynamics f and the reward function ρ. For 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. In case of 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 reused center one) from step $k^\dagger$ to step $D_{i^\dagger}$.

Recall that the continuous input c is bounded and in $[0, 1]$. This unit interval can be obtained by rescaling a different bounded interval, so bound constraints on c are natively supported. Other constraints for the continuous action can also be implemented in the algorithm, by excluding subtrees of solutions that would violate the constraint. However, the analysis that follows in Section 5 will not take such constraints into account.

5 Analysis

In this section, we prove that the sub-optimality of the OPHIS and SOPHIS algorithms converges to 0 at certain rates with increasing budget n. Recall that the budget represents the number of system dynamics simulations that we are allowed to do. First, we will find an upper bound for the diameter of an arbitrary set i at some depth H in the tree, in Section 5.1. For this, we will look at the way the number of splits per dimension evolves. Then, in Section 5.2, as preliminaries for getting the convergence rates of the algorithms, we define a near-optimal tree that the algorithms focus on expanding, along with its branching factor. Afterwards, using this branching factor, we will establish a relation between the depth H and the budget n for OPHIS and SOPHIS, in Sections 5.3 and 5.4, respectively. Each of these relations, by replacement in the diameter formula, leads to convergence rates: for the OPHIS algorithm in Theorem 11, and for SOPHIS in Theorem 13.

It is important to note that we follow similar steps as [**?**], but we heavily adapted them to include the fact that we also have discrete splits. The OPC and SOPC algorithms in [**?**] only deal with continuous splits, and do not have such an intricate space of expansions. We also have fully new questions that did not arise before, such as the difference between the number of discrete split dimensions D and the number of continuous ones C.

5.1 Splitting behavior and upper bound for diameter

First, we define the depth of a given set as being the sum of continuous and discrete splits made in order to reach the set. The number of discrete splits for a set will be D, and the total number of continuous splits is denoted by h. Therefore:

$$H = h + D \tag{15}$$

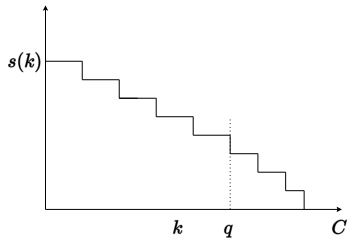

Fig. 4. An example of the splitting function $s(k)$

First, we will derive an upper bound for the diameter. We will focus in the beginning on the number of continuous splits, h. Let us define the continuous splitting function $s(k)$, which is the number of continuous splits per dimension k. An example of this function is given in Figure 4. This function is decreasing with at most 1 at each k, as we will see next. There will be ranges of dimensions for which the splitting function is the same. Quantity q represents the number of ranges at the end, that have a smaller length than the rest.

Assumption 4. $M\gamma > 2$.

This assumption is easy to satisfy, since M is at least 3, so as to reuse the center sequence, and generally, we would use values for γ of at least 0.7. Smaller values for γ determine a very short-horizon, almost myopic objective that will likely not work in most problems.

In what follows, we take a better look at the s-ranges.

Lemma 5. *a) The first k in a constant s-range is preferred for refinement to later dimensions in the same range. b) $s(k)$ decreases in steps of at most 1.*

Apart from the proof of Lemma 10, which was already given in [**?**], the proofs for all theorems and lemmas, including that of Lemma 5, are given in the supplementary material at `http://busoniu.net/files/papers/sophis_suppl.pdf`.

Next, we need a better understanding of how the s ranges look. For this, we will denote the lengths of the ranges with $\tau_0, \tau_1, ..., \tau_N$, with N being the last range, which is infinitely long and for which $s = 0$, meaning that there has been no continuous split for those dimensions. We will seek upper and lower bounds on these range lengths.

Lemma 6. *For any set, we have:*

$$\begin{cases} \tau_0 \leq \tau^* \\ \tau_j \in \{\tau^* - 1, \tau^*\}, \quad 1 \leq j < N - q \\ \tau_j < \tau^* - 1, \quad N - q \leq j < N \\ \tau_N = \infty \end{cases} \tag{16}$$

where $\overline{\tau} = \frac{\log(M)}{\log(1/\gamma)}$ and $\tau^* = \lceil \overline{\tau} \rceil$. This means that other than the first and last q ranges, all s-ranges have either the length τ^* or $\tau^* - 1$.

Next, we look at the difference between C and D, denoted G, which is nearly constant.

Lemma 7. *Let $\overline{G}$ be the smallest positive integer j, for which $L_\rho \frac{1-(\gamma L_f)^j}{1-\gamma L_f} > \frac{\gamma^j}{1-\gamma}$. G increases from 0 to $\overline{G}$ in the beginning, and then it always oscillates between $\overline{G}$ and $\overline{G} - 1$.*

This is important in our analysis, where we will use upper and lower bounds on C, and since G is roughly constant, bounds on C will translate to bounds on D.

Overall, Lemmas 5, 6, and 7 are important in the following way for determining the upper bound on the diameter. Since the contribution to the diameter of a continuous dimension depends on the interval length $a_k = M^{-s(k)}$, we need to find a lower bound on $s(k)$ for a fixed h. Knowing how the s-ranges look like from Lemma 6, we can get a lower bound on s for a fixed C, followed by both upper and lower bounds on C. This will eventually give us an overall lower bound on s, as a function of h and of the lower bound on C. With the help of Lemma 7, this translates to a function of h and the lower bound on D, which can then be utilized in the diameter formula. Detailed proofs are in the supplementary material mentioned above. Overall, this results in the following upper bound for the diameter:

Lemma 8. *For some positive constants c_1, c_2, c_3 and any set i at any depth $H = h + D$, $\delta(i) \leq c_1 \Big(c_2 + \sqrt{2h(\tau^* - 1)}\Big) \gamma^{\frac{\sqrt{2h(\tau^*-1)}\overline{\tau}}{\tau^*}} + c_3 \gamma^{\sqrt{2h(\tau^*-1)}}$.*

To understand this result, note that the first term in the summation dominates the second one. Also, due to the fact that $\gamma < 1$, the term $\gamma^{\frac{\sqrt{2h(\tau^*-1)}\overline{\tau}}{\tau^*}}$ asymptotically dominates $\sqrt{2h(\tau^* - 1)}$, causing the convergence to 0 of the diameter. Using the asymptotic notation $\tilde{O}$, we will have:

$$\delta(i) = \tilde{O}\Big(\gamma^{\frac{\sqrt{2h(\tau^*-1)}\overline{\tau}}{\tau^*}}\Big) \tag{17}$$

The $\tilde{O}$ notation is derived from the Bachmann–Landau notation, O. When the order of complexity of some algorithm is $O(f(y))\log(f(y))$, we say $\tilde{O}(f(y)) = O(f(y))\log(f(y))$, ignoring the logarithmic term. Here, $f(y)$ is any function; in (17), f is the power of γ and generic argument y is replaced by h.

Overall, we can observe that we have an exponential decrease in $\sqrt{h}$, as in [**?**]. However, the analysis is much more involved, as we must take into account the discrete splits as well.

5.2 Near-optimal tree and branching factor

In this subsection, we discuss the tree of near-optimal nodes and define its branching factor m.

Let us define the set of near-optimal nodes at depth H:

$$\mathcal{T}_H^* = \{i \text{ at } H \mid v(i) + \delta_H \geq v^*\} \tag{18}$$

This is a sub-tree of the full tree explained earlier, in Section 4.1, with an example in Figure 3. Both OPHIS and SOPHIS will refine nodes from the near-optimal tree. Now, denote the branching factor as m, as defined below.

Definition 9. The asymptotic branching factor is the smallest m such that $\exists \mathcal{C} \geq 1$ for which $|\mathcal{T}_H^*| \leq \mathcal{C} m^H$, $\forall H$, where $|\cdot|$ represents the cardinality of the set.

Branching factor m represents the complexity of the problem. In case of a smaller branching factor, the problem is simpler. The least complex problems correspond to $m = 1$, meaning that only the optimal path will be explored. It is important to note that m is not necessarily an integer. It is at least 1, but its maximum value depends on both M, the number of newly formed sets in case of a continuous split, and $p + 1$, the number of possible discrete actions, representing the number of newly formed sets in a discrete expansion. We define the maximum possible branching factor at any node as $Z = \max(M, p+1)$. Therefore $m \in [1, Z]$, different from the branching factors defined in [**?**] and [**?**], where there are either only continuous splits, or discrete ones.

5.3 Convergence rate of OPHIS

So far, we have discussed the way that the splits look, in the end getting a relation between the diameter and the sum of continuous splits. Now, we want to go further, to get a connection between the depth H and budget n, and link both to the near-optimality. It is important to remember that the depth H is the sum of h and D. Firstly, we provide a bound equal to a diameter on the near-optimality of the sequence returned that is explicitly available for use *a posteriori*, once OPHIS has run, in Lemma 10. Then, the relation between the depth and the budget, together with G allows us to use (17), which is stated in h. This, in turn, leads to a relation between the diameter and the budget, given in Theorem 11. In the end, taking into consideration also Lemma 10, we get a connection between the sub-optimality and the budget.

The a posteriori bound on the near-optimality of OPHIS is the following [**?**]:

Lemma 10. *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.

Such properties are standard for OP algorithms; both OPD and OPC ensure similar bounds. However, the proofs are different, as it can be seen in the supplementary material. Notably, a required property of increasing set values along continuous splits is not trivial for OPHIS, since we need to handle discrete and continuous splits at the same time. Note that we can also 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 [**?**].

Next, we give the relation between the budget and sub-optimality of the algorithm, an a-priori convergence rate guarantee.

Theorem 11. *For large budget n, we have:*

a) For $m > 1$: $v^* - v(i^*) = \tilde{O}\left(\gamma \sqrt{\frac{2\bar{\tau}^2(\tau^*-1)\log n}{\tau^{*2}\log m}}\right)$

b) For $m = 1$: $v^* - v(i^*) = \tilde{O}\left(\gamma^{n^{1/4}\frac{\bar{\tau}}{\tau^*}\sqrt{\frac{2(\tau^*-1)}{Z\mathcal{C}}}}\right)$

Note that both quantities reach 0 asymptotically. The convergence rate depends on the complexity m. We have a faster decrease for a less complex problem (smaller m) and vice-versa. This is similar to OPC in [**?**]. For $m = 1$, we have a convergence to 0 with $n^{1/4}$, as in OPC [**?**].

5.4 Convergence rate of SOPHIS

In this section, we want to find a lower bound on the depth H reached by SOPHIS given a budget n, which ensures we expand a node containing an optimal solution at that depth; and then combine the lower bound with Equation (17) to get a convergence rate of this second algorithm. Recall that H also takes into account the discrete expansions, being the sum of all splits needed for a certain set.

Lemma 12. *Given the budget n, define $H(n)$ to be the smallest depth for which the following inequality holds:*

$$\mathcal{C} Z H_{\max}^2(n) \sum_{H'=0}^{H} m^{H'} \geq n \tag{19}$$

Then, SOPHIS expands a node containing the optimal solution at depth $\underline{H} = \min\{H(n), H_{\max}(h)\}$, and the sequence returned is $\delta_{\underline{H}}$-optimal.

Recall that $Z = \max\{M, p+1\}$. Reference [**?**] proves this in the supplementary material. Their proof applies directly here, where instead of M, we have Z, and instead of h, we have H. Next, we find the convergence guarantees of the SOPHIS algorithm.

Theorem 13. *Consider the sequence $\hat{u}$ and the corresponding set i^* returned by SOPHIS. For large n:*
a) for $m > 1$, we take $H_{\max} = n^{\epsilon}$, with $\epsilon \in (0, 0.5)$ and

we have:

$$v^* - v(i^*) = \tilde{O}\left(\gamma\hat{}\left(\frac{\bar{\tau}}{\tau^*}\sqrt{\frac{(\tau^*-1)(1-2\epsilon)\log n}{\log m}}\right)\right)$$

b) for $m = 1$, we take $H_{\max} = n^{1/3}$, and we have:

$$v^* - v(i^*) = \tilde{O}\left(\gamma\hat{}\left(n^{1/6}\frac{\bar{\tau}}{\tau^*}\sqrt{2(\tau^*-1)\min\{\frac{1}{\mathcal{C}Z}, 1\}}\right)\right)$$

We can observe that for $m > 1$, the exponent is $\frac{1-2\epsilon}{2}$ times smaller than the one of OPHIS. This means that we lose a bit of convergence speed compared to the first method, but not a lot if ϵ is small. Again, we see that the branching factor m is a key factor in the rate, with a smaller m leading to a faster convergence to 0. For $m = 1$, the rate decreases with $n^{1/6}$, which is a bit of a loss in comparison to OPHIS, but still good, as it is an exponential of a power of the budget n.

One issue that we have to discuss for SOPHIS is the choice of $H_{\max}$. As in practice we do not know the actual branching factor m, we cannot set $H_{\max}$ "optimally", i.e. per the rule that for $m = 1$, $n^{1/3}$ should be used, and for $m > 1$, n^{ϵ} with $\epsilon \in (0, 0.5)$ is needed. A practical solution is to first take $H_{\max}$ as $n^{1/3}$, in the hope of a simple problem. In case the problem turns out to be more complicated, this leads to a slower convergence rate, but the algorithm remains valid.

This concludes our a-priori convergence rate analysis for the two algorithms. The upper bound for the diameter was a key step in establishing a relation between the budget n and the convergence rate. We were then able to prove that the sub-optimality of both OPHIS and SOPHIS converge to 0 at different rates. This proves the performance of our algorithms. Next, we give some simulation results for both of them.

6 Simulation results

This section discusses the results of applying the algorithms for a hybrid-input problem. A two-link robot arm example is presented, with simulation results given for both OPHIS and SOPHIS. In a previous paper [**?**], OPHIS was also applied to a simple, two-tanks problem, where it succeeded in recovering existing results from the literature.

The robot arm has 2 joints, one continuously controlled and one which can either have a brake set or not [**?**]. The state vector is represented by the two angles and the angular velocities $\mathbf{x} = [\theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2]$. 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 [**?**], together with the model, and they are not presented here, due to lack of space. The parameters used are the same as in [**?**]. 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 using the Euler method, with 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:

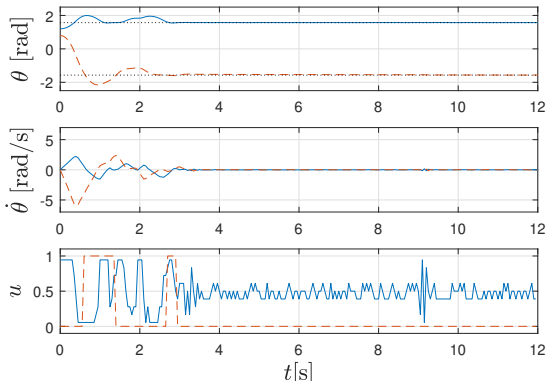

Fig. 5. OPHIS: Evolution in time of the states and actions

$$\rho(\mathbf{x}_k, u_k) = (10 - |x_{1_{k+1}} - x_{1_f}| - |x_{3_{k+1}} - x_{3_f}|)/10 \quad (20)$$

where x_{k+1} is the state at the next step. We use a non-differentiable reward to show that the algorithm works in this case.

The following values were used for both algorithms: $M = 3$, budget $n = 5000$, $L_f = 0.8$, $L_\rho = 1.2$, $\gamma = 0.8$. The starting position is $\mathbf{x}_0 = [1.2, 0, 0.8, 0]$. The desired final state is $\mathbf{x_f} = [\pi/2, 0, -\pi/2, 0]$. In addition, for SOPHIS, we take $H_{\max} = n^{0.35}$. Recall that both algorithms are applied in receding horizon, and the simulation is done over 12 seconds.

The results for OPHIS can be seen in Figure 5, 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. The last subplot presents the evolution in time of the 2 actions: with blue continuous line c and with red dashed line d. 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 in steady state.

Figure 6 presents the results using SOPHIS. with the subplots structured as in Figure 5. As we can see, the angles reach their reference values (in black dotted line) much faster than with OPHIS. This is also the case for the angular velocities, which can be seen stabilizing around 0 much quicker. We notice fewer switches of discrete input d than when using OPHIS. Overall, SOPHIS gives better empirical results.

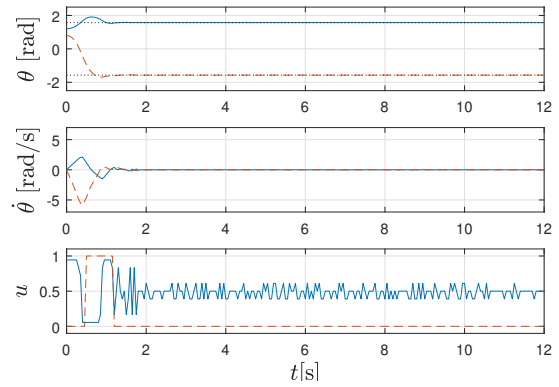

Fig. 6. SOPHIS: Evolution in time of the states and actions

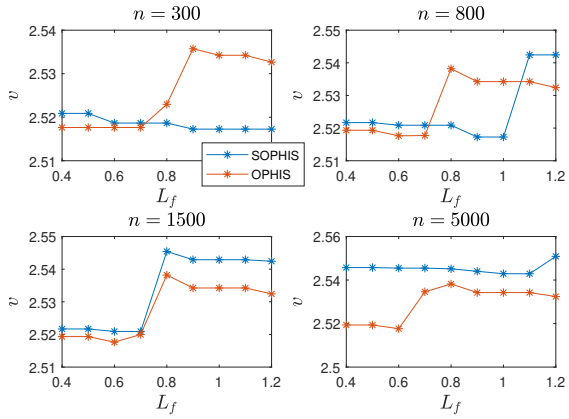

Fig. 7. Discounted returns of the two algorithms for several L_f and n values

A comparison with [**?**] 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. Still, as previously stated, for the two-tanks system presented in [**?**], we obtain the same results as in the literature [**?**].

We also compare OPHIS and SOPHIS in terms of discounted return, for several L_f values and different budgets. The comparison is presented in Figure 7. For small budgets, OPHIS is better since a focused search on one value of the Lipschitz constant likely makes more sense when computation is limited. When the budgets are larger, SOPHIS starts actually exploiting its potential by spending the extra computation to expand sets for many possible values of the Lipschitz constant, which in effect is similar to automatically finding the best value of this constant (for set selection). So we see that it starts dominating OPHIS. In addition, as presented before, for a large budget, SOPHIS manages to bring the states to the reference value much quicker. Furthermore, we observe flatter curves in Figure 7 for SOPHIS. This means that we have a smaller sensitivity to the value of L_f.

In the supplementary material, a comparison is made between the two algorithms when a disturbance is present.

In terms of computational time, for a budget $n = 1000$, the execution time of one open-loop run is approximately $0.4s$. If we increase the budget to $n = 2000$, the execution time becomes approximately $2.2s$, while a budget $n = 5000$ implies $9.7s$. The simulations have been run in Matlab, which is therefore not yet ready for real-time control; however, C would accelerate the algorithm, see [**?**] for an example of real-time control using SOPC implemented in C and for a larger discussion on the topic.

7 Conclusions

Two algorithms are proposed for systems with with both continuous and discrete actions: Optimistic Planning for Hybrid-Input Systems (OPHIS) and Simultaneous OPHIS. While OPHIS refines one set per iteration, using computed upper bounds for the returned values, SOPHIS simultaneously refines several sets per iteration. This eliminates the dependency of the Lipschitz constant in the set selection. Theoretical guarantees are given for both methods, where convergence rates are proven, depending on a measure of problem complexity. In simulations, the algorithms are successful for a two-link robot arm problem, with SOPHIS proving to be the better choice when a larger budget is available.

In future work we aim to study stability properties [**?**,**?**] and perhaps even exploit stability in order to achieve tighter bounds.