Real-Time Simultaneous Optimistic Planning for Hybrid-Input Nonlinear Optimal Control

Elvin Pop
Department of Automation
Technical University of Cluj-Napoca
Romania
elvin.pop@aut.utcluj.ro

Ioana Lal
Department of Automation
Technical University of Cluj-Napoca
Romania
ioanalal04@gmail.com

Lucian Buşoniu
Department of Automation
Technical University of Cluj-Napoca
Romania
lucian.busoniu@aut.utcluj.ro

***Abstract*—Simultaneous Optimistic Planning for Hybrid-Input Systems (SOPHIS) is a powerful method for the near-optimal control of nonlinear systems with hybrid – continuous and discrete – inputs, which works by iteratively splitting sets of input sequences. The generality of SOPHIS however comes at high computational costs that are often untenable in real-time control, especially for fast unstable systems. We introduce two modifications that make SOPHIS more suitable for real-time control: running it on a separate machine, over multiple sampling periods, while applying several inputs to the system during this time; and parallelizing the algorithm by splitting several sets simultaneously across multiple threads. Experiments investigate two parallelization schemes, the impact of thread count on the execution time, and the influence of the prediction horizon and budget; the latter on a real-life fast unstable system, a rotary inverted pendulum. In the experiments, the discrete input controls the quantization accuracy of the control action sent to the system.**

***Index Terms*—nonlinear optimal control, optimistic planning, real-time control, parallelization, predictive control**

I. Introduction

This paper focuses on real-time near-optimal control of nonlinear hybrid-input systems, which have both a continuous and a discrete input. Such problems are often encountered in robotics [3], the automotive industry [20], industrial multi-tank systems [19] or hydraulic systems [17].

In prior work [13], we proposed a method for hybrid-input optimal control called Simultaneous Optimistic Planning for Hybrid-Input Systems (SOPHIS). Optimistic planning [16] is a general flavor of nonlinear model-predictive control (MPC) [10] that works for a large class of non-linear, non-differentiable dynamics and non-quadratic, non-differentiable objectives. SOPHIS partitions the set of input sequences into subsets organized in a tree structure. It splits at each iteration several such sets at a range of tree depths, chosen so that the optimal solution(s) are likely to be in these sets. At the end, a near-optimal sequence of inputs is returned, of which the first input is applied, and then the procedure repeats in receding horizon. The suboptimality of the sequence decreases at well-characterized rates as a function of the number n of calls to the system dynamics; and SOPHIS exhibits good performance in simulations [13].

This work was been financially supported by DECIDE, project no. 57/14.11.2022 funded under the PNRR I8 scheme by the Romanian Ministry of Research, Innovation, and Digitisation.

The main issue in applying SOPHIS is its high computational cost, since its generality means that at each discrete time step it must examine many subsets using a large budget n (in general exponential in the horizon, and often in the tens of thousands in practice). Conventionally, all this computation must finish in a negligible portion of the sampling period, which is often unfeasible, especially for fast unstable nonlinear systems. On the one hand, the fast and unstable nature of such systems means that sampling periods must be small (e.g. on the order of ms for a drone [12]), so little time is available to compute; and on the other hand, the nonlinearity sometimes means that longer-horizon sequences must be found, leading to larger cost n (consider e.g. an inverted pendulum [15] that must be swung several times before being stabilized, or other pendulum-like systems like ballbots [7]).

Our objective in this paper is to mitigate these issues by designing a version of SOPHIS more suitable for real-time control. To this end, we make two main changes in the method. First, (i) we allow computation to run for several sampling periods, applying in the meantime a subsequence consisting of several inputs from the sequence previously returned by the algorithm. Each next SOPHIS call is done with the predicted state at the end of the current subsequence. SOPHIS is run on a different machine than the one that interacts in closed loop with the system, thereby separating computation from control, and the two machines communicate over a network. Second, (ii) we parallelize SOPHIS so that several sets are split at once, on different threads.

Procedure (i) is natural in MPC, and as early as 1999, [18] intermittently applied sequences of actions and shifted the MPC horizon by the lengths of the applied sequences. Since then, many other variations of this idea have been proposed [1], [5], [6], [9], [11], [21], sometimes including computational delay in the design [4], [8], but rarely separating control from computation like we do. One possible exception is [21], which uses a distributed embedded computation scheme. The closest approach to ours is [22], where (i) was applied to discrete-input OP. A key difference between the present approach and all these previous methods is that the parallelization approach is unique to SOPHIS, since this algorithm provides a natural way in which to distribute computation: over the collection of sets that are split at each iteration.

For a rotary pendulum example in which the discrete input

is the quantization accuracy with which the continuous input is sent to the system, we experimentally investigate several questions: the difference between two parallelization schemes, the execution time as a function of the number of threads, the influence of the prediction horizon, and of the budget. The latter two experiments are run on the real-life system.

Next, Section II outlines SOPHIS, while Section III introduces the adjustments made in order to use it in real-time. Experimental results are given in Section IV, and Section V concludes.

II. Background

This section summarizes Simultaneous Optimistic Planning for Hybrid-Input Systems (SOPHIS) [13]. The algorithm works for discrete-time, nonlinear systems with hybrid inputs:

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

where the input $u_k = [c_k, d_k]^T$ has both a continuous scalar component c_k and a discrete part $d_k \in \{0, 1, ..., p\}$. A reward $r_{k+1} = \rho(x_k, u_k)$ is assigned to every state-action pair, or equivalently to any transition from x_k to x_{k+1} using input u_k. Given an infinitely-long sequence of actions $\mathbf{u}_\infty = (u_0, u_1, ...)$, discount factor $\gamma \in (0, 1)$ and reward function $\rho(x, u)$, define the infinite-horizon discounted value:

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

The purpose is to find an optimal value $v^* = \sup_{\mathbf{u}_\infty} v(\mathbf{u}_\infty)$ and any sequence $\mathbf{u}^*_\infty$ that would achieve it. The system must satisfy [13]:

Assumptions. *(i)* $c_k, r_k \in [0, 1]$.
(ii) 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* $c, c' \in [0, 1]$*:*

$$\|f(x, [c, d]^T) - f(x', [c', d]^T)\| \leq L_f(\|x - x'\| + |c - c'|)$$
$$|\rho(x, [c, d]^T) - \rho(x', [c', d]^T)| \leq L_\rho(\|x - x'\| + |c - c'|)$$

(iii) $\gamma L_f < 1$

In Assumption (i) the unit interval is taken for convenience, and other intervals can simply be rescaled. Assumption (iii) means that together with discount factor γ, the dynamics become contractive.

SOPHIS iteratively splits sets, each set an infinitely-long product of pairs (μ_k, σ_k):

$$\mathbb{S}_i = \prod_{k=0}^{\infty} (\mu_{i,k}, \sigma_{i,k}) \tag{3}$$

where μ_k represents the interval in which continuous action c_k lies and σ_k is either the full set of possible discrete actions or just one definite value from the set. We refer to index k as the dimension (or step). The sets are organized in a tree structure, from a root node, which has all intervals μ equal to $[0, 1]$ and all $\sigma = \{0, 1, ..., p\}$. Iteratively, well-chosen leaf nodes on the tree are expanded and their children are added to the tree. There are two types of splits: continuous or discrete. A set i also has two attributes, C_i and D_i, denoting the number of continuously and discretely split 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 $k \geq D_i$, $\sigma_{i,k} = \{0, 1, ..., p\}$. Note that from the design of the algorithm, $D_i \geq C_i$ at all times, for any node; and at the root $C_0 = D_0 = 0$.

A continuous split is done along any dimension $k \leq C_i$, by adding M children to the tree. The children inherit all (μ, σ) pairs from their parent, except μ_k. This interval will be split into M subintervals, one for each child. A discrete split, on the other hand, is always done along dimension $k = D_i$. It entails adding $p + 1$ children to the tree, which inherit all existing (μ, σ) pairs and in addition will now have σ_{D_i} as a definite action, one value for each child, instead of the full set of possible actions (which was associated to the parent).

We have explained how the splits work. Now, we move on to the selection of the nodes to split. First, define the depth H of a node (set) i in the tree as the number of expansions done starting from the root to get to that node. Note that the number of continuous expansions up to node i is not C_i, but $h_i = \sum_{k=0}^{C_i - 1} s(k)$, where $s(k)$ is the number of splits done on dimension k. H_i is now $h_i + D_i$. The SOPHIS algorithm splits several nodes per iteration: at each depth, it splits an un-expanded node with the highest value v among the nodes at that depth. The value of a node is $v(i) = \sum_{k=0}^{D_i} \rho(x_k, [c_k, d_k]^T)$, where c_k is the center of interval $\mu_{i,k}$. Iterations continue for as long as budget n is still available, where n is the number of calls to the dynamics and reward functions, f and ρ. In order to prevent an iteration from running indefinitely, a maximum value $H_{\max}(n)$ is given to the algorithm as an input; it represents the maximum depth up until which to expand nodes.

Now, for each node selected to be split, we must choose the dimension and type of expansion. For this, define the diameter of a node:

$$\delta(i) = L_\rho \sum_{k=0}^{D_i - 1} a_{i,k} \gamma^k \frac{1 - (\gamma L_f)^{D_i - k}}{1 - \gamma L_f} + \frac{\gamma^{D_i}}{1 - \gamma} \tag{4}$$

where $a_{i,k}$ is the length of the interval $\mu_{i,k}$. We aim to reduce the diameter of the node, so we look at the dimension which has the greatest contribution to δ, denoted $k^\dagger$.Therefore, 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 split discretely, along $k = D_{i^\dagger}$. Otherwise, we have a continuous split, along dimension $\min(k^\dagger, C_{i^\dagger})$.

At the end, the algorithm outputs finite sequence $\hat{\mathbf{u}}$, corresponding to the node with the largest value v among all nodes. SOPHIS runs in open loop, and it is meant to be applied in receding horizon. This means that in principle at each step in time t, the algorithm is applied starting from state x_t. It then applies the first input from the returned sequence to the real system. SOPHIS is then run again in open-loop, starting from resulting state x_{t+1} and so on. A closed-loop control scheme is thus achieved.

Regarding tuning parameters, the first is the split factor M, and in practice, it is suggested to take $M = 3$. The Lipschitz constants L_f, L_ρ are usually not known in practice, so we treat them as tuning parameters, keeping in mind that Assumption (iii) requires $\gamma L_f < 1$. Lastly, for the maximum depth $H_{\max(n)}$, a good starting point is $n^{1/3}$. Overall, [13] proves that the near-optimality of SOPHIS converges to 0 with increasing budget n, at well-characterized rates. We discussed SOPHIS briefly; for more insight and detail, see [13].

Next, we move to the main contribution of this paper: the real-time version of SOPHIS.

III. Real-Time SOPHIS

This section discusses the adaptations made to SOPHIS in order to use it in real time for fast systems. Two adjustments are necessary, and they are discussed in turn next.

A. Separating Control From Computation

Conventional receding-horizon control dictates that at each step t, SOPHIS is run from x_t, only the first action u_t of the returned sequence is applied, and then SOPHIS runs again from x_{t+1}. For this to be feasible, the algorithm must run in significantly less time than T_s, the sampling time of the system. This is however often impossible, especially with fast, unstable nonlinear systems, for which T_s must be small; examples include drones [12], pendulums [15], or ball-balancing robots similar to them [7] etc. The required budget n also grows large for complex nonlinear dynamics, making the problem worse.

To delve into the first adaptation, note first that we run control loop on one machine, and SOPHIS on another. The two machines communicate over a network, with the control machine acting as a server and the SOPHIS machine as a client. Moreover, similarly to [22], we propose to apply a subsequence of the sequence returned by SOPHIS to the system. Denote the sequence returned as $\mathbf{u}_{T_m}^m$, meaning that it has length T_m and was computed by the m^{th} run of the algorithm. While the control machine applies to the system the subsequence containing the T inputs at the start of u_{T_m}, a different machine runs SOPHIS again, starting however from a *predicted* state $\hat{x}_{t+T}$. This predicted state is found by simulating the system dynamics from the current state x_t, using the first T inputs from the sequence. Then, by the time the control machine finishes with applying the inputs, the computation machine has already finished a new open-loop SOPHIS run and returns a new output sequence $\mathbf{u}_{T_{m+1}}^{m+1}$. Keep in mind that T_m and T_{m+1} might not be equal. This procedure is repeated as long as the experiment runs; a pseudocode of it is given in Algorithm 1.

In the pseudocode, SOPHIS(x, n) represents one run of the SOPHIS algorithm, starting from state x with budget n. In order to choose T, we must keep in mind that the time needed for the algorithm to run with budget n must be smaller than TT_s, where T_s is the sampling time. Thus, if we choose a small value for T, we might end up with too little time to compute an output sequence. On the other hand, $\forall m \geq 0$, we must have $T_m \geq T$, so T has to be small enough for SOPHIS to be able to output a sequence of at least this length. An important remark is that we do not take into consideration the mismatches between the model and the real system (which might occur due to model uncertainties, perturbations etc), so a too large sub-sequence length T could result in large discrepancies between the real state and the predicted one, and therefore poor performance. Taking these into consideration, a balance between small and large T must be found. Later on, we provide experiments that show its effect.

Algorithm 1: Real-time SOPHIS

Input: state x_0, budget n, sub-sequence length T

SOPHIS Machine: $\mathbf{u}_{T_0}^0$ =SOPHIS(x_0, n);
Control Machine `thread1()`
  initialize time step $t = 0$, $m = 0$;
  while experiment still running **do**
    `thread2(`$\mathbf{u}_T^m$`)`;
    read current state x_t;
    find predicted state $\hat{x}_{t+T}$ from x_t using $\mathbf{u}_T^m$;
    $\mathbf{u}_{T_{m+1}}^{m+1}$=`sophis(`$\hat{x}_{t+T}$`)`;
    update $t = t + T$, $m = m + 1$
Control Machine `thread2(`$\mathbf{u}_T$`)`
  apply actions from $\mathbf{u}_T$ once every T_s
SOPHIS Machine `sophis(`$\hat{x}_{t+T}$`)`
  return $\mathbf{u}$ =SOPHIS$(\hat{x}_{t+T}, n)$;

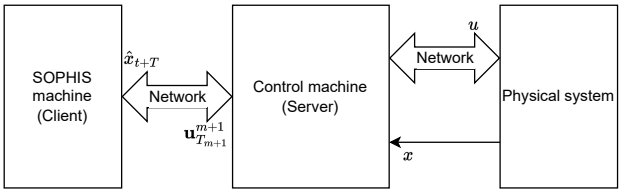

Fig. 1: Overall control scheme

Of course, the control scheme also has a third component beside the two machines: the physical system, see Figure 1. The physical system is connected to the control machine, which reads the values of the states and sends control inputs, possibly also over a network (shown for u in the figure). On the other side, the control machine (server) communicates with the SOPHIS machine (client), which runs the adapted algorithm. SOPHIS receives the predicted state $\hat{x}_{t+T}$ from the control machine and sends back the input sequence $\mathbf{u}_{T_{m+1}}^{m+1}$.

An example of the timings for a few steps in Algorithm 1 is shown in Figure 2. We only show a time interval starting at step t and the m^{th} call to SOPHIS. We notice how every T steps, the SOPHIS algorithm is called, and once every step t, an action from the returned sequence is sent to the system, denoted by $u_{T,j}^l$ to represent the j^{th} action of sequence $\mathbf{u}_T^l$ returned by the l^{th} call to SOPHIS.

B. Parallelization

A second adaptation performed to apply SOPHIS in real-time is to parallelize each of the algorithm's iterations. More

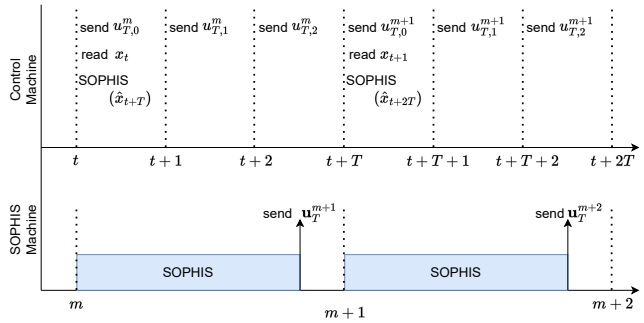

Fig. 2: Example of timings for $T = 3$

explicitly, instead of going sequentially through each depth on the tree with unexpanded nodes and splitting them one by one, we now split several nodes at a time, using one thread for each, where each chosen node still has the largest v at that depth. Note that each thread will expand a node from a different depth. The number of threads is given as an input parameter to the algorithm, and is denoted by τ. Therefore, given that at some point in an iteration we are at depth H, we will expand nodes at depths $H, H+1, H+2, ..., H+\tau-1$, wrapping around to the smallest depth with unexpanded nodes as soon as we exceed $H_{\max}(n)$.

Once all the entire collection of τ notes have been expanded, two options arise for the depth at which to start expanding in the next round: $H+1$ or $H+\tau$. The choice between the two needs to be made for each different system. Rule $H+1$ is expected to work better in systems where there is generally one trajectory that dominates all others in terms of rewards. In this case, we expect that the highest-value node at depth $H+1$ will be a child of the highest-value node at depth H. The $H+1$ strategy handles this case well, while the $H+\tau$ strategy does not, because the next iteration begins at depth $H+\tau$, which means the children that would have been expanded by the original algorithm will be skipped. The other variant, $H+\tau$, is expected to work better for systems where it is better to search many sequences in a breadth-first manner. The two versions will be compared empirically later on.

The new parallelized SOPHIS procedure is presented in Algorithm 2, where only the relevant details are included. The dimension selection, discrete/continuous split type selection and splitting procedure remain the same as in SOPHIS. Notation $\mathbb{A}$ refers to the full collection of sets (leaves of the tree), whereas $\mathbb{A}_j$ are the leaves at depth j. At line 8, the selection between $H+1$ and $H+\tau$ is based on which variant of the algorithm one wants to use.

With the adaptations presented above, SOPHIS can now be used in real-time experiments. Next, results are presented for the control of an rotary inverted pendulum.

IV. Results

This section presents and discusses the results of real-time SOPHIS with the adaptations of Section III.

The algorithm is used to control a rotary inverted pendulum system which has 4 states: $x = [\theta, \dot{\theta}, \alpha, \dot{\alpha}]$, where θ is the angle of the rotary arm of the system, and α is the angle of the pendulum itself (both are measured in rad), see [2] for the dynamics. The continuous input c is the usual voltage to be applied to the pendulum, while the discrete input d is the quantization level of c. We apply a ternary hierarchical quantization scheme that matches the iterative interval refinement in SOPHIS, where a larger discrete value corresponds to more hierarchical levels and thus a more precise quantization, see [14] for details. This is motivated by reducing network usage on the input channel between the control machine and the system.

Algorithm 2: Parallelized SOPHIS

Input: τ

1 **while** budget still available **do**
2 H = smallest depth with unexpanded nodes;
3 **if** $H \geq H_{\max}(n)$ **then**
4 stop and exit the loop;
5 **else**
6 **while** $H < H_{\max}(n)$ **do**
7 expand τ sets $i^\dagger_j = \arg\max_{i \in \mathbb{A}_j} v(i)$, one on each thread; with $j \in \{H, H+1, ..., H+\tau-1\}$;
8 $H = H+\tau$ or $H = H+1$

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

Next, Section IV-A covers the effect of the two parallelization variants and of the thread count on the performance of the algorithm; and Section IV-B examines the execution time as a function of the thread count. For these two experiments, simulations are sufficient since these algorithm behaviors are less influenced by mismatches between the simulation and the real system. For the next experiments, we move to the real rotary pendulum. Namely, Sections IV-C and IV-D respectively cover the effect of the prediction horizon and of the budget on the performance. Section IV-D also includes an example trajectory of the pendulum controlled with real-time SOPHIS.

A. Parallelization Variants

As previously stated, two parallelization variants were considered, $H+1$ and $H+\tau$. We ran simulations of these variants with the number of threads τ ranging from 1 to 16. The comparison between the two variants can be seen in Figure 3, where the vertical axis R gives the mean of the rewards at each sampling period. The $H+1$ variant resulted in a much better mean reward than the $H+\tau$ variant.[1] This leads us to the conclusion that the system is of the type where the optimal sequences are clearly distinguishable on the tree, which the $H+1$ variant handles better, as discussed in Section III-B.

Note that in the $H+1$ variant the performance is good irrespective of the number of threads τ. In contrast, the $H+\tau$

[1]Even though the differences between the rewards are numerically small, they correspond to large differences in practical system performance.

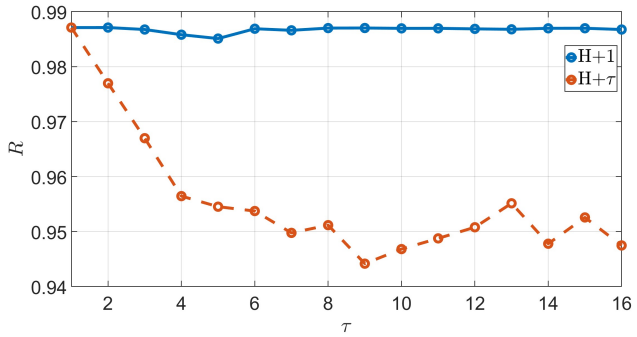

Fig. 3: Mean reward for the two parallelization variants and a varying thread count τ (simulation)

variant skips τ nodes each iteration, and so for larger τ the search is taken further away from good sequences.

B. Execution Time

The algorithm was simulated with a varying number of threads. We measured the time required for the algorithm to return one sequence of inputs using a large budget of $n = 20000$. This measurement was done on a laptop with an AMD Ryzen 7 5700U processor, which has 8 cores and 16 threads. Running this algorithm on different processors may give different results, but we expect the overall dependence of the execution time on τ to stay the same. This dependence can be seen in Figure 4. As τ grew, the execution time dropped until reaching a lower limit of approximately 17ms. This saturation could be the result of either operating system limitations on processor usage, or limitations of the parallelization scheme.

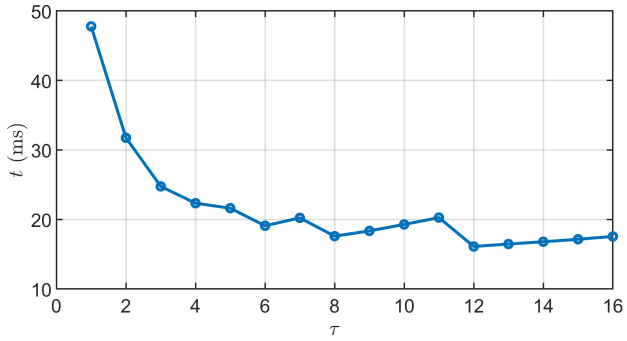

Fig. 4: Evolution of execution with the thread count τ (simulation)

Another important consideration related to time constraints is the sampling period of the control loop. The algorithm must complete within T sampling periods while also leaving room for the latency in communication between the client and the server. Recall that T is the length of the applied subsequence, and therefore also the prediction horizon for the next call to SOPHIS. For example, in the case of the rotary pendulum system, a sampling period of 50ms is small enough for the algorithm to stabilize the system. With a budget of $n = 20000$ and 1 thread, the algorithm runs in 45ms, which, given that the communication latency is smaller than 5ms satisfies the time constraint.

C. Prediction Horizon

In case the required budget to control a given system results in an execution time greater than the sampling period of the system, or the communication latency causes the control loop to violate the time constraint, we must increase the horizon T. We therefore analyzed the effect of T on the performance of the algorithm. From here on, we report real-life experiments with the rotary pendulum.

We ran the experiment with two different strategies. In the first, we used a fixed budget of $n = 30000$ regardless of T, and in the second, we used the maximum budget possible while still satisfying the time constraint: thus, the maximum budget grows with T. The result can be seen in Figure 5. Overall, the performance of the algorithm decreases with the horizon T. This is expected because as we increase T, modeling errors start to have a larger effect. In the case of the rotary pendulum, both the fixed-budget and max-budget strategies showed similar results, implying that modeling errors are more important than the choice of strategy. This is most likely due to the unstable nature of the system, making its behavior difficult to predict accurately over large periods of time. We expect the max-budget strategy to perform better when a sufficiently accurate model of the system is available.

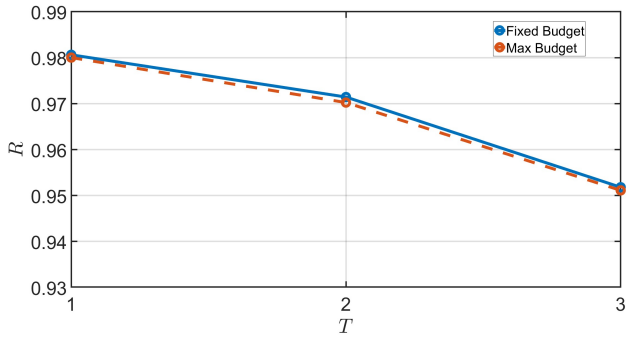

Fig. 5: Evolution of performance with the length T of the prediction horizon (real-life experiment)

D. Budget

In order to analyze the effect of the budget n on the algorithm performance, we ran experiments on the real system with n taking several values in the range $[1000, 30000]$. The result can be seen in Figure 6. We observed that $n = 10000$ is sufficient for good performance, and increasing n above this value did not result in better performance. It is important to note that (up to noise) larger budgets do not decrease performance.

An example of an experiment run on a real rotary pendulum using real-time SOPHIS can be seen in Figure 7. Here, the algorithm was used with a sampling time of $T_s = 40$ms,

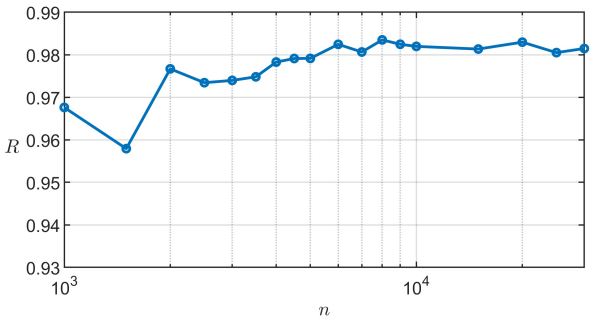

Fig. 6: Evolution of performance with the budget n (real-life experiment)

budget $n = 20000$, prediction horizon $T = 1$, a thread count $\tau = 8$, and (note that for $\tau = 8$ the algorithm takes around 20 ms in Figure 4). Although the results in Fig. 7 are those of a single run, they are repeatable and typical of the algorithm's performance on this system. In the trajectory, we observe some oscillations in θ. This is most likely due to the discretization of the input applied to the system, along with system's inherent instability and the effects of measurement noise on the algorithm. A video demonstration is available at http://rocon.utcluj.ro/files/rtsophis_rotpend.mp4.

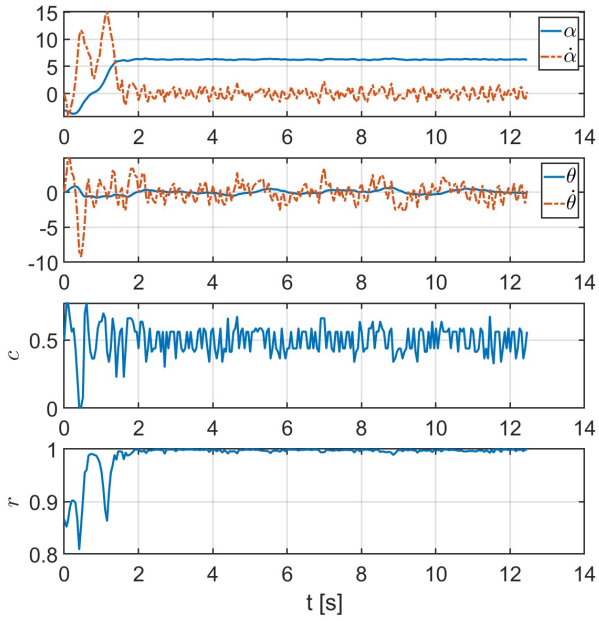

Fig. 7: Real-time SOPHIS: Evolution in time of states, actions, and rewards

V. Conclusions

We adapted SOPHIS – an algorithm for near-optimal control of systems with hybrid inputs – for real-time usage, and demonstrated good performance in simulations and experiments on a fast, unstable rotary pendulum. Future work may focus on analyzing the impact of these real-time adaptations on the near-optimality guarantees, starting from ideas in [22]; on applying the same ideas to the non-simultaneous method OPHIS; or on real-time control in more challenging problems such as acrobatic drone flight.

References

[1] L. G. Bleris and M. V. Kothare, "Real-time implementation of model predictive control," in *Proceedings of the 2005 American Control Conference*. IEEE, 2005, pp. 4166–4171.

[2] L. Buşoniu, E. Páll, and R. Munos, "Discounted near-optimal control of general continuous-action nonlinear systems using optimistic planning," in *Proceedings of the 2016 American Control Conference (ACC)*. IEEE, 2016, pp. 203–208.

[3] 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.

[4] W.-H. Chen, D. Ballance, and J. O'Reilly, "Model predictive control of nonlinear systems: Computational burden and stability," *IEEE Proceedings on Control Theory and Applications*, vol. 147, no. 4, pp. 387–394, 2000.

[5] M. Diehl, H. G. Bock, J. P. Schlöder, R. Findeisen, Z. Nagy, and F. Allgöwer, "Real-time optimization and nonlinear model predictive control of processes governed by differential-algebraic equations," *Journal of Process Control*, vol. 12, no. 4, pp. 577–585, 2002.

[6] M. Diehl, R. Findeisen, and F. Allgöwer, "A stabilizing real-time implementation of nonlinear model predictive control," in *Real-Time PDE-Constrained Optimization*. SIAM, 2007, pp. 25–52.

[7] P. Fankhauser and C. Gwerder, "Modeling and control of a ballbot," B.S. thesis, Eidgenössische Technische Hochschule Zürich, 2010.

[8] R. Findeisen and F. Allgöwer, "Computational delay in nonlinear model predictive control," in *Proceedings International Symposium on Advanced Control of Chemical Processes*, Hong Kong, 2004, pp. 427–432.

[9] P. J. Gawthrop and L. Wang, "Intermittent predictive control of an inverted pendulum," *Control Engineering Practice*, vol. 14, no. 11, pp. 1347–1356, 2006.

[10] L. Grüne and J. Pannek, *Nonlinear Model Predictive Control: Theory and Algorithms*. Springer, 2011.

[11] L. Jin, R. Kumar, and N. Elia, "Model predictive control-based real-time power system protection schemes," *IEEE Transactions on Power Systems*, vol. 25, no. 2, pp. 988–998, 2009.

[12] V. Kangunde, R. S. Jamisola Jr, and E. K. Theophilus, "A review on drones controlled in real-time," *International Journal of Dynamics and Control*, vol. 9, no. 4, pp. 1832–1846, 2021.

[13] I. Lal, I.-C. Morărescu, J. Daafouz, and L. Buşoniu, "Optimistic planning for control of hybrid-input nonlinear systems," *Automatica*, vol. 154, p. 111097, 2023.

[14] I. Lal, I.-C. Morărescu, J. Daafouz, and L. Buşoniu, "Near-optimal control of nonlinear systems with hybrid inputs and dwell-time constraints," *IEEE Control Systems Letters*, vol. 7, pp. 2455–2460, 2023.

[15] G. Medrano-Cersa, "Robust computer control of an inverted pendulum," *IEEE Control Systems Magazine*, vol. 19, no. 3, pp. 58–67, 1999.

[16] R. Munos, "The optimistic principle applied to games, optimization and planning: Towards foundations of Monte-Carlo tree search," *Foundations and Trends in Machine Learning*, vol. 7, no. 1, pp. 1–130, 2014.

[17] 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.

[18] E. Ronco, T. Arsan, and P. Gawthrop, "Open-loop intermittent feedback control: practical continuous-time GPC," *IEE Proceedings on Control Theory and Applications*, vol. 146, no. 5, pp. 426–434, 1999.

[19] 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*, vol. 5. IEEE, 1997, pp. 3495–3499.

[20] A. J. Van Der Schaft and J. M. Schumacher, *An introduction to hybrid dynamical systems*. Springer London, 2007, vol. 251.

[21] P. D. Vouzis, L. G. Bleris, M. G. Arnold, and M. V. Kothare, "A system-on-a-chip implementation for embedded real-time model predictive control," *IEEE Transactions on Control Systems Technology*, vol. 17, no. 5, pp. 1006–1017, 2009.

[22] T. Wensveen, L. Busoniu, and R. Babuka, "Real-time optimistic planning with action sequences," in *Proceedings of the 20th International Conference on Control Systems and Computer Science*, 2015, pp. 923–930.