Technical report 17-017

Optimistic planning with an adaptive number of action switches for near-optimal nonlinear control*

K. Máthé, L. Buşoniu, R. Munos, and B. De Schutter

* This report can also be downloaded via https://pub.bartdeschutter.org/abs/17_017.html

Optimistic Planning with an Adaptive Number of Action Switches for Near-Optimal Nonlinear Control

Koppány Máthé[a], Lucian Buşoniu[a], Rémi Munos[b], Bart De Schutter[c]

[a] *Department of Automation, Technical University of Cluj-Napoca, Romania (koppany.mathe@aut.utcluj.ro, lucian.busoniu@aut.utcluj.ro)*
[b] *Google DeepMind (munos@google.com)*
[c] *Delft Center for Systems and Control, Delft University of Technology, the Netherlands (b.deschutter@tudelft.nl)*

Abstract

We consider infinite-horizon optimal control of nonlinear systems where the control actions are discrete, and focus on optimistic planning algorithms from artificial intelligence, which can handle general nonlinear systems with nonquadratic costs. With the main goal of reducing computations, we introduce two such algorithms that only search for constrained action sequences. The constraint prevents the sequences from switching between different actions more than a limited number of times. We call the first method optimistic switch-limited planning (OSP), and develop analysis showing that its fixed number of switches S leads to polynomial complexity in the search horizon, in contrast to the exponential complexity of the existing OP algorithm for deterministic systems; and to a correspondingly faster convergence towards optimality. Since tuning S is difficult, we introduce an adaptive variant called OASP that automatically adjusts S so as to limit computations while ensuring that near-optimal solutions keep being explored. OSP and OASP are analytically evaluated in representative special cases, and numerically illustrated in simulations of a rotational pendulum. To show that the algorithms also work in challenging applications, OSP is used to control the pendulum in real time, while OASP is applied for trajectory control of a simulated quadrotor.

Keywords: optimal control, planning, nonlinear predictive control, near-optimality analysis.

1. Introduction

Optimal control problems arise in numerous areas of technology. Our focus here is on optimal control in discrete time, so as to maximize a discounted sum of rewards (negative costs). *Optimistic planning* (OP) techniques [1] solve this problem locally for any given state, by exploring tree representations of possible sequences of actions (control inputs) from that state, where the tree depth of each sequence is equal to its length. Given a computational budget of tree node expansions, performance grows with the resulting depth of the tree, which can be seen as an adaptive horizon. OP works for general dynamics and rewards, and provides a tight characterization of the relation between the computational budget and near-optimality. Motivated by these features, a number of OP algorithms have been introduced, e.g. [2, 3, 4, 5], which have proven useful in practical problems [5, 6]. OP is usually applied online in receding horizon, as a type of model-predictive control (MPC).

In this paper, we consider deterministic systems with discrete (or discretized) actions, and introduce two new OP techniques tailored for sequences that are constrained to switch only a limited number of times between different discrete actions. Inheriting the generality of OP, these techniques are able to deal with nonlinear dynamics and nonquadratic reward functions. The switch constraint is motivated by two classes of problems. In the first class (i), the loss of performance induced by the constraint is negligible – such as in time-optimal control, where solutions are of the bang-bang type. In the second class (ii), the switch constraint must be imposed due to the problem's nature, accepting the resulting performance degradation –

for example, to decrease computation time or because setting the actuator to a new discrete level is costly. Examples of the latter type include traffic signal control [7], water level control by barriers and sluices [8], networked control systems [9], etc.

First, we propose *optimistic switch-limited planning* (OSP): an algorithm that only explores sequences with at most S switches, with S fixed. This allows a significant reduction in computational complexity with respect to the state-of-the-art OP algorithm in the discrete-action, deterministic case: OP for deterministic systems (OPD) [10]. Indeed, we show that the computational effort needed by OSP to reach a given depth in the tree is polynomial in this depth, rather than exponential as in OPD. Therefore, given a computational budget n, the tree depth grows quickly and OSP converges faster to the switch-constrained optimal solution than OPD would converge to the unconstrained one. The convergence rate is dictated by the degree of the polynomial, a complexity measure for the optimal control problem.

A limitation of OSP is the need to manually tune the number of switches S. A too small value can lead to suboptimal solutions, while allowing too many switches may lead to unneeded computation. We therefore develop optimistic *adaptive* switch-limited planning (OASP), which automatically finds a good S. The value of S is increased adaptively, exploring sequences with more action switches when indicated by an increment rule. We illustrate two such rules, and analyze both variants in the same special cases as OSP was analyzed.

OSP is applied in receding horizon simulations to the problem of swinging up a rotational pendulum. Note that this problem is in class (i) where near-optimal sequences switch rarely. To illustrate class (ii), in particular systems where switches are costly, we show how OSP can take into account bandwidth limitations in networked control systems. Here, the constraint is enforced in closed loop, so that along any range of N consecutive steps there are at most S switches, where N is a parameter. Furthermore, OSP is applied to control the physical pendulum in real time. To evaluate the second algorithm, OASP, it is compared to OPD and OSP in simulations of the rotational pendulum, showing that in certain cases OASP performs better than the other methods, while remaining competitive in other cases. Finally, OASP is applied to the more complex control problem of trajectory control for quadrotors, showing the benefits of the novel algorithm over OPD and over a classical linear-quadratic regulator.

Like the entire class of OP algorithms, OSP and OASP are related to Monte-Carlo tree search [11], heuristic search [12], and planning for robotics [13]. The complexity measure of OSP (polynomial degree) is related to similar measures in other optimistic algorithms, e.g. the branching factor of near-optimal sequences in OPD [10], the near-optimality exponent in the stochastic case [4], or the near-optimality dimension in optimization [1]; due to the different structure of the explored tree, these measures do not work in the switch-constrained problem of OSP, and the new polynomial degree is needed.

In the MPC field, similar constraints on the number of action changes have been exploited to decrease computation, e.g. in the linear case in [7, 14, 15], later extended to the nonlinear case as time-instant optimization MPC [8]. Applications include hybrid [7, 16, 15] and hierarchical control [17]. In [18], the solutions are constrained to hold the command constant for a preset number of steps. In these works, an off-the-shelf optimizer (e.g. of the mixed-integer linear programming type [19]) is usually applied, and the computational effort is investigated empirically. Compared to this, the main advantage of our approach is an analytical characterization of the relationship between the computational effort and near-optimality, for the complete algorithm down to the implementation of the optimizer. A second axis of related work in MPC concerns complexity analysis, typically for linear quadratic problems, see e.g. [20]. A particularly strong work thread is in explicit MPC [21], where the optimal state feedback law is piecewise affine and the complexity of the online search for the current affine region is characterized, see e.g. [22, 23, 24]. Overall, MPC typically uses a fixed, finite horizon, and its main strengths include stability guarantees, mechanisms to handle constraints, and output feedback techniques. In contrast, OSP and OASP focus on the generality of the nonlinear dynamics they can address, while providing near-optimality and convergence rate guarantees with respect to the infinite-horizon optimum.

This paper is a revised and extended version of our conference article [25], where OSP was introduced. The present paper provides more details and insight into the analysis of OSP, while its empirical evaluation is done using a different control problem, with entirely new real-time results. The main novelty compared to [25] is however the adaptive algorithm OASP, with its analysis, numerical evaluation, and application to simulated quadrotor trajectory control.

Next, Sec. 2 gives the necessary background, Sec. 3 introduces and analyzes OSP, and Sec. 4 similarly presents and studies OASP. Experimental results for the two methods are provided in Sec. 5 and 6, respectively. Finally, Sec. 7 concludes the paper.

2. Background: Markov Decision Processes and Optimistic Planning for Deterministic Systems

Consider a Markov decision process (MDP) describing an optimal control problem with state $x \in X$, action $u \in U$, transition function $f : X \times U \to X, f(x,u) = x'$ and an associated reward function $\rho : X \times U \to \mathbb{R}$. The function $f(x,u)$ describes the transition from state x to x' when applying action u, i.e. the system dynamics. Each transition is rewarded by $\rho(x,u)$.

We assume that the action space U is finite and discrete, $U = \left\{u^1, ..., u^M\right\}$, and the system dynamics $f(x,u)$ and the reward function $\rho(x,u)$ are known. Additionally, to facilitate the analysis, the reward function is assumed to be bounded to the unit interval, $\rho(x,u) \in [0,1], \forall x, u$. The only restrictive part here is the boundedness of the reward, which is often assumed in AI approaches to solving MDPs; then, the rewards can be scaled and translated to the unit interval without affecting the optimal solution.

The objective is to find for any given state x_0 an infinite action sequence $h_\infty = (u_0, u_1, ...)$ that maximizes the value function (discounted sum of rewards):

$$v(h_\infty) = \sum_{k=0}^{\infty} \gamma^k \rho(x_k, u_k) \tag{1}$$

where $k \geq 0$ is the discrete time step, $x_{k+1} = f(x_k, u_k)$, and $\gamma \in (0,1)$ is the discount factor. The optimal value is denoted by $v^* = \sup\limits_{h_\infty} v(h_\infty)$.

Optimistic Planning for Deterministic Systems (OPD) [10, 1] is an extension of the classical A^* tree search to infinite-horizon problems. OPD looks for v^* by creating a search tree starting from x_0 that explores the space of action sequences by simulating their effects, until a given computational budget is exhausted. This budget is denoted by n and measures the number of nodes the algorithm is allowed to expand in the search tree, where expanding a node means adding M child nodes to it, one corresponding to each action from U. Fig. 1 shows an example of a tree after $n = 3$ expansions have been performed.

A node at depth d is equivalent to the action sequence $h_d = (u_0, u_1, ..., u_{d-1})$ leading to it: e.g. in Fig. 1, for the bold node at depth $d = 3$ one has $h_3 = (u^1, u^2, u^2)$. Consider any infinitely long action sequence h_∞ that starts with h_d. Now, define the following lower bound on $v(h_\infty)$:

$$\nu(h_d) = \sum_{k=0}^{d-1} \gamma^k \rho(x_k, u_k) \leqslant v(h_\infty) \tag{2}$$

and the following upper bound on $v(h_\infty)$:

$$b(h_d) = \nu(h_d) + 1 \cdot \gamma^d + 1 \cdot \gamma^{d+1} + ... = \nu(h_d) + \frac{\gamma^d}{1-\gamma} \geqslant v(h_\infty) \tag{3}$$

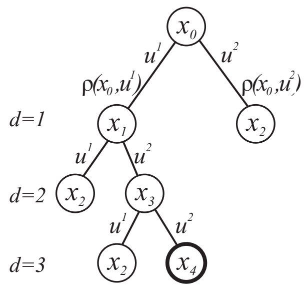


Figure 1: OPD search tree with states in the nodes and actions and rewards on the arcs (corresponding to transitions). The size of the action space is $M = 2$. Note that some states may appear several times in the tree as results of different action sequences. The algorithm does not exploit these duplicates, but keeps them separate on the tree.

Note that these bounds are valid because $\gamma < 1$ and the rewards take values between 0 and 1.

The leaf with the highest b-value is said to be *optimistic*. OPD runs the tree search iteratively, where at each iteration a leaf that is optimistic on the current tree is expanded. When there are multiple nodes with the same b-value, a *tie breaking rule* is used to select the node to expand. This rule may e.g. pick the earliest created node. At the end of the tree construction, the algorithm returns an action sequence h_d^* with the highest ν-value. Usually, e.g. in order to deal with unmodeled effects, only the first action of this sequence is applied to the system, after which the loop is closed and the algorithm is reapplied for the new state, leading to a receding-horizon scheme.

3. Optimistic Switch-Limited Planning

Optimistic switch-limited planning (OSP) is based on the same principle as OPD: it iteratively and optimistically constructs a search tree from x_0, by simulating action sequences starting from that state. After the algorithm finishes, like OPD, OSP chooses the action sequence h_d that maximizes $\nu(h_d)$. The novelty of OSP is a constraint applied to the algorithm: *never expand a node that has more than S action switches in its action sequence.*[1] Some examples of finite action sequences respecting this constraint are: for $S = 1$, $h_3 = (u^2, u^1, u^1)$, for $S = 2$, $h_3 = (u^2, u^1, u^1)$ with one switch, or $h_5 = (u^2, u^3, u^3, u^1, u^1)$ with two switches. An example of an OSP search tree is presented in Fig. 2.

We now fully analyze OSP when applied locally at a given state x_0, and later we will explain how to empirically apply it in closed loop. Using the constraint on the number of switches, the search space is restricted and the optimal solution may fall outside it. Denote the optimal return with sequences having at most S switches by v_S^*, where $v_S^* \leq v^*$. In problem class (i) (see Sec. 1), the loss $v^* - v_S^*$ due to enforcing the switch limitation is small by assumption, whereas in class (ii) it may be significant but it must be accepted due to the problem constraints or the need to reduce computations. As $S \to \infty$, v_S^* should approach v^*.

Intuitively, the constraint allows OSP to construct deeper search trees than OPD for a given problem, as in place of the nodes eliminated by the constraint OSP will explore other nodes that may be at larger depths. Therefore, for the same budget n, OSP generally ensures a smaller distance to the constrained optimum v_S^* than OPD would ensure with respect to v^*.

[1]This means that some nodes created will have $S + 1$ switches. To avoid this, we could change the expansion rule so that for nodes that have S switches, it only creates the single child that keeps the action constant. However, this would mean that expanding a node costs a varying amount of simulations, so we choose not to do it. With some care it can be seen that neither the analysis, nor the closed-loop usage explained at the end of this section are affected by this choice, so in fact the algorithm can be implemented either way.

To make the following statements more concise, we say that OSP is ε-optimal if the solution h_d^* it returns satisfies $v_S^* - \nu(h_d^*) \leq \varepsilon$. The following lemma gives an a posteriori bound, which is available after the algorithm has terminated.

Lemma 1. OSP only expands nodes that satisfy the relation $v_S^* - \nu(h_d) \leq \frac{\gamma^d}{1-\gamma}$, and it is $\frac{\gamma^{d_{\max}}}{1-\gamma}$-optimal, where $d_{\max}$ is the depth of the deepest expanded node.

Proof. At any iteration of the OSP algorithm, there exists at least one leaf sequence h_d in the search tree that forms the initial part of a constrained optimal solution. Therefore, using the definition of the b-value (3), there exists h_d such that $b(h_d) \geq v_S^*$. Since the algorithm always expands the node with the highest b-value, all the expanded nodes will satisfy the relation $b(h_d) \geq v_S^*$. As $b(h_d) = \nu(h_d) + \frac{\gamma^d}{1-\gamma}$, this relation can be rewritten $v_S^* - \nu(h_d) \leq \frac{\gamma^d}{1-\gamma}$, which proves the first part of the lemma.

Moreover, this relationship holds in particular at $d_{\max}$, for any expanded node $h_{d_{\max}}$. Then, as $\nu(h_d^*) \geq \nu(h_{d_{\max}})$ by definition, one gets $v_S^* - \nu(h_d^*) \leq \frac{\gamma^{d_{\max}}}{1-\gamma}$ from which the second part of the lemma follows. ■

Such a property is standard for optimistic algorithms. In particular, the corresponding result for OPD, in which v_S^* would be substituted by v^*, was given in a different form in [10].

Now, note that due to Lemma 1 and the switch constraint, at depths up to d OSP only expands nodes in the set:

$$H_d = \left\{ h_{d'} | \ d' \leq d; s(h_{d'}) \leq S; v_S^* - \nu(h_{d'}) \leq \frac{\gamma^{d'}}{1-\gamma} \right\}$$

where $s(h_{d'})$ counts the number of switches in the action sequence $h_{d'}$. Using the cardinality of this set one can characterize the depth the algorithm will reach.

As a first step, to provide a better insight, we take a small detour to explain how OPD behaves compared to OSP. In OPD, the nodes possibly expanded at depth d do not have to satisfy the switch limitation, instead they must only be $\frac{\gamma^d}{1-\gamma}$-optimal. It was shown then in [10] that the tree of such nodes grows with a branching factor $K \in [1, M]$, a measure of complexity of the OPD control problem. Then, the number of expandable nodes grows exponentially with the depth, so that the cardinality of the entire tree up to depth d is dominated by the number of nodes at this last depth: it is $O(K^d)$. In the case of OSP this is no longer valid, as the search tree grows polynomially with the depth. To develop an intuition on why this holds, consider the special, worst case of uniform expansion. In this case, all the nodes from a given depth are expanded before expanding deeper nodes. OSP then distributes up to S switches along sequences of length d, i.e. searches among all combinations of up to S elements from d – and the number of such combinations grows only polynomially with d. A formal analysis of this case is provided in Section 3.1 below.

Now, to describe *in general* the cardinality of H_d, a new complexity measure is needed, defined as follows.

Definition 1. Let $c > 0$ and $\sigma \in [1, S+1]$ be so that $|H_d| \leq c \cdot d^\sigma$. We define the *near-optimality degree* as the smallest value of σ so that the relationship holds.

The measure σ is called the near-optimality degree because it plays a similar role to the near-optimality dimension from the optimization algorithms at the basis of OP [1]. Note that σ always exists in the stated interval, because our special edge cases analyzed below provide its smallest and largest possible values (1 and $S+1$, respectively).

Theorem 1. Given a computational budget n, the OSP algorithm is $\frac{\gamma^{(n/c)^{1/\sigma}}}{1-\gamma}$-optimal, where c is the constant from the definition of σ.

Proof. According to Lemma 1, the OSP algorithm is $\frac{\gamma^{d_{\max}}}{1-\gamma}$-optimal if a node at depth $d_{\max}$ was expanded. Now, we calculate a lower bound on $d_{\max}$ a priori from the cardinality of set H_d. Define d^* to be the smallest depth so that $n \leq |H_{d^*}| = c \cdot {d^*}^\sigma$; this means the algorithm has surely expanded nodes at d^* (but possibly not yet at $d^* + 1$), so $d_{\max} \geq d^*$. Moreover, $d^* \geq (n/c)^{1/\sigma}$, hence the same holds for $d_{\max}$, and OSP is

$\frac{\gamma^{(n/c)^{1/\sigma}}}{1-\gamma}$-optimal. ■

A smaller σ corresponds to a slower growth of H_d, so that a given budget n allows reaching larger depths and thus a better solution. In particular, in the best case $\sigma = 1$, which means that H_d grows linearly with d, and suboptimality shrinks exponentially with increasing n. Otherwise, in general, the relationship is stretched-exponential (i.e. exponential in a power of n). Note that σ cannot usually be computed, so the near-optimality cannot be determined in advance using Theorem 1. Nevertheless, the result provides confidence that OSP converges quickly to the constrained optimum, faster in simpler problems.

In the remainder of this section, Theorem 1 and σ are illustrated for several interesting cases.

3.1. Uniform Search Trees ($\sigma = S + 1$)

Consider a problem where all the rewards are identical, say equal to 1 or to 0. Note that in this case, $v^* = v_S^*$ for any value of S.

Proposition 1. In the case of identical rewards, $\sigma = S + 1$ and the OSP algorithm is $\frac{\gamma^{\frac{1}{M}(n/c')^{1/(S+1)}}}{1-\gamma}$-optimal, where c' is a positive constant.

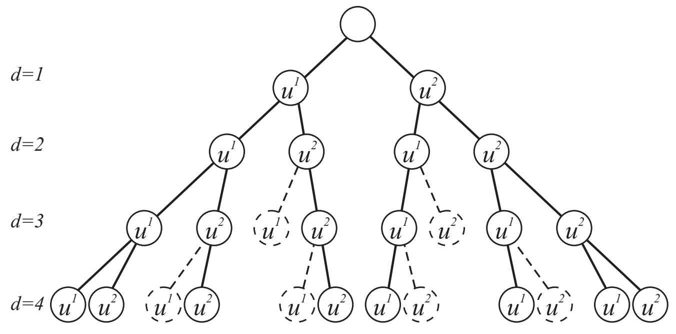


Figure 2: Uniform expansion of the OSP search tree, having $M = 2$ different actions and $S = 1$ switch allowed. The action leading to a node is put in the node. Nodes preceded by continuous lines are expanded with all M actions. Nodes on the dashed paths are no longer expanded, since they have more than S switches in their action sequence.

Proof. For identical rewards, OSP will explore the tree uniformly, in the order of depth. We first count the number of expandable nodes at each depth separately. Consider a vector N_d with $S + 1$ elements, in which every element $N_d(s)$ gives the number of nodes at depth d with s action switches, for $s = 0, ..., S$. In the example from Fig. 2, we have $N_1 = [2, 0]$, $N_2 = [2, 2]$, $N_3 = [2, 4]$, $N_4 = [2, 6]$.

As each node is expanded with exactly one child having the same action as the parent node (i.e. the same number of action switches) and $M - 1$ children with one more switch, the vector N_d can be calculated recursively as:

$$N_1(0) = M, N_1(s) = 0, s = 1, ..., S$$
$$N_d(s) = N_{d-1}(s) + (M - 1) \cdot N_{d-1}(s - 1), s = 0, ..., S,$$
for $d \geq 2$, where $N_{d-1}(s - 1) = 0$ for $s - 1 < 0$.

Using these relations, after some calculations, one obtains the number of nodes from a depth d:

$$N_d = \sum_{s=0}^{S} N_d(s) = \sum_{s=0}^{S} M \cdot (M - 1)^s \cdot \binom{d-1}{s}, d \geq 2$$

with $N_1 = M$, from where the exact number of expandable nodes in the uniform search tree of OSP, up to depth d, is:

$$|H_d| = \sum_{i=1}^{d} N_d = M + \sum_{i=2}^{d} M \sum_{s=0}^{S} (M-1)^s \cdot \binom{i-1}{s}$$

For large i, a good upper bound on the combination $\binom{i-1}{s}$ is i^s. Upper-bounding the inner sum as $(M \cdot i)^S$ and then the outer sum as $d \cdot M \cdot (M \cdot d)^S$, asymptotically, the number of nodes in the uniform search tree of OSP is $|H_d| \leq c'[M \cdot d]^{S+1}$ with c' some positive constant, and $\sigma = S+1$. Therefore, taking d^* the smallest so that $n \leq c'[M \cdot d^*]^{S+1}$, $d^* \geq \frac{1}{M}\left(\frac{n}{c'}\right)^{1/(S+1)}$ and OSP is $\frac{\gamma^{\frac{1}{M}(n/c')^{1/(S+1)}}}{1-\gamma}$-optimal. ■

Note that the resulting expression differs from the one from Theorem 1 by including M, the number of actions, which results in a more precise expression and a different constant c'; if we were to ignore this refinement, we would obtain the general result of Theorem 1.

We have obtained an interesting worst case, where σ is the largest possible. The bound achieved by OSP here is also the smallest achievable in a worst-case sense, which means that for any planning algorithm, and any value of S and n, one can construct a problem (constrained to sequences with at most S switches) for which the distance from the optimal value is[2] $\Omega(\gamma^{\frac{1}{M}(n/c')^{1/(S+1)}})$. To see this, choose the largest D so that $n \geq c'[M(D-1)]^{(S+1)}$, assign rewards of 1 for some arbitrary sequence h_d^* satisfying the constraint, but only starting from level $D+1$ onward, and rewards of 0 everywhere else. Then, OSP has uniformly expanded all nodes up to $D-1$ but none at $D+1$, so it has no information and must make an arbitrary action choice, which may not be optimal, leading to a sub-optimality of $\frac{\gamma^{D+1}}{1-\gamma} = \Omega(\gamma^{\frac{1}{M}(n/c')^{1/(S+1)}})$. An algorithm that does not expand uniformly may miss the optimal sequence for an even larger number of expansions n, so its near-optimality is at least as large. This fact also shows that OSP *behaves correctly* in the uniform case: as long as only uniform rewards are observed, the tree must be expanded uniformly, and this behavior is reflected in the bound.

3.2. Single Optimal Path ($\sigma = 1$)

In this case, a single sequence has maximal rewards (equal to 1), and all other transitions have a reward of 0, see Fig. 3. The optimal path switches S' times where S' is unknown but finite.

Proposition 2. When there is a single optimal path, $\sigma = 1$ and OSP is $\frac{\gamma^{n-c''}}{1-\gamma}$-optimal with respect to v_S^*, with c'' a positive constant.

Proof. Consider two cases: either the path with rewards of 1, belongs to the constrained search space H_d or not. In the first case, i.e. when $S \geqslant S'$, the analysis is trivial: the algorithm expands only on the optimal path. In this case, taking a computational budget of n expansions, the tree will reach the depth $d = n$.

In the second case, when $S < S'$, the unconstrained optimal path, with rewards of 1, belongs to the constrained search space H_d only up to depth d_0, as illustrated in Fig. 3. After that it becomes different from the constrained optimal path, which simply keeps the action constant although getting rewards of 0 below d_0. Denote h_{d_0} the deepest node from the optimal path belonging to the OSP tree that had a reward of 1. Additionally, denote h_{d_1} a node from the optimal path having $d_1 < d_0$. The children of h_{d_1} not on the optimal path and at a relative depth k' to node h_{d_1} are denoted $h_{d_1+k'}$. Similarly, the children of h_{d_0} will be denoted as h_{d_0+k}.

[2] Here and in the sequel, notations $g = O(h)$ and $g = \Omega(h)$ mean that g asymptotically grows, respectively, at most as fast / at least as fast as h.

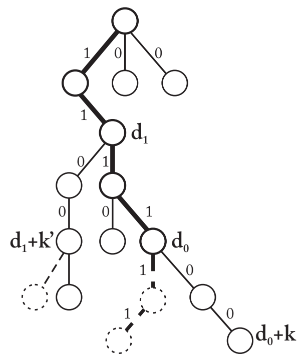


Figure 3: OSP search tree with $M = 3$ different actions, $S = 3$ switches allowed and a single optimal path, marked with bold. Rewards are marked on the transitions. All the rewards outside the optimal path are zero. Dashed circles indicate nodes not belonging to the OSP tree. Note that some leaf nodes are not represented to reduce the complexity of the figure.

Now, since the algorithm always expands the node with the highest b-value, after expanding the 1-reward path up to depth d_0, the choice between the children of h_{d_0} and h_{d_1} for further expansion is made using their b-values (3). We will show that children $h_{d_1+k'}$ start being dominated by (i.e. having smaller b-values than) h_{d_0+k} after finite k'. For any k, $\nu(h_{d_0}) < b(h_{d_0+k})$, so it suffices to show that:

$$\nu_(h_{d_0}) \geq b(h_{d_1+k'})$$
$$\frac{1-\gamma^{d_0}}{1-\gamma} \geq \frac{1-\gamma^{d_1}}{1-\gamma} + \frac{\gamma^{d_1+k'}}{1-\gamma}$$
$$\frac{\gamma^{d_1}-\gamma^{d_0}}{1-\gamma} \geq \frac{\gamma^{d_1+k'}}{1-\gamma}$$

But there exists a finite $\bar{k}$ so that this relation holds for any $k' \geq \bar{k}$. So at most a finite subtree, of relative depth at most $\bar{k}$, may be expanded for any node h_{d_1} before the nodes h_{d_0+k} start dominating. Therefore, asymptotically, the tree construction will end up expanding only these children. As nodes h_{d_0+k} have only one unconstrained child (that has no more than S switches), reaching a depth $d = d_0 + k$ will require d expansions on the constrained optimal path. Adding to this the finite number of expansions due to the children $h_{d_1+k'}$ for all d_1 as a constant c'', the total number of expanded nodes in the search tree is $|H_d| \leq c'' + d$. Hence, in this special case, $\sigma = 1$, $n \leq c'' + d$ and the algorithm is $\frac{\gamma^{n-c''}}{1-\gamma}$-optimal. ■

Thus, in this case where a "maximal" amount of structure exists in the reward function, the problem becomes easy, represented by a value of $\sigma = 1$, and the near-optimality is exponential in n rather than stretched-exponential like before. Again, we have obtained a refined expression with a different constant than in Theorem 1.

Next, we briefly discuss practical aspects regarding the closed loop implementation of the algorithm. In closed loop, online use of the method, one may apply it at each encountered state x_k, send the first action of the sequence returned to the actuator, and then repeat the process in receding horizon. The resulting closed-loop sequence may have more than S switches, but this procedure is appropriate for problems where the number of switches is not a real constraint in the problem. Recall that in such problems, the space of solutions searched by the algorithm is restricted either due to prior knowledge on their structure, in case (i) of Sec. 1, or for computational reasons in case (ii).

When switches are costly in the real world (still in case (ii) of Sec. 1), the constraint must be enforced in closed loop. It is not practical to enforce only S switches for the entire infinite horizon, as after they are exhausted the algorithm can no longer react to unmodeled effects (e.g. disturbances). Instead, we suggest ensuring that at most S switches are applied for any range of consecutive N steps, where N is a tuning parameter. This can be implemented easily, by keeping track of the past N actions (over the range $k-N, \dots, k-1$) and ensuring that nodes violating the condition are not expanded by the algorithm. Further, for the ranges of steps where no more switches are allowed, the action is simply held constant and OSP is only reapplied when a new switch becomes eligible.

Note that with the first procedure, the state at steps $k+2, k+3, \dots$ can leave the tree seen at step k, because the closed-loop sequence is unconstrained (x_{k+1} is on the tree at k by definition). Our analysis does not cover this closed-loop effect, instead only characterizing the open-loop sequence found at each step. Nevertheless, since the loop is closed at $k+1$, $k+2, \dots$, the algorithm can react in receding horizon by computing new sequences for these states, which is expected to keep performance acceptable. With the second procedure, the same effect occurs, but more slowly due to the limited number of switches allowed. Moreover, in either case, due to modeling errors the states will deviate from the predicted ones, but as usually in MPC, the receding-horizon feedback mechanism is also expected to compensate for this.

We will test both procedures in the upcoming experiments of Sec. 5.

4. Optimistic Adaptive Switch-Limited Planning

The OSP algorithm discussed so far searches in a constrained space of action sequences with at most a fixed number of S switches. Recall from above that in many problems, S is not a hard constraint, being instead a tunable parameter that increases the computational efficiency of the algorithm, which can therefore be set to arbitrary values.

To eliminate the need to tune S, while still keeping the benefits of OSP as much as possible, next, an optimistic *adaptive* switch-limited planning (OASP) algorithm is introduced. OASP uses the same principle as OSP: explore a search tree optimistically, but never expand nodes that have more than S action switches in their action sequence. However, OASP allows for incrementing S based on a certain criterion that will be called the *increment rule*. As a simple example, a simple increment rule may be to start with $S_0 = 0$ and increase S whenever a depth $(S+1) \cdot d_0$ is reached. In this example, presented also in Fig. 4, OASP explores the search tree with $S = 0$ until d_0, starting from where it explores the tree with $S = 1$ until depth $2 \cdot d_0$, and so on. After incrementing S, the algorithm may return to lower depths, as also seen in Fig. 4, since the b-values of nodes at smaller depths that were earlier not eligible for expansion might be larger, and if they are now eligible due to the increased S they will be expanded.

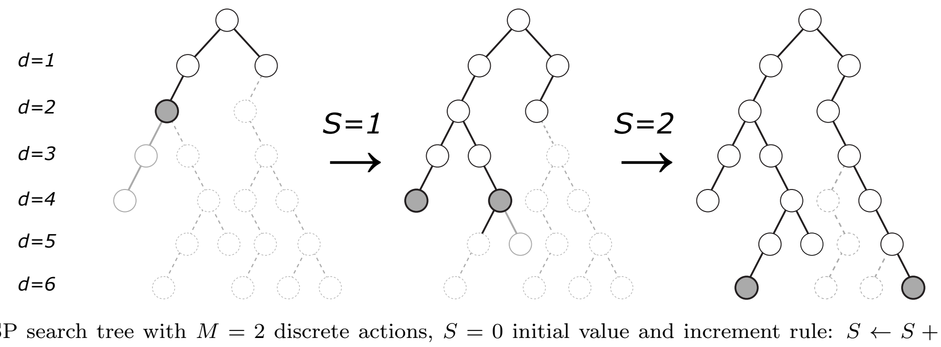


Figure 4: OASP search tree with $M = 2$ discrete actions, $S = 0$ initial value and increment rule: $S \leftarrow S + 1$ if the deepest expanded node is at $d = (S + 1) \cdot 2$. Nodes not yet expanded are marked in gray outline, nodes not eligible for expansion are marked with dashed line, whereas nodes filled with gray are those where the increment rule can be activated. All the nodes have 2 children, although the figure does not show all of them to limit the focus to the expanded nodes.

The choice of the increment rule is crucial for the performance of the algorithm. Two increment rules will be used, different from the simple one used as an example. To develop an intuition, recall that OSP ensures a near-optimality of $\gamma^{d_{\max}}/(1-\gamma)$ where $d_{\max}$ is the deepest expanded depth with the current value of S. However, if the goal is to reach the unconstrained optimum v^*, then it is only useful to increase $d_{\max}$ until the suboptimality due to the limited depth of exploration is smaller than the suboptimality due to the limited value of S, $v^* - v^*_S$. So, in principle, S should be increased as soon as $v^* - v^*_S > \gamma^{d_{\max}}/(1-\gamma)$. Now of course, the algorithm does not know v^* or even v^*_S, so as proxy it will replace the left-hand side of the inequality by a (very rough) estimate of the gradient of v^*_S with respect to S: the difference between either b-values or ν-values for the latest two values of S, scaled by a constant β.

Specifically, consider a *b-rule*, defined as:

$$S = 0.\ S \leftarrow S + 1, \text{ if } \ b^*_{S-1} - b^*_S \geqslant \frac{1}{\beta} \cdot \frac{\gamma^{d'}}{1-\gamma} \tag{4}$$

where b^*_{S-1} denotes the maximum of the b-values before the algorithm incremented S to its current value, b^*_S is the maximum value in the current search tree with S action switches allowed, β is a tuning parameter, and d' is the depth of the deepest node expanded so far. Also, $b^*_{-1} = \frac{1}{1-\gamma}$ by definition. The OASP algorithm that uses the b-rule is called OASP-b.

A second rule, called *ν-rule*, is defined as:

$$S = 0.\ S \leftarrow S + 1, \text{ if } \ \nu^*_S - \nu^*_{S-1} \geqslant \frac{1}{\beta} \cdot \frac{\gamma^{d'}}{1-\gamma}, \ \text{ or if } \ S < \frac{d'}{d_{\text{lim}}} \tag{5}$$

where, similarly to the b-rule, ν^*_{S-1} is the maximum of the ν-values before S is incremented to its current value, ν^*_S is the currently highest ν-value, and β and d' have the same meaning as before. Note that the ν-rule contains an extra increment condition that prevents the algorithm from keeping S constant indefinitely, which is important in certain special cases, as it will become apparent in the analysis from Sec. 4.1. We set $\nu^*_{-1} = 0$, and refer to the algorithm that uses the ν-rule as OASP-ν.

Next, the performance of OASP is analyzed in the same special cases as OSP from Section 3: uniform tree and single optimal path. Sections 4.1 and 4.2 discuss the performance of OASP in the case of two different uniform search trees, while Sec. 4.3 analyzes OASP in the case of a search tree with a single optimal path.

4.1. Uniform 0-reward Tree

Consider a uniform search tree with rewards of 0 for all the nodes. In this case $v^* = v^*_S$ for any value of S, and thus the near-optimality of the methods can directly be compared.[3]

Proposition 3. In the case of identical 0 rewards, OASP-b is $O\left(\gamma^{d_0 \log n}\right)$-optimal, while OASP-$\nu$ is $O\left(\gamma^{d_{\text{lim}} \log n}\right)$-optimal, where d_0 is a finite depth defined in the proof.

Proof. For uniform rewards of 0, $\nu(h_d) = 0$ and $b(h_d) = \frac{\gamma^d}{1-\gamma}$ for any node h_d at depth d, i.e. the ν-values are zero everywhere, and the b-values decrease as the depth increases. Therefore, the highest b-values are at the lowest unexpanded depths, and the algorithms explore the tree in the order of the depths.

In the case of OASP-b, when $S = 0$, for any $\beta > 0$ there is a depth $d_0 = d' + 1$ that validates the condition $b_{-1} - b_0 = \frac{1}{1-\gamma} - \left(0 + \frac{\gamma^{d_0}}{1-\gamma}\right) \geqslant \frac{1}{\beta} \cdot \frac{\gamma^{d'}}{1-\gamma}$, i.e. $1 \geqslant \left(\frac{1}{\beta \cdot \gamma} + 1\right) \cdot \gamma^{d_0}$. Recall that d' denotes the depth of the deepest

[3]It is still, however, useful to judge the near-optimality of the algorithms by their largest expanded depth, via $\gamma^{d_{\max}}/(1-\gamma)$, because – like in Section 3.1 – examples where they have this suboptimality can be constructed.

node expanded so far in the search tree. After this condition is validated, the algorithm reaches depth d_1 with $S = 1$, and the condition becomes $b_0 - b_1 = \frac{\gamma^{d_0}}{1-\gamma} - \frac{\gamma^{d_1}}{1-\gamma} \geqslant \frac{1}{\beta}\frac{\gamma^{d'}}{1-\gamma}$, i.e. $1 \geqslant \left(\frac{1}{\beta \cdot \gamma} + 1\right) \cdot \gamma^{d_1 - d_0}$. Note that this is validated for $d_1 = 2d_0$, and so on. Thus, for an arbitrary S, one obtains that S is incremented at every depth that is a multiple of d_0. Thus, since the tree is explored in the order of the depths, to reach a certain depth $d = (S+1)d_0 - 1$, OASP-b explores all the sequences with $S - 1$ action switches up to that depth, and, after expanding a node from $d_S = (S+1)d_0$, it starts to explore sequences with S action switches.

Note that sequences with S switches are explored if and only if the tree has reached at least depth d_S. The algorithm may then return to lower depths with higher values of S, but will never expand sequences with $S + 1$ switches before reaching the depth d_{S+1}. Thus, one can conclude that up to a depth d_S, the tree expanded by OASP is included in the tree that OSP would expand up to d_S with a fixed S constraint.

It is also true that once the algorithm expands a sequence with S switches, it has reached for sure at least depth d_S. Thus, it is sufficient to perform the analysis based on the highest S reached, denoted S^*. Then, an upper bound on the number of expansions in the uniform 0-reward tree results from the uniform tree expansion special case from Sec. 3.1, namely at most $c \cdot (M \cdot d^{S^*})^{S^*+1}$ expansions, with c some positive constant. So, based on the analysis of OSP from Sec. 3.1, OASP-b is $O\left(\gamma^{\frac{1}{M} n^{1/(S^*+1)}}\right)$-optimal.

Now, S^* is of course unknown, so we will find a lower bound S. Take the smallest S that, for a given computational budget n, satisfies

$$n \leqslant c \cdot (M \cdot d_S)^{S+1} \tag{6}$$

Replacing $d_S = S \cdot d_0$ in (6), one can determine S by first rewriting

$$\begin{aligned} n &\leqslant c \cdot (M \cdot (S+2) \cdot d_0)^{S+1} \\ M \cdot d_0 \cdot \log \tfrac{n}{c} &\leqslant e^{\log(M \cdot d_0 \cdot (S+2))} \cdot \log(M \cdot d_0 \cdot (S+2)) \end{aligned} \tag{7}$$

Using the Lambert W function [26], which satisfies

$$z = e^{W(z)} \cdot W(z), \forall z \in \mathbf{R}^+ \tag{8}$$

and knowing that W is increasing for positive values [26], it can be applied to both sides of (7), obtaining

$$W\left(M \cdot d_0 \cdot \log \frac{n}{c}\right) \leqslant W(z) = \log(M \cdot d_0 \cdot (S+2)) \tag{9}$$

Knowing that the Lambert function for real values can be bounded as [26]

$$\log x - \log\log x + \frac{1}{2}\frac{\log\log x}{\log x} \leqslant W(x) \leqslant \log x - \log\log x + \frac{e}{e-1}\frac{\log\log x}{\log x} \tag{10}$$

a lower bound is used to replace the function from (9), namely $W(x) \geqslant \log x - \log\log x = \log \frac{x}{\log x}$. Then,

$$\begin{aligned} \log \tfrac{M \cdot d_0 \cdot \log \frac{n}{c}}{\log(M \cdot d_0 \cdot \log \frac{n}{c})} &\leqslant \log(M \cdot d_0 \cdot (S+2)) \\ S &\geqslant \tfrac{\log \frac{n}{c}}{\log(M \cdot d_0 \cdot \log \frac{n}{c})} - 2 \end{aligned} \tag{11}$$

Thus, since $S^* \geqslant S$, the near-optimality of OASP-ν can be approximated as

$$\begin{aligned} O\left(\gamma^{\frac{1}{M} n^{1/(S^*+1)}}\right) &\leqslant O\left(\gamma^{\frac{1}{M} n^{1/(S+1)}}\right) \leqslant O\left(\gamma^{\frac{1}{M} n^{1/(S+2)}}\right) \leqslant O\left(\gamma^{\frac{1}{M} n^{\log(M \cdot d_0 \cdot \log \frac{n}{c})/\log \frac{n}{c}}}\right) \leqslant \\ &\leqslant O\left(\gamma^{\frac{1}{M} n^{\log(M \cdot d_0 \cdot \log n)/\log n}}\right) = O\left(\gamma^{d_0 \cdot \log n}\right) \end{aligned} \tag{12}$$

In the case of OASP-ν, the condition $\nu_S^* - \nu_{S-1}^* = 0 \geqslant \frac{1}{\beta} \cdot \frac{\gamma^{d'}}{1-\gamma}$ is never satisfied. The value of S is only incremented by the extra criterion, at depths multiple of the limit value $d_{\lim}$ from (5). Then, the analysis from the above case of OASP-b can be reused with d_0 replaced by $d_{\lim}$, so that OASP-ν expands $n = O\left((M \cdot d_{\lim} \cdot (S^* + 2))^{S^*+1}\right)$ nodes in the uniform 0-reward tree, and is $O\left(\gamma^{d_{\lim} \cdot \log n}\right)$-optimal. ■

As a baseline, for this tree, OPD expands $n = O(M^{d_{\max}})$ nodes, with $d^{\max}$ the depth of the deepest expanded node, so it has $d^{\max} = \Omega\left(\log n / \log M\right)$, and is $O\left(\gamma^{\log n / \log M}\right)$-optimal [10]. OSP is $O\left(\gamma^{\frac{1}{M} n^{1/(S+1)}}\right)$-optimal, as discussed in Sec. 3.1, see also Table 1 for a synthetic comparison.

In this special case, the OASP methods have a similar convergence rate to OPD, somewhat faster due to the d_0 or respectively $d_{\lim}$ multiplier in the near-optimality exponent. The increment rules of OASP increase S to infinity correctly but unnecessarily, as all the sequences obtain the same reward. On the other hand, the constant switch constraint of OSP allows the algorithm to explore deeper and to provide a better near-optimality bound. However, as shown in the special case from Sec. 4.3 below, sometimes the constant S used with OSP will get the algorithm stuck in suboptimal solutions.

4.2. Uniform 1-reward Tree

Consider a uniform tree with constant rewards of 1. Again, $v^* = v_S^*$ for any value of S, and the near-optimality of the methods is directly comparable.

Proposition 4. In the case of identical rewards of 1, OASP-b is $\frac{\gamma^{n/M}}{1-\gamma}$-optimal, while OASP-$\nu$ is $O\left(\gamma^{d_0 \log n}\right)$-optimal, where d_0 is a finite depth defined in the proof.

Proof. In this case, for any algorithm, a node h_d from depth d has $\nu(h_d) = \frac{1-\gamma^d}{1-\gamma}$ and $b(h_d) = \frac{1}{1-\gamma}$. Since the b-values are the same for all the nodes, the tree expansion is performed in order of node creation.

OASP-b will never increment S, since $b_{-1}^* - b_0^* = 0$ for any node, and the condition from the b-rule is never activated. Thus, OASP-b explores this tree with $S = 0$ on M constant-action paths, expanding $n = M \cdot d_{\max}$ nodes and being $\frac{\gamma^{n/M}}{1-\gamma}$-optimal.

The increment condition of OASP-ν is validated for a given $d_0 = d' + 1$. Starting with $S = 0$, one has $\nu_0^* - 0 = \frac{1-\gamma^{d_0}}{1-\gamma} \geqslant \frac{1}{\beta} \cdot \frac{\gamma^{d'}}{1-\gamma}$, or equivalently $1 \geqslant \left(\frac{1}{\beta \cdot \gamma} + 1\right) \cdot \gamma^{d_0}$. Having this condition validated, the algorithm reaches depth d_1 with $S = 1$, and the condition becomes $\nu_1^* - \nu_0^* = \frac{1-\gamma^{d_1}}{1-\gamma} - \frac{1-\gamma^{d_0}}{1-\gamma} \geqslant \frac{1}{\beta} \cdot \frac{\gamma^{d'}}{1-\gamma}$, i.e. $1 \geqslant \left(\frac{1}{\beta \cdot \gamma} + 1\right) \cdot \gamma^{d_1 - d_0}$. Note that this is validated for $d_1 = 2d_0$, and so on, like in the case of OASP-b from Sec. 4.1. Thus, following the same analysis line as in Sec. 4.1, OASP-ν is $O\left(\gamma^{d_0 \cdot \log n}\right)$-optimal. ■

Note that OPD and OSP have the same performance as in the previous case of the uniform 0-reward tree. OPD is $O\left(\gamma^{\log n / \log M}\right)$-optimal, whereas OSP is $O\left(\gamma^{\frac{1}{M}(n/c)^{1/(S+1)}}\right)$-optimal. Thus, as OASP-$b$ keeps $S = 0$, in this special case it explores much deeper and provides a tighter near-optimality bound, even better than OSP. On the other hand, OASP-ν has a similar performance as OPD but with faster convergence to the near-optimal solution, like in the previous special case.

4.3. Single Optimal Path

Take a tree that has zero rewards for all the nodes except for a single optimal path that switches S' times where S' is unknown but finite, see an example in Fig. 3. Note that even in this case, $v^* = v_S^*$ as soon as $S \geq S'$ switches are allowed.

Proposition 5. In the case of a single optimal path, OASP with both increment rules is $O\left(\gamma^n\right)$-optimal.

Proof. The two OASP methods explore this tree similarly to each other. They start exploring the optimal path with $S = 0$. Depending on the chosen value of β and the depth at which the first action switch in the optimal path occurs, both methods expand some nodes on suboptimal paths (i.e. with zero rewards). OASP-b has to get a high enough difference between the b-values, i.e. expand a sufficient number of suboptimal nodes until incrementing S and returning to the optimal path. OASP-ν has to explore nodes, before the reached depth is high enough to activate the extra increment condition. When the increment conditions are satisfied, both methods increment S and keep exploring the optimal path up to the next action switch. There, further suboptimal paths are explored until S is incremented, an so on, until $S \geqslant S^{'}$. Up to this point, a constant-sized subtree is explored. Thereafter, the OASP methods explore only on the optimal path. Therefore, the OASP algorithms expand $n = O(d_{\max})$ nodes and are $O(\gamma^n)$-optimal. ■

As a comparison, OPD explores only the optimal path and has thus $n = d^{\max}$ expansions and $\frac{\gamma^n}{1-\gamma}$-optimality. OSP with $S \geqslant S^{'}$ has exactly the same number of expansions and near-optimality as OPD. However, taking $S < S^{'}$ makes OSP follow the optimal path until the available switches are consumed, starting from where on OSP explores zero rewarded paths and has thus *constant suboptimality.* Thus, this special case illustrates how the increment rule of OASP helps the algorithm work with similar performance to OPD even in situations when OSP would fail to find near-optimal solutions.

Special case	**OPD**	**OSP**	**OASP-b**	**OASP-ν**
Uniform 1-rewards	$O\left(\gamma^{\log n/\log M}\right)$	$O\left(\gamma^{\frac{1}{M}n^{1/(S+1)}}\right)$	$\boldsymbol{O\left(\gamma^{n/M}\right)}$	$O\left(\gamma^{d_0\cdot\log n}\right)$
Uniform 0-rewards	$O\left(\gamma^{\log n/\log M}\right)$	$\boldsymbol{O\left(\gamma^{\frac{1}{M}n^{1/(S+1)}}\right)}$	$O\left(\gamma^{d_0\cdot\log n}\right)$	$O\left(\gamma^{d_{\lim}\cdot\log n}\right)$
Single optim. path	$\boldsymbol{O\left(\gamma^{n}\right)}$	$O(1)$, suboptimal	$\boldsymbol{O(\gamma^{n})}$	$\boldsymbol{O(\gamma^{n})}$

Table 1: Near-optimality of the algorithms. For uniformity, all the rates are shown in asymptotic notation and (where meaningful) as a power of γ. For each case, the best performing algorithm or algorithms are highlighted in bold.

Table 1 summarizes the results in all these cases. A general convergence rate of OASP, for any problem, has not been obtained yet. Nevertheless, the analysis above illustrates that OASP with both increment rules performs better than OPD or OSP in certain types of problems, where it is therefore interesting; and gracefully degrades either to OSP or OPD in some other problems.

OASP can be used in closed loop the usual way, by applying the first action of each sequence returned and then recalculating actions in receding horizon.

5. Experimental Evaluation of OSP

Our first batch of experiments uses the Quanser rotational pendulum, shown in Fig. 5. This system consists of a heavy rod, the pendulum, sitting on an unactuated rotational joint at the end of an intermediate, horizontal link actuated through a motor. The state vector is $[\theta, \dot{\theta}, \alpha, \dot{\alpha}]^{\mathrm{T}}$ where the variables are the angle $\alpha \in [-\pi, \pi)$ rad of the pendulum (zero when pointing up), the angle $\theta \in [-\pi, \pi)$ rad of the horizontal link (zero when pointing forward), and their angular velocities $\dot{\alpha}, \dot{\theta}$ in $[-100, 100]$ rad/s. The two angles "wrap around" the ends of the interval. The input voltage u varies in the range of $[-6, 6]$ V. The dynamics are:

$$\begin{array}{rcl} \ddot{\alpha} & = & (ad\sin\alpha - b^2\dot{\alpha}^2\sin\alpha\cos\alpha - be\dot{\theta}\cos\alpha + bfu\cos\alpha)/(ac - b^2\cos^2\alpha) \\ \ddot{\theta} & = & (-bc\dot{\alpha}^2\sin\alpha + bd\sin\alpha\cos\alpha - ce\dot{\theta} + cfu)/(ac - b^2\cos^2\alpha) \end{array} \tag{13}$$

with $a = 0.0112$, $b = 0.0046$, $c = 0.0048$, $d = 0.2099$, $e = 0.0729$, $f = 0.1281$, computed from the physical parameters. The goal of the control problem is to reach the zero equilibrium, but the system is underactuated

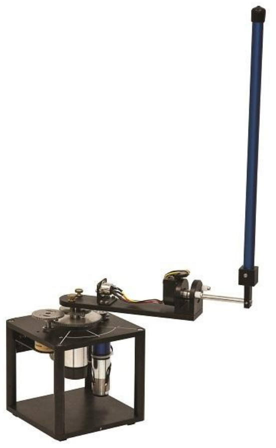


Figure 5: Quanser rotational pendulum

so the pendulum must first be swung back and forth to accumulate energy, so long trajectories must be found.

In all the experiments, the planners work with the normalized reward function, defined as $\rho(x,u) = 1 - (x^TQx + Ru^2)/r_{\max}$, with $Q = \mathrm{diag}(0.1, 0.1, 1.0, 0.001)$, $R = 0.1$, and $r_{\max} \approx 1024.46$ representing the largest possible value of the unnormalized cost $x^TQx + Ru^2$. The discount factor in (1) is set to $\gamma = 0.98$ for all the methods. All the experiments compare the methods for the same range of computational budgets.

Although the rotation pendulum is a standard problem, it is an appropriate benchmark for our algorithm: it is nonlinear, requires large horizons to plan the swings, and the swings exhibit the limited-switch property, so the basic problem is in class (i). So, first, Sec. 5.1 provides simulation results for OSP without enforcing the constraint in closed loop. Then, the results from Sec. 5.2 show how limiting the number of switches in closed loop can help when the pendulum is controlled via a network. Finally, Sec. 5.3 shows the performance of OSP when applied on the real rotational pendulum.

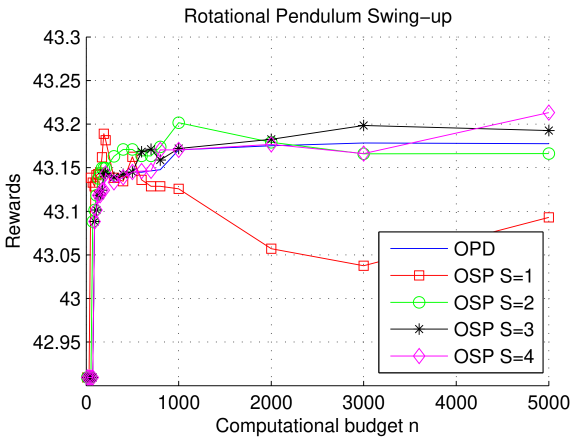
5.1. Rotational Pendulum Swing-up Simulations

In the simulations from this section, the algorithms are tested online, in receding horizon from initial state $x_0 = [-\pi, 0]^T$ and for a duration of 100 steps, with a sampling time of 0.05s, so that the trajectory length is 2.5s. The discretized actions are $-6, 0, 6\,\mathrm{V}$, so $M = 3$. For the resulting finite sequence of control actions, the truncated discounted return is calculated.

Fig. 6 shows the resulting returns for varying computational budgets. An important remark is that by increasing the value of S, OSP converges to OPD as expected from the construction of the algorithm. Choosing $S = 1$, OSP obtains a sub-optimal solution where the difference between v^* and v^*_S is clearly visible for n large. For lower values of n OSP with $S = 1$ gets higher returns, which can be explained by luck. Taking higher values for S, the advantage of OSP is clearly reflected: OSP obtains the same return as OPD for lower values of n. In other words, for intermediate values of S, OSP is able to obtain a result of the same quality as OPD or better, using fewer computational resources.

5.2. Application to Simulation of Networked Control Systems

In networked control systems, the controller is connected to the system by a communication network shared with other devices. We consider the setting where state measurements can be performed at every



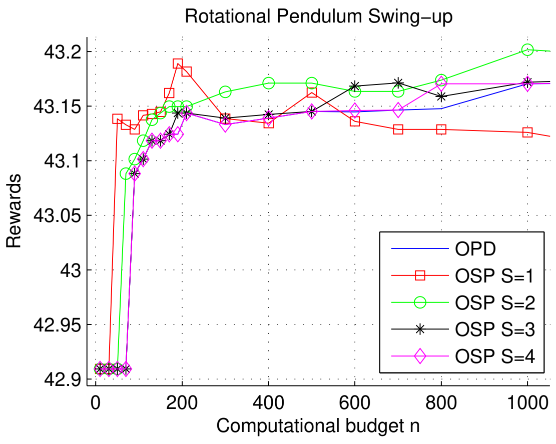


Figure 6: Rotational pendulum return for OPD and OSP: simulation results for budgets up to 5000 expansions (left), zoomed in for budgets up to $n = 1000$ (right).

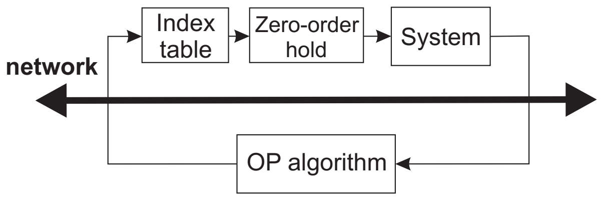


Figure 7: Networked control system architecture

step, while changes in the control action are expensive and should be minimized; this is standard in so-called event-triggered control [9]. The algorithm is used as explained at the end of Section 3: a new action is transmitted only when a switch occurs, and otherwise the old action is maintained. By setting the ratio of S switches (transmissions) per N steps, OSP can be used to fine-tune the usage of the network. Discretizing the actions in a small number of actions is also useful [27], since it reduces the size of the control packets to a few bits (the action index), requiring a local table to transform the action index back into the true value on the system side. Fig. 7 shows this architecture.

This setting is illustrated for the rotational pendulum. Using the same discretized actions as above, OSP is used to enforce at most $S = 3$ action changes over $N = 6$ or 9 steps (since the sampling time is 0.05s and $M = 3$, no more than 9 bits of data are sent in 0.3 or 0.45s, respectively). Fig. 8 shows that OSP is able to perform similarly to OPD even when the constraint is applied online, though one needs a trade-off between the desired performance and the imposed constraints expressed by S and N.

5.3. Experiments with a Real Rotational Pendulum

This section shows experimental results when OSP is used to swing up the real rotational pendulum. The experiments are run with a C++ implementation of OSP in order to increase computation speed. OSP is evaluated for inputs $U \in \{-6, 0, 6\}$ V, switch constraint $S = 3$, and budget of $n = 2100$ expansions, without enforcing the constraint in closed loop. Since the computation time of the algorithm is not negligible, we apply the real-time planning approach from [28] to plan actions one step ahead, using the sampling period $T_S = 0.05$ s to run the algorithm. Fig. 9 plots the obtained trajectories and corresponding inputs and rewards.

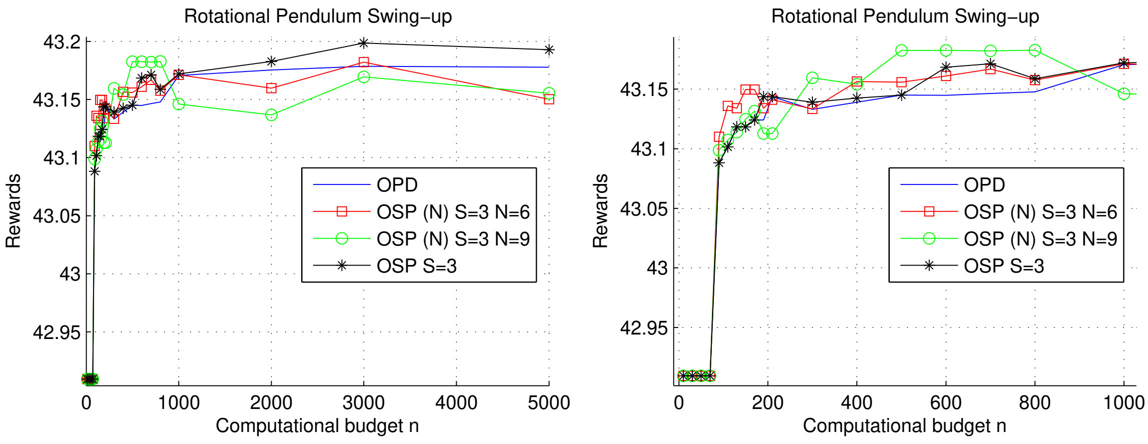


Figure 8: Networked control system: simulation results for budgets up to 5000 expansions (left), zoomed in for budgets up to $n = 1000$ (right).

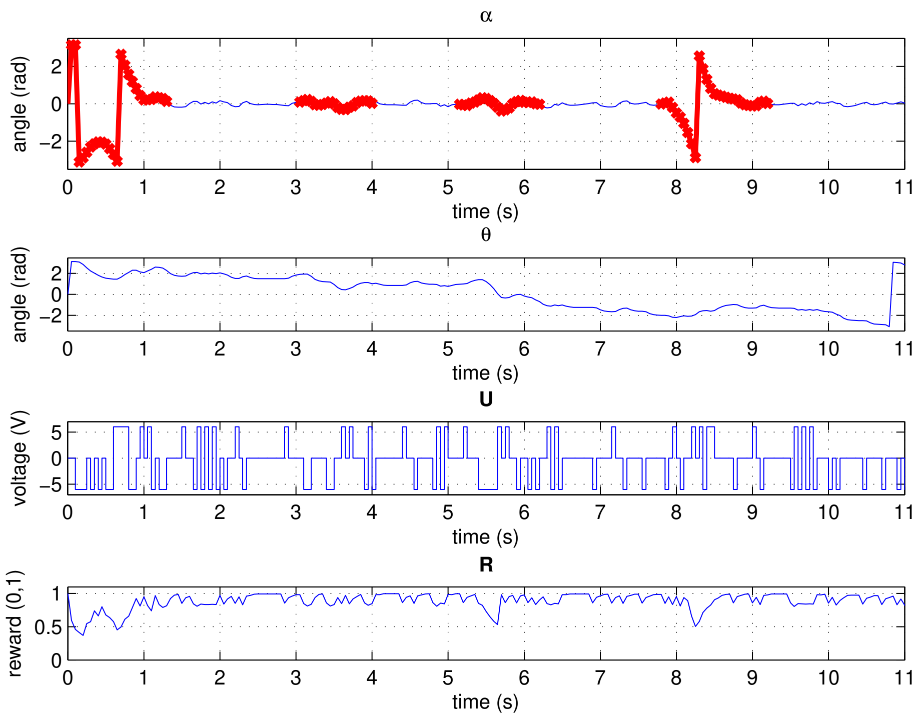


Figure 9: Rotational pendulum online trajectory: the graphs show, from top to bottom, the evolution of the angular position of the pendulum (α) and of the actuated joint (θ), the control inputs U, and the rewards.

The first graph from Fig. 9 shows the evolution of the angular position of the pendulum, where zero means that the pendulum is pointing up. A very first remark is that the algorithm is able to bring up and stabilize the pendulum. Four sections of the trajectory are highlighted. The first one marks the initial swing-up. The other three sections highlight periods when perturbations were introduced by softly hitting the pendulum. The events from around 3.5 s and 5.5 s correspond to small hits, when the pendulum destabilizes only a little, and the planner is able to immediately correct the position. The last event marks a more pronounced perturbation, leading to higher angular velocity of the pendulum for which a solution that keeps the pendulum in a pointing up state cannot be found, so it is reswung. A video of this experiment can be viewed at `http://rocon.utcluj.ro/files/rotpend_osp.mp4`.

6. Experimental Evaluation of OASP

This section presents experimental results for OASP using simulations in two control problems. First, Sec. 6.1 compares OASP to OPD and to OSP using again the rotational pendulum, while evaluating the effectiveness of the two proposed increment rules of OASP, the b-rule and the ν-rule. Based on the results OASP-ν is selected to be further evaluated. The second set of experiments, presented in Sec. 6.2, uses the more involved problem of quadrotor flight path control. OASP is compared against OPD and to a baseline, simple linear quadratic regulator (LQR), when the methods are used for determining the sequence of actions to take in order to guide a quadrotor through several waypoints while minimizing a certain cost.

6.1. Rotational Pendulum Simulations

Consider again the rotational pendulum control problem of Sec. 5. The OPD, OSP and OASP algorithms are evaluated online, i.e. calculating and applying actions in closed loop, for a number of 100 discrete time steps, with computational budgets n up to 5000 expansions. The OSP algorithm is evaluated with $S = 3$ based on the best settings from Sec. 3, while for OASP-ν and OASP-b we performed preliminary tuning of the parameter β in order to obtain the best results. OASP-ν uses $\beta = 9$, while OASP-b takes $\beta = 1500$.

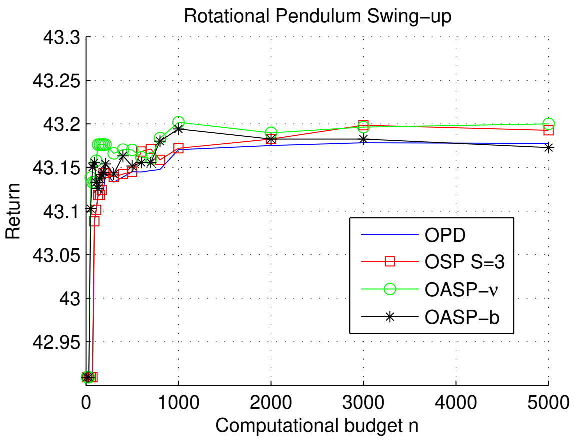
Fig. 10 shows the results. An important remark is that the OASP methods outperform OPD and OSP for low values of n, meaning that OASP in both of its variants is able to find a near-optimal solution with less computation, already for less than 100 expansions per closed-loop step. Also, as seen in the left graph of Fig. 10, OASP-ν has better performance for higher budgets too.

Since the performance of OASP depends on tuning the new parameter β from the increment rules (4) and (5), in Fig. 11 we investigate the sensitivity with respect to this parameter. Note that d_{lim} is set large enough, so that the ν-rule never activates that branch. For both OASP variants, too low values of β lead to suboptimal performance due to increasing S too slowly, while very large values increase S too fast, losing the performance benefits over OPD. The major difference between the variants is that OASP-ν is more robust to changes of the parameter β, due to which the next experiment is narrowed down to evaluating only OASP-ν against other methods. Recall that a fundamental advantage compared to OSP is that OASP will eventually approach the unconstrained optimum, even when a badly chosen β makes it approach it slowly.

6.2. Simulation Results for Quadrotor Trajectory Control

In this section, OASP is compared against OPD using the problem of quadrotor flight path control. The methods are used for determining the sequence of actions to take in order to guide a quadrotor through a sequence of waypoints while minimizing a certain cost. Also, the algorithms are compared to a baseline, simple linear quadratic regulator (LQR). The LQR controller [29, 30], similarly to OPD and OASP, optimizes a quadratic cost function but is designed based on linearized dynamics, see [31] for details.

The quadrotor is modeled using twelve states representing the position, orientation, linear and angular velocity of the quadrotor in space, with respect to each axis, x, y and z. An additional state, x_{13} is used to mark the index of the current waypoint towards which the quadrotor is heading. Note that x_{13} is not



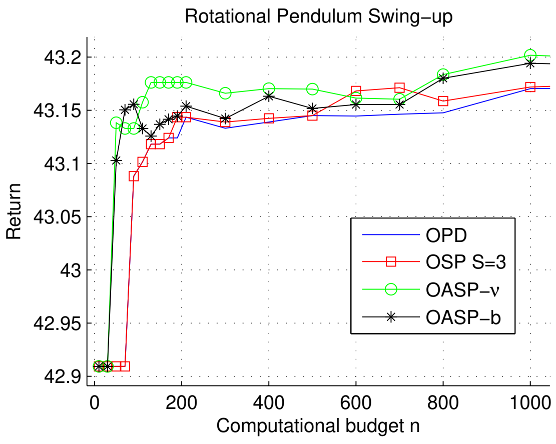
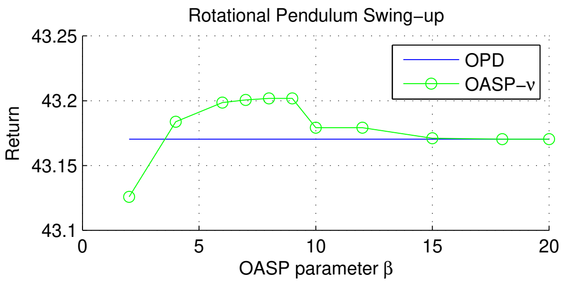


Figure 10: Rotational pendulum: cumulative rewards for computational budget up to $n = 5000$ (left), and the same results zoomed in for budgets up to $n = 1000$ expansions (right).



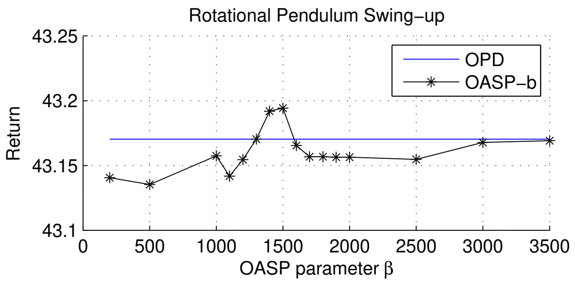


Figure 11: Rotational pendulum: given a fixed computational budget of $n = 1000$ expansions, the left graph shows the return obtained by OASP-ν for $\beta \in [2, 20]$, while the right graph presents the returns of OASP-b for $\beta \in [200, 3500]$. Both graphs present as a reference the return obtained by OPD with $n = 1000$.

part of the dynamics, but is used in the rewards to indicate the currently targeted waypoint. The inputs are $U = [U_{\text{coll}}, U_\phi, U_\theta, U_\psi]$, i.e. the collective force acting on the z axis, and the torques acting on the three axes. The transition function is obtained by using a sampling time $T_S = 0.004$ s and numerically integrating the continuous-time dynamics. The dynamics from [32] are used:

$$\begin{cases} \ddot{x} &= [c(\phi)s(\theta)c(\psi) + s(\phi)s(\psi)]\,\frac{U_{\text{coll}}}{m} \\ \ddot{y} &= [c(\phi)s(\theta)s(\psi) - s(\phi)c(\psi)]\,\frac{U_{\text{coll}}}{m} \\ \ddot{z} &= -g + c(\phi)c(\theta)\frac{U_{\text{coll}}}{m} \\ \ddot{\phi} &= \dot{\theta}\dot{\psi}\frac{I_y - I_z}{I_x} + \frac{1}{I_x}U_\phi \\ \ddot{\theta} &= \dot{\phi}\dot{\psi}\frac{I_z - I_x}{I_y} + \frac{1}{I_y}U_\theta \\ \ddot{\psi} &= \dot{\phi}\dot{\theta}\frac{I_x - I_y}{I_z} + \frac{1}{I_z}U_\psi \end{cases} \tag{14}$$

with $m = 0.4472$ kg the total mass of the quadrotor, $g = 9.80665$ m/s^2 the gravitational acceleration, and $(I_x, I_y, I_z) = (0.0020, 0.0016, 0.0035)$ Nms^2 the moments of inertia on the corresponding axes.

OPD and OASP are used with four macro actions that, in turn, determine a sequence of low-level control inputs U. The four macros are: fly ahead, turn left, turn right, and hover. Each macro action lasts for

$50 \cdot T_S = 0.2$ s, during which the quadrotor is expected to fulfill the desired behavior. All these macros can be applied in any state of the system, and they are inspired from the existing high-level control of the AR.Drone quadrotor we simulate [33], which exposes only high-level linear and angular velocity commands. The macro actions are achieved using simple LQR controllers, for more details refer to [31].

The control goal of passing through all the waypoints on the shortest path is defined using the reward function:

$$\rho(x, u) = \left(\frac{x_{13} - 1}{N} + \frac{1}{N}(1 - (x - Y(x_{13}))^T Q(x - Y(x_{13})) \right) / r_{\max} \tag{15}$$

where N marks the total number of waypoints, $Y(x_{13})$ is the coordinate of the waypoint towards which the quadrotor should currently be heading, and the maximum unnormalized reward is $r_{\max} = 2800.84$. The term $\frac{x_{13}-1}{N}$ increases as more waypoints are reached. It is enough to reach a neighborhood of 0.2 m on each axis around a waypoint to consider it reached and increment x_{13}. The second term of the reward function provides a score on how close the quadrotor is to the current waypoint, where we set $Q = \text{diag}(1, 1, 3,\ 0, 0, 0,\ 0, 0, 0,\ 0.1, 0.1, 0.1, 0)$. Thus, the largest weight is put on the z axis position of the quadrotor, so that the altitude is kept as stable as possible. Large weights are also set for the other two positions, so that the waypoints are reached. The last three weights, equal to 0.1, prevent the quadrotor from performing sudden rotations on any axis. Finally, the constant $r_{\max}$ normalizes the reward to $[0, 1]$.

Two experiments are run, in which five waypoints are defined, having the coordinates $(x, y, z) = (6, 0, 1)$, $(9, 2, 1)$, $(8, 5, 1)$, $(5, 4, 1)$, and $(1, 2, 1)$, all expressed in meters. The quadrotor starts from a hovering state, oriented towards the positive direction on the x axis, and should fly through the waypoints and stop at the last one. The experiments are performed with a budget of $n = 1000$ expansions for OPD and OASP, while OASP is configured with the ν-rule and $\beta = 18$.

In the first experiment the starting position is $(x, y, z) = (3, 0, 1)$. The left graph from Fig. 12 shows that the LQR controller obtains an almost perfect flight trajectory, because the quadrotor dynamics remain in a near-linear regime for which the LQR controller (designed based on linearized dynamics) works well. The side view of the trajectory indicates the quadrotor had some small variations in the flight altitude. The right graph from Fig. 12 presents the trajectories obtained by OPD and OASP. Although both planning methods find a solution to fly through the waypoints, the trajectories are sub-optimal. The solutions can be improved e.g. by increasing the computational budget, or by tuning the macro actions. Nevertheless, the flight path obtained with OASP is clearly better than the one of OPD.

The second experiment starts from a more distant location, $(x, y, z) = (25, 2, 1)$. i.e. to the right of the waypoints. The quadrotor has to turn back before flying towards the waypoints. The additional turning and the larger distance increase the number of required actions before the first waypoint is reached, and thus the required planning horizon. In contrast to the first experiment, LQR fails to find the correct flight direction and drives the quadrotor to an incorrect location. This is because the quadrotor evolves in a wider range of states where the linearized dynamics are no longer valid. On the other hand, as shown in the right graph from Fig. 13, OPD also fails to find a solution. The same computational budget is however enough for OASP to find a solution, which confirms the benefit of working with constrained planners.

7. Conclusions and Future Work

This paper has presented two algorithms for near-optimal control, tailored to compute solutions that switch a limited number of times between different discrete actions. The first algorithm, called optimistic switch-limited planning (OSP), uses a fixed switch constraint. By taking advantage of this property, our analysis showed that computation in the case of OSP is polynomial in the adaptive horizon of the computed solution, in contrast to the exponential complexity achieved by unconstrained OP algorithms; and therefore

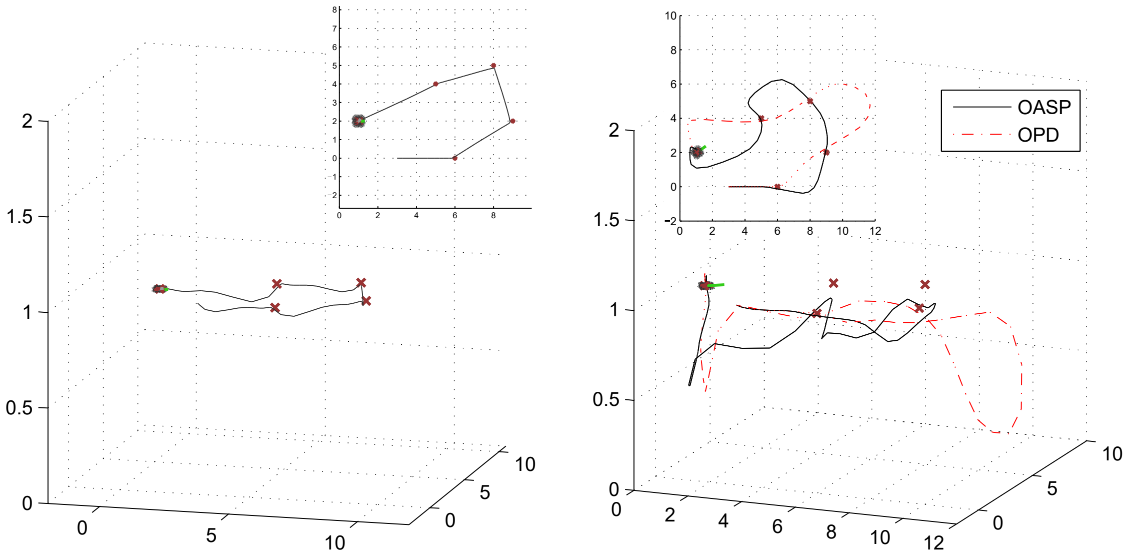


Figure 12: Quadrotor online simulations: path planned with LQR (left) and with OPD and OASP (right). Both figures show the flights in side view, while the insets show the trajectory in top view.

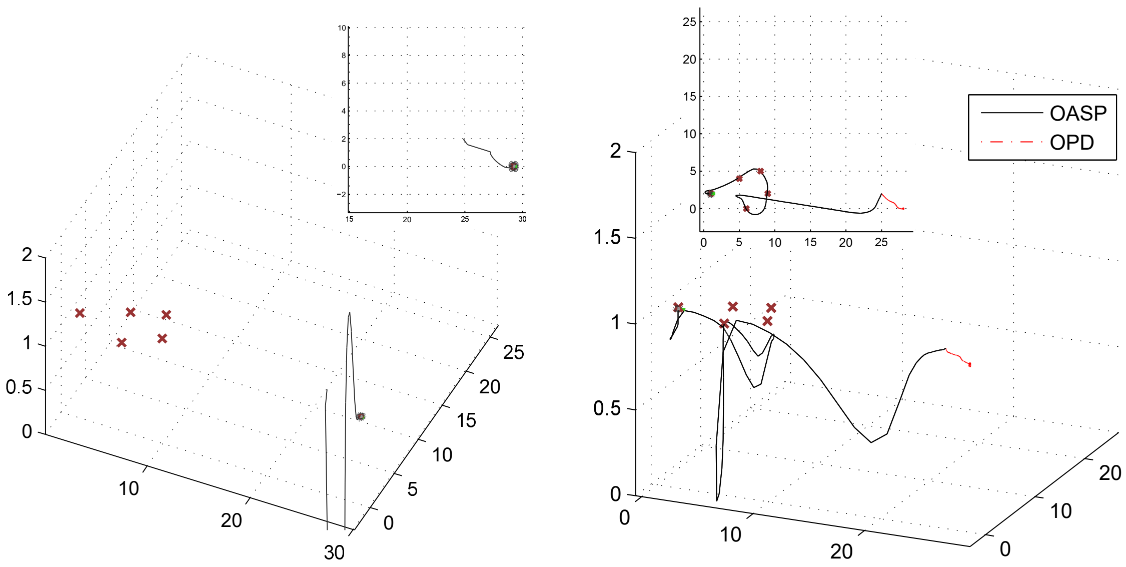


Figure 13: Quadrotor online simulations: path planned with LQR (left) and with OPD and OASP (right).

that OSP approaches the constrained optimum fast. Extensive simulations confirmed the analytical properties of the algorithm and illustrated that it works well in receding horizon. As a more general class of applications, we also explained how the switch constraint can be applied to limit bandwidth requirements in networked control systems. Furthermore, the algorithm was applied to a real rotational pendulum, with good results.

Since OSP uses a fixed value of the switch constraint S, the optimal solution may fall outside its search space. In the second part of the paper, we developed an extension of OSP, called OASP, introducing an *increment rule* that adapts S depending on the evolution of the cumulative rewards. OASP was evaluated for two possible increment rules for S, analyzing the performance of the algorithm in several special cases of search trees. Although a general characterization of the performance of OASP was not obtained yet, the analysis in these cases confirmed that, unlike OSP, OASP exploits the adaptation rule to obtain a near-optimal solution. Furthermore, OASP outperforms OPD for certain types of control problems, and gracefully degrades to the performance of OPD in other cases. OASP was evaluated in simulations of rotational pendulum swing-up and of quadrotor trajectory control, confirming the benefits of the novel algorithm.

Future work will focus on comparisons with more classical model-predictive control methods, analyzing the closed-loop performance of OSP and OASP, and on completing the analysis of OASP with a general characterization of its convergence rate.

Acknowledgments

This work was supported by a grant from Siemens, Reference No. 7472/3202246859, by the Agence Universitaire de la Francophonie (AUF) and the Romanian Institute for Atomic Physics (IFA) under the AUF-RO project 09-AUF NETASSIST, and by a grant of the Romanian National Authority for Scientific Research and Innovation, CNCS/CCCDI –UEFISCDI, project number PN-III-P2-2.1-BG-2016-0140.

References

[1] R. Munos, The optimistic principle applied to games, optimization and planning: Towards foundations of Monte-Carlo tree search, Foundations and Trends in Machine Learning 7 (1) (2014) 1–130.

[2] L. Kocsis, C. Szepesvári, Bandit based Monte-Carlo planning, in: Proceedings 17th European Conference on Machine Learning (ECML-06), Berlin, Germany, 2006, pp. 282–293.

[3] S. Bubeck, R. Munos, Open loop optimistic planning, in: Proceedings 23rd Annual Conference on Learning Theory (COLT-10), Haifa, Israel, 2010, pp. 477–489.

[4] L. Buşoniu, R. Munos, Optimistic planning for Markov decision processes, in: Proceedings 15th International Conference on Artificial Intelligence and Statistics (AISTATS-12), La Palma, Canary Islands, Spain, Vol. 22 of JMLR Workshop and Conference Proceedings, 21–23 April 2012, pp. 182–189.

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

[6] S. Gelly, Y. Wang, R. Munos, O. Teytaud, Modification of UCT with patterns in Monte-Carlo Go, Tech. rep., INRIA, Paris-Sud, France (2006).

[7] B. De Schutter, B. De Moor, Optimal traffic light control for a single intersection, European Journal of Control 4 (3) (1998) 260–276.

[8] H. van Ekeren, R. Negenborn, P. van Overloop, B. De Schutter, Time-instant optimization for hybrid model predictive control of the Rhine-Meuse delta., Journal of Hydroinformatics 15 (2) (2013) 271–292.

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

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

[11] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, S. Colton, A survey of monte carlo tree search methods, IEEE Transactions on Computational Intelligence and AI in Games 4 (1) (2012) 1–43. doi:10.1109/TCIAIG.2012.2186810.

[12] S. Edelkamp, S. Schrödl, Heuristic Search: Theory and Applications, Morgan Kauffman, 2012.

[13] S. M. La Valle, Planning Algorithms, Cambridge University Press, 2006.

[14] B. De Schutter, Optimal control of a class of linear hybrid systems with saturation, SIAM Journal on Control and Optimization 39 (3) (2000) 835–851.

[15] J. Alende, Y. Li, M. Cantoni, A {0,1} linear program for fixed-profile load scheduling and demand management in automated irrigation channels, in: Proceedings 48th IEEE Conference on Decision and Control (CDC-09), Shanghai, China, 2009, pp. 597–602.

[16] C. O. Martinez, A. Bemporad, A. Ingimundarson, V. P. Cayuela, On hybrid model predictive control of sewer networks, identification and control, in: R. S. S. Pena, J. Q. C. V. Puig Cayuela (Eds.), Identification and Control: The Gap between Theory and Practice, Springer, 2007, pp. 87–114.

[17] A. Sadowska, B. De Schutter, P.-J. van Overloop, Event-driven hierarchical control of irrigation canals, in: Proceedings of the USCID Seventh International Conference on Irrigation and Drainage, Phoenix, Arizona, 2013.

[18] C. Liu, W.-H. Chen, J. Andrews, Piecewise constant model predictive control for autonomous helicopters, Robotics and Autonomous Systems 59 (7) (2011) 571–579.

[19] M. J. Alves, J. Clímaco, A review of interactive methods for multiobjective integer and mixed-integer programming, European Journal of Operational Research 180 (1) (2007) 99–115.

[20] X. Li, T. E. Marlin, Model predictive control with robust feasibility, Journal of Process Control 21 (3) (2011) 415–435.

[21] A. Bemporad, M. Morari, V. Dua, E. N. Pistikopoulos, The explicit linear quadratic regulator for constrained systems, Automatica 38 (1) (2002) 3–20.

[22] P. Tøndel, T. A. Johansen, A. Bemporad, An algorithm for multi-parametric quadratic programming and explicit MPC solutions, Automatica 39 (3) (2003) 489–497.

[23] C. Wen, X. Ma, B. E. Ydstie, Analytical expression of explicit MPC solution via lattice piecewise-affine function, Automatica 45 (4) (2009) 910–917.

[24] F. Bayat, T. A. Johansen, A. A. Jalali, Using hash tables to manage the time-storage complexity in a point location problem: Application to explicit model predictive control, Automatica 47 (3) (2011) 571–577.

[25] K. Mathe, L. Busoniu, R. Munos, B. De Schutter, Optimistic planning with a limited number of action switches for near-optimal nonlinear control, in: Proceedings IEEE 53rd Annual Conference on Decision and Control (CDC), Los Angeles, CA, USA, 15–17 December 2014, pp. 3518–3523.

[26] A. Hoorfar, M. Hassani, Inequalities on the Lambert W function and hyperpower function, Journal of Inequalities in Pure and Applied Mathematics 9 (2) (2008) 5–9.

[27] C. De Persis, P. Frasca, Robust self-triggered coordination with ternary controllers, IEEE Transactions on Automatic Control 58 (12) (2013) 3024–3038.

[28] T. Wensveen, L. Buşoniu, R. Babuška, Real-time optimistic planning with action sequences, in: 20th International Conference on Control Systems and Computer Science (CSCS-15), Bucharest, Romania, 2015, pp. 923–930.

[29] D. E. Kirk, Optimal Control Theory. An Introduction, Dover Publications, Inc., 2004.

[30] F. L. Lewis, D. L. Vrabie, V. L. Syrmos, Optimal Control, 3rd Edition, John Wiley & Sons, Inc., 2012.

[31] A. K. Máthé, Nonlinear control for commercial drones in autonomous railway maintenance, Ph.D. thesis, Technical University of Cluj-Napoca, Romania (September 2016).
URL `http://rocon.utcluj.ro/files/MATHE_drones_in_railways.pdf`

[32] Z. Dydek, A. Annaswamy, E. Lavretsky, Adaptive control of quadrotor UAVs: A design trade study with flight evaluations, IEEE Transactions on Control Systems Technology 21 (4) (2013) 1400–1406.

[33] Parrot AR.Drone.
URL `http://ardrone2.parrot.com/ar-drone-2/specifications/`