Real-Time Optimistic Planning with Action Sequences

Thijs Wensveen, Lucian Buşoniu, Robert Babuška

***Abstract*—Optimistic planning (OP) is a promising approach for receding-horizon optimal control of general nonlinear systems. This generality comes however at large computational costs, which so far have prevented the application of OP to the control of nonlinear physical systems in real-time. We therefore introduce an extension of OP to real-time control, which applies open-loop sequences of actions in parallel with finding the *next* sequence from the *predicted* state at the end of the current sequence. Exploiting OP guarantees, we provide conditions under which the algorithm is provably feasible in real-time, and we analyze its performance. We report successful real-time experiments for the swingup of an inverted pendulum, as well as simulation results for an acrobot, where the impact of model errors is studied.**

I. Introduction

We consider the problem of optimally controlling a nonlinear system in discrete time, so that a discounted infinite-horizon sum of rewards is maximized [1]. An optimistic planning (OP) [2] type of algorithm is adopted, which uses the model to compute an adaptive-horizon sequence of control actions that is near-optimal for the current state of the system, applies the first action of this sequence, and then repeats the procedure. OP methods are therefore a type of receding-horizon model-predictive control (MPC) [3]. However, they integrate a mixture of artificial intelligence, machine learning, and global optimization ideas, which leads to quite a different approach from classical MPC. First, OP is highly general, allowing arbitrary nonlinear system dynamics and nonquadratic reward functions. Second, it provides a quantitative relationship between computation invested and near-optimality.

Several OP algorithms exist [4]–[7] and have shown good performance in simulation [7], [8]. Here we will consider the simplest such algorithm: optimistic planning for deterministic systems (OPD) [4]. OPD works for a finite number of discrete actions, and explores the infinitely deep tree of possible action sequences, by always refining further a sequence with the largest upper bound – hence the "optimistic" label. OPD is relatively unaffected by the state dimensionality. It is typically used with a fixed computational budget, meaning that the algorithm performs a fixed number of tree node expansions before returning a sequence of actions.

An unavoidable price must be paid for the generality of OPD: computational complexity. The algorithm generally requires a budget that is exponential (in a well-characterized way) in the depth of the tree explored, and this depth directly relates to solution quality. Due to this, to our knowledge no real-time application of OPD to physical control systems has been reported so far. The main problem is that OPD is applied in receding horizon while assuming that it takes negligible time to compute a sequence after measuring the state. This is of course not true, and so real-time applicability is hampered.

We propose here an extension of OP that solves this problem: real-time OP with sequences (RTOPS). Instead of single actions, RTOPS applies open-loop sequences, increasing the time available for planning. While the current sequence is being applied on a separate thread, replanning of a new sequence starts immediately, from the predicted state at the end of the current sequence. In this way, computation can take advantage of the entire available time, while actions are always available to apply to the system. Exploiting the complexity analysis of OPD, we derive conditions on the computation budget and the length of the applied subsequence that guarantee the real-time feasibility of RTOPS. We examine the performance impact of the length of the applied initial subsequence. Real-time control results for swingup of an inverted pendulum are shown, in addition to simulation results for the acrobot (a two-link robot arm with a fixed end-joint and an actuated middle joint). For the acrobot, we also study the impact of model errors.

Our closest related work is [9], where sequences computed with OP are also applied, but with a different goal: to minimize communications in networked control. However, [9] still assumes that sequences are found near-instantaneously after measuring the state. RTOPS addresses this by replanning from the predicted state, in parallel to the control thread.

Applying sequences to increase the time available for computation is a simple but effective idea, and so it has been investigated before. As early as 1999, [10] introduced the idea of intermittently moving the horizon in an MPC setting, applying a sequence of actions and then shifting the horizon by the length of the applied sequence. In [11] another intermittent approach is described, very similar to ours: a predicted state, based on the current state and the current action sequence, is used to find the next action sequence. This approach is studied in more detail in [12], where it is split into three modules: a state predictor, an open-loop control policy, and an intermittent controller. In [13] the computational delay δ is explicitly included in the design, by only computing at step k a sequence for the interval $[k+\delta, k+N]$ where N is the horizon. Similarly, [14] takes a sampled-data approach and designs the control at the ith step for the time interval $[t_i + \delta, t_{i+1} + \delta]$, already covering the delay at the next computation. Both [13] and [14] focus on stability guarantees.

All these works *assume* the existence of a maximum computation time that makes the approach feasible. In contrast, we provide an *explicit characterization* of how (and when) this computation time can be guaranteed. The specific properties of OPD are essential to this characterization. Our overall main novelty is integrating OP and real-time control.

Next, Section II formalizes the optimal control problem and OPD. Section III describes RTOPS and its analysis, while Section IV gives our experiments. Section V concludes.

T. Wensveen and R. Babuška are with the Delft Center for Systems and Control, Delft University of Technology, the Netherlands (thijswensveen@gmail.com, r.babuska@tudelft.nl). L. Buşoniu is with the Department of Automation, Technical University of Cluj-Napoca, Romania (lucian@busoniu.net). This work was supported by a grant of the Romanian National Authority for Scientific Research, CNCS-UEFISCDI, project number PNII-RU-TE-2012-3-0040.

II. Optimal control problem and OPD algorithm

We consider deterministic, discrete-time nonlinear systems, with dynamics described by $x_{k+1} = f(x_k, u_k)$, with $x_k \in X$ the state and $u_k \in U$ the action at time step k. Here X and U are the state and action spaces, respectively. A reward is assigned to each state transition, defined by $r_k = \rho(x_k, u_k)$, where $\rho : X \times U \to \mathbb{R}$ is the reward function. We adopt for a large part the notations as used in [9], as they are suitable for the real-time extension here.

In the sequel it is assumed firstly that all rewards are in the interval $[0, 1]$. Note however that any bounded reward function can be scaled to this interval without changing the optimal solution. The main way to achieve boundedness is by saturating a possibly unbounded original reward function. Moreover, the physical limitations of the system may be meaningfully modeled by saturating the states and actions, in which case a reward bound follows from the saturation limits. Secondly, we also assume that the action space is discrete or discretized, with cardinality $|U| = K$. Many systems have inherently finitely-many actions, because they are controlled by switches, e.g. water barriers and sluices [15]. When the actions are originally continuous, discretization reduces performance, but the loss is often manageable. On the other hand, our approach for real-time control works for continuous-action planning algorithms as well, such as our method in [16] – but the complexity analysis prerequisite for real-time guarantees is not yet available for this method.

The objective is to optimally control the system, by applying a sequence of actions that maximizes cumulative rewards. To formalize this, we define $\boldsymbol{u}_d = [u_0, u_1 \dots u_{d-1}]$ to be an action sequence of length d. The value of an (infinitely long) sequence of actions starting from state x_0 is then:

$$v_{x_0}(\boldsymbol{u}_\infty) = \sum_{k \geq 0} \gamma^k \rho(x_k, u_k) \tag{1}$$

where u_k are the actions from the infinite length sequence $\boldsymbol{u}_\infty$, and the states are the result of applying these actions, $x_{k+1} = f(x_k, u_k)$. The discount factor γ is constrained to the interval $[0, 1)$ and is used to differentiate between the importance of present rewards and future rewards. Discounting is necessary for our planning algorithms, and many other works in optimal control use it. Given (1), the optimal value is defined as $v^*_{x_0} = \sup_{\boldsymbol{u}_\infty} v_{x_0}(\boldsymbol{u}_\infty)$.

In the OPD algorithm, a given amount of computation is imposed, where computation is considered proportional to the number of calls to the model (transition and reward functions); and the algorithm is expected to return the best possible sequence of actions within this allowed computation. To this end, OPD searches the space of all infinite length action-sequences starting from state x_0, by developing a look-ahead tree in which nodes represent states and branches represent actions, see Figure 1. Each node is reachable following a unique path in the tree, corresponding to a unique sequence of actions. Note, however, that states in the tree are not necessarily unique, as it is possible that identical states are reached from the initial state via different paths.

Three quantities are defined for any sequence $\boldsymbol{u}_d$: a lower bound on the values (1) of the infinite-length action sequences

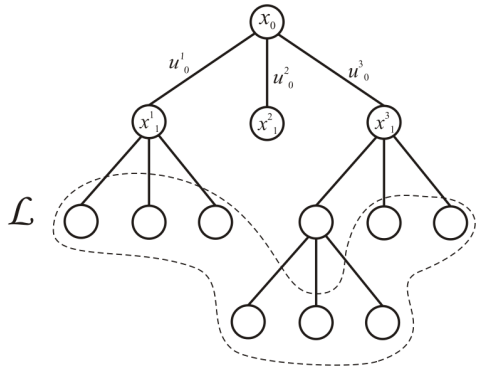

Fig. 1. Optimistic planning tree example for $K = 3$ actions. Branches are labeled by the action u^l_d, where l is again the index of the action and d is the depth of the node at which the action was applied, and the corresponding children nodes are labeled by the resulting states x^l_d. The leaves $\mathcal{L}$ are enclosed by a dashed line.

$\boldsymbol{u}_\infty$ starting from state x_0 and having $\boldsymbol{u}_d$ as initial subsequence, an upper bound on these values, and an optimal value among them. The lower bound is:

$$\ell_{x_0}(\boldsymbol{u}_d) = \sum_{k=0}^{d-1} \gamma^k \rho(x_k, u_k) \tag{2}$$

where u_k are again actions from the sequence $\boldsymbol{u}_d$. This is a lower bound because all future rewards are non-negative. The upper bound is defined as:

$$b_{x_0}(\boldsymbol{u}_d) = \ell_{x_0}(\boldsymbol{u}_d) + \frac{\gamma^d}{1-\gamma} \tag{3}$$

where $\gamma^d + \gamma^{d+1} + \dots = \frac{\gamma^d}{1-\gamma}$ is the reward obtained if all future rewards after depth d would be equal to the maximum reward of 1. Finally, the optimal value among infinite length sequences starting with $\boldsymbol{u}_d$ is:

$$v_{x_0}(\boldsymbol{u}_d) = \ell_{x_0}(\boldsymbol{u}_d) + \gamma^d v^*_{x_d} \tag{4}$$

that is, the value of following actions $\boldsymbol{u}_d$ from x_0 and continuing optimally from x_d onwards.

OPD develops the tree optimistically, by always expanding a leaf node with the largest upper bound: $\operatorname{argmax}_{\boldsymbol{u}_d \in \mathcal{L}} b_{x_0}(\boldsymbol{u}_d)$, motivating the algorithm's name. Each expansion generates K new nodes, one for each of the K actions available in the current state, and so an expansion requires K calls to the model. A computational budget of n node expansions is imposed, and after exhausting it, OPD returns a sequence that has the largest *lower* bound among all sequences explored: $\boldsymbol{u}^0_{d_0} = \operatorname{argmax}_{\boldsymbol{u}_d \in \mathcal{L}} \ell_{x_0}(\boldsymbol{u}_d)$, interpreted as a safe – rather than optimistic – choice.

In [2], [4] it is shown that the length d_0 of the returned action sequence $\boldsymbol{u}^0_{d_0}$ is guaranteed to be the largest expanded depth in the tree,[1] and that the suboptimality of this sequence is bounded as follows:

$$v^*_{x_0} - v_{x_0}(\boldsymbol{u}^0_{d_0}) \leq \frac{\gamma^{d_0}}{1-\gamma} \tag{5}$$

In this case we say that the sequence is "$\frac{\gamma^{d_0}}{1-\gamma}$-optimal".

[1] Or one action longer, in which case by convention we eliminate this action.

Now, the depth reached by OPD depends on the size of the near-optimal subtree $\mathcal{T}^*(x_0)$ [2], [9]:

$$\mathcal{T}^*(x_0) = \left\{ \boldsymbol{u}_d \mid d \geq 0,\ v^*_{x_0} - v_{x_0}(\boldsymbol{u}_d) \leq \frac{\gamma^d}{1-\gamma} \right\} \quad (6)$$

This is because OPD *only* expands nodes that belong to this subtree. To formalize the relationship between computation and depth, define the asymptotic branching factor $\kappa(x_0) \in [1, K]$ of this subtree as follows:

$$\kappa(x_0) = \limsup_{d \to \infty} |\mathcal{T}^*_d(x_0)|^{1/d} \quad (7)$$

where $|\mathcal{T}^*_d(x_0)|$ represents the number of nodes in $\mathcal{T}^*(x_0)$ at depth d. Branching factor $\kappa(x_0)$ is a measure of the complexity of OPD. If $\kappa(x_0) > 1$, reaching a depth d in the tree OPD requires $n = O(\kappa(x_0)^d)$ expansions [2].[2] In the special case that $\kappa(x_0) = 1$, only $n = O(d)$ expansions are needed. Intuitively this can be explained as follows: if there is only one action at each depth that is near-optimal as $d \to \infty$ (i.e. $\kappa(x_0) = 1$), then asymptotically each expansion will increase the depth of the tree. Otherwise, if more near-optimal actions are available at each depth, the near-optimal subtree grows exponentially and the general formula holds.

By solving for the depth d_0 in the number of expansions in the near-optimal subtree above (either $n = O(\kappa(x_0)^d)$ or $n = O(d)$) and then replacing this depth in the near-optimality bound (5), we find a bound stated directly in terms of the computation n invested. All this is summarized in the following theorem, together with a synthesis of the other properties we discussed above. In this theorem, $c(x_0)$ is related to the constants in the $O(\cdot)$ expressions above, which are used e.g. to cover the nonasymptotic regime of tree growth.

Theorem 1 (see [2], [9]): When called with budget n at state x_0, OP returns a sequence $\boldsymbol{u}^0_{d_0}$ of length d_0 equal to the largest expanded depth on the tree, and which is $\frac{\gamma^{d_0}}{1-\gamma}$-optimal. Further:

- When $\kappa(x_0) > 1$, a depth of $d_0 \geq c(x_0) \frac{\log n}{\log \kappa(x_0)}$ is reached and the sequence is $O(n^{-\frac{\log 1/\gamma}{\log \kappa(x_0)}})$-optimal.
- When $\kappa(x_0) = 1$, a depth of $d_0 \geq c(x_0) n$ is reached and the sequence is $O(\gamma^{c(x_0)n})$-optimal.

where $c(x_0) > 0$ is a state-dependent constant.

III. Real-Time Optimistic Planning with Action Sequences

A. Algorithm

In principle, OPD should be applied at every step (every state encountered) during the interaction with the system, where it would return an action sequence. The first action from this sequence would be applied to the system, the new state would be measured and then the whole procedure would be repeated in receding horizon. The problem with this is the implicit assumption that OPD takes negligible time to execute (much less than a sampling time), which is not true. Being designed for very general problems, OPD is relatively computationally costly. So the procedure is not implementable in practical, real-time control.

To address this issue, we propose the following approach. OPD is called to obtain a sequence of actions, which is no different from the typical setting. However, instead of just applying the first action of the sequence, the entire sequence or an initial subsequence thereof will be used. While this current sequence is being applied, on a different computation thread a next action sequence is being computed from the *predicted* state at the end of the current sequence, to make sure that this new sequence will be appropriate at the time the current one runs out. At that time, this new sequence starts being applied, and so on. Thus a significant amount of time (equal to the sampling interval times the length of the applied sequence) is available to execute OPD at each call. Since OPD returns sequences of actions by default, the complexity of the planning is not increased with this approach, effectively increasing the ability to meet real-time constraints. The resulting algorithm is referred to as Real-Time Optimistic Planning with Sequences, RTOPS. We have to require that initially enough (but finite) time is available for computing the first sequence, as no previous sequence is available to ensure the real-time constraints are met. An alternative would be to apply an a priori, suboptimal controller during this time if one is available, e.g. if the system is at an equilibrium, a sequence of actions with value zero. This zero-sequence solution is the one we adopt in our experiments.

The following notations are used to formalize the algorithm: $\boldsymbol{u}^m_{d_m}$ is an action sequence of length d_m, computed at the mth call to OPD, $m = 0, 1, \ldots$. The indices m are omitted when the calculation time is not relevant. Predictions are indicated with a hat, e.g. $\hat{x}_{k+d'}$ is the predicted state at time step $k+d'$.

RTOPS is driven by two parameters: the computation n allowed at each call, and the length d' of the initial subsequence to apply from the sequence returned (which may be of larger depth d_m). These two parameters can in general be set independently, although of course they must satisfy constraints to ensure real-time applicability, and our analysis below details these constraints. Algorithm 1 summarizes the procedure, where OPD(x, n) means that OPD is applied in state x with budget n. We emphasize that model (and therefore prediction) errors are not explicitly taken into account in the algorithm and analysis; however, in Section IV-A we empirically study the effect of model mismatch, and analyzing this effect is our first priority in ongoing work.

B. Real-time guarantees

The applicability of RTOPS in real-time requires one crucial condition. Simply put, this condition is that a sequence of length $d \geq d'$ must be computable within a time that is at most equal to the time it takes the system to go through d' actions. Denote the execution time of one node expansion by T_e, then a budget n can be chosen as long as $nT_e \leq d'T_s$, where T_s is the sampling interval. The condition then becomes that OPD called with budget n always returns a sequence of length $d \geq d'$. To make this condition explicit, we need to take into account the complexity of the problem, as measured by the asymptotic branching factor κ.

[2]Let $g, h : (0, \infty) \to \mathbb{R}$, then statement $g(t) = O(h(t))$ for large t means that $\exists t_0, c > 0$ so that $g(t) \leq ch(t),\ \forall t \geq t_0$.

Algorithm 1 Pseudo-code for RTOPS

Input: initial state x_0, budget n, subsequence length d'

$k \leftarrow 0$, $m \leftarrow 0$
apply OPD(x_0, n) to obtain $\boldsymbol{u}_{d_0}^0$
send subsequence $\boldsymbol{u}_{d'}^0$ to buffer
(buffer sends actions to system, while planning continues)
loop
    measure current state x_k
    simulate $\boldsymbol{u}_{d'}^m$ from x_k, to find prediction $\hat{x}_{k+d'}$
    apply OPD$(\hat{x}_{k+d'}, n)$, obtaining $\boldsymbol{u}_{d_{m+1}}^{m+1}$
    (if needed, wait until buffer is empty)
    send subsequence $\boldsymbol{u}_{d'}^{m+1}$ to buffer
    $k \leftarrow k + d'$, $m \leftarrow m + 1$
end loop

To provide a uniform guarantee, we introduce the worst-case branching factor and constant c among all states, $\bar{\kappa} = \sup_{x \in X} \kappa(x)$, $\underline{c} = \inf_{x \in X} c(x)$. We will analyze the theoretical case where $\bar{\kappa}$ is known, making a further distinction between $\bar{\kappa} > 1$ and $\bar{\kappa} = 1$. Most problems are likely to have a branching factor with $1 < \bar{\kappa} < K$. However, in practice $\bar{\kappa}$ and $\underline{c}$ are often not known a priori, so to obtain a safe real-time constraint we must also consider an absolute worst-case scenario where for some states all nodes at each depth need to be expanded, leading to $\bar{\kappa} = K$. In this case, OPD would work the same as uniform planning, or breadth-first search, where nodes are expanded in the order of their depth (and the constant is not needed).

Theorem 2: RTOPS is feasible in real-time if the settings n and d' jointly satisfy the following conditions:

$$n \leq d' T_s / T_e \tag{8}$$

and either of:

- for $\bar{\kappa} > 1, \underline{c}$ known: $d' \leq \underline{c} \dfrac{\log n}{\log \bar{\kappa}}$ (9)
- for $\bar{\kappa} = 1, \underline{c}$ known: $d' \leq \underline{c} n$ (10)
- for $\bar{\kappa}, \underline{c}$ unknown: $d' \leq \dfrac{\log[n(K-1)+1]}{\log K} - 1$ (11)

Proof: Consider first $\bar{\kappa} > 1, \underline{c}$ known. Then, for any state x for which $\kappa(x) > 1$, by Theorem 1 OPD reaches depth $d \geq c(x) \frac{\log n}{\log \kappa} \geq \underline{c} \frac{\log n}{\log \bar{\kappa}}$, by definition of $\bar{\kappa}$ and $\underline{c}$. Hence, taking d' that satisfies (9) ensures $d \geq d'$ is reached, and a subsequence of length d' is available. Further, (8) ensures that computation finishes within d' steps, and therefore the algorithm is viable in real-time. States where $\kappa(x) = 1$ are trivially handled by noticing that $c(x) d \geq \underline{c} n \geq \underline{c} \frac{\log n}{\log \bar{\kappa}}$.

The case $\bar{\kappa} = 1$ is handled similarly using the appropriate formula in Theorem 1.

Finally, when the entire tree must be expanded, given a value of n take d to be the smallest depth so that $n \leq \sum_{i=0}^{d} K^i = \frac{K^{d+1}-1}{K-1}$, which means at least some nodes at d were expanded, and the sequence returned has at least length d. Solving for d we get $d \geq \frac{\log[n(K-1)+1]}{\log K} - 1$, and so choosing d' to satisfy (11) ensures real-time feasibility. ∎

So far n was set independently from d'. It will often be best to simply select the largest n allowed by (8) so as to fully exploit the available time. Otherwise, the algorithm will waste time, which is not desirable unless other considerations must be taken into account, such as battery life if RTOPS is applied on a mobile device. Therefore, we next study the case when only d' is freely selected, while the budget is kept fixed to the maximum possible, $n = \lfloor d' T_s / T_e \rfloor$ where $\lfloor \cdot \rfloor$ is the floor operator. In this case the feasibility condition is given only in terms of d'.

Corollary 3: When $n = \lfloor d' T_s / T_e \rfloor$, RTOPS is feasible in real-time if the setting for d' satisfies either of:

- $\bar{\kappa} > 1, \underline{c}$ known: $d' \dfrac{T_s}{T_e} - \bar{\kappa}^{d'/\underline{c}} - 1 \geq 0$ (12)
- $\bar{\kappa} = 1, \underline{c}$ known: $d' (\underline{c} \dfrac{T_s}{T_e} - 1) - \underline{c} \geq 0$ (13)
- $\bar{\kappa}, \underline{c}$ unknown: $(d' \dfrac{T_s}{T_e} - 1)(K-1) - K^{d'+1} + 1 \geq 0$ (14)

Proof: In general, conditions (9)-(11) can all be written as $d' \leq g(n)$ for some strictly increasing function g. We have $n > d' T_s / T_e - 1$, so if we ensure $d' \leq g(d' T_s / T_e - 1)$, then due to the increasing nature of g, $d' \leq g(n)$ is implied. The conditions then are simply obtained by filling in from (9)-(11) the appropriate forms of g in each case. ∎

Several remarks are in order. First, even if branching factor $\bar{\kappa}$ and constant $\underline{c}$ are assumed known, all the conditions are conservative in general, because $\bar{\kappa}$ and $\underline{c}$ are worst-case values. In fact, given the chosen budget n (much) larger depths d may be reached, improving the quality of the sequences, since their near-optimality is $\frac{\gamma^d}{1-\gamma}$ by (5). Conservativeness of course increases even more when $\bar{\kappa}$ is unknown.

Second, it is possible that no values of d' and n exist that satisfy the conditions. This simply means that for the problem considered, OPD requires too much computation to be applied in real-time. Either a more powerful processor is needed or the code must be optimized (e.g., for the system simulator).

Third, the execution time T_e can be found as follows in practice. OPD is applied offline from some representative system states and its execution time is measured. Dividing the overall time by the total number of expansions gives an approximation of T_e.

Last but not least, measuring the computation budget by the number of expansions implicitly assumes that the other operations (computing lower and upper bounds, navigating the tree to find optimistic sequences, etc.) take negligible time. This is justified by the fact that OPD targets complex nonlinear systems, and simulating such systems is often computationally intensive (e.g. by requiring numerical integration). In practice it may however be important to take into account tree operations, and we provide a heuristic to do this when selecting T_e. The time complexity of tree operations in OPD depends on the branching factor κ. It is highest for $\kappa = 1$, when the algorithm has to navigate at each iteration along a path of depth $O(n)$, giving overall time complexity of $O(n^2)$ (for $\kappa > 1$, the time complexity is only $O(n \log n)$ [6]). Therefore, to avoid computation taking longer in practice than expected, in the preliminary experiment to find T_e the tree could be forced to have $\kappa = 1$, leading to the largest complexity in tree operations. This can be done by creating an artificial reward

function that always gives rewards of 1 for one action and rewards of 0 for all other actions.

C. Performance guarantees

Besides real-time applicability, it is also important to understand the quality of the solution produced by RTOPS. In [9] an important property was shown for OPD when it applies sequences of actions: a near-optimality guarantee holds that only depends on the length d_0 of the first sequence computed, and is not affected by the length d' of the subsequences applied. This property directly applies to RTOPS, since it finds the first sequence in the same way as the algorithm in [9]:

Theorem 4 ([9]): RTOPS satisfies:

$$v^*_{x_0} - v_{x_0}([\boldsymbol{u}^0_{d'}, \boldsymbol{u}^1_{d'}, \dots]) \leq \frac{\gamma^{d_0}}{1-\gamma} \tag{15}$$

where $[\boldsymbol{u}^0_{d'}, \boldsymbol{u}^1_{d'}, \dots]$ is the infinitely long sequence applied by RTOPS in closed-loop and d_0 on the right side is the depth reached on the first planning instance.

Here and in the sequel, operator $[\cdot, \dots, \cdot]$ denotes the concatenation of two or more sequences. Intuitively, the near-optimality of the long sequence $\boldsymbol{u}^0_{d_0}$ is $\frac{\gamma^{d_0}}{1-\gamma}$ due to Theorem 1, and in addition to that replanning earlier will retain at least the same bound as the original sequence.

Although this near-optimality *bound* does not change with the length of the applied subsequence, this does not mean that the actual performance is the same. While no definite answer can be given as to whether longer or shorter subsequences perform better, since either case is possible depending on the problem [9], bounds can be given on the performance loss in both cases. Recall that $\boldsymbol{u}^0_{d'}$ is the initial, applied subsequence of length d', which we take here strictly smaller than the length d_0 of the full sequence $\boldsymbol{u}^0_{d_0}$ returned by RTOPS. Also, $\boldsymbol{u}^1_{d_1}$ is the sequence found by RTOPS when re-planning from state $x_{d'}$, which results from applying sequence $\boldsymbol{u}^0_{d'}$.

Theorem 5: Applying a shorter subsequence $\boldsymbol{u}^0_{d'}$ and re-planning a new sequence $\boldsymbol{u}^1_{d_1}$ may lose or gain value, compared to the value $v(\boldsymbol{u}^0_{d_0})$ of the full sequence returned by RTOPS, up to the following bounds:

$$v_{x_0}(\boldsymbol{u}^0_{d'}) - \frac{\gamma^{d_0}}{1-\gamma} \leq v_{x_0}(\boldsymbol{u}^0_{d_0}) \leq v_{x_0}([\boldsymbol{u}^0_{d'}, \boldsymbol{u}^1_{d_1}]) + \frac{\gamma^{d'+d_1}}{1-\gamma} \tag{16}$$

Moreover, the bounds are tight in general: for either bound, a problem exists where the bound holds with equality.

Proof: The second inequality was proven in [9], and an example was given there in which the bound is tight. To prove the first inequality, observe that $v^*_{x_0} - v_{x_0}(\boldsymbol{u}^0_{d_0}) \leq \frac{\gamma^{d_0}}{1-\gamma}$ from Theorem 1, and also that $v^*_{x_0} - v_{x_0}(\boldsymbol{u}^0_{d'}) \geq 0$ by definition of the optimal value. Then:

$$v_{x_0}(\boldsymbol{u}^0_{d'}) - v_{x_0}(\boldsymbol{u}^0_{d}) =$$
$$\left[v^*_{x_0} - v_{x_0}(\boldsymbol{u}^0_{d_0})\right] - \left[v^*_{x_0} - v_{x_0}(\boldsymbol{u}^0_{d'})\right] \leq \frac{\gamma^{d_0}}{1-\gamma}$$

which is equivalent to the desired result.

An example is constructed to prove that this bound is also tight, as shown in Figure 2. In this example, all rewards are equal to zero in the explored tree at the first planning instance, so OPD chooses the sequence arbitrarily, by some tie-breaking rule. Then, the tree can be constructed in such a way that the tie-breaking rule selects a sequence $\boldsymbol{u}_d$ after which, even with replanning, the high-reward path is unreachable. Re-planning after applying the shorter subsequence $\boldsymbol{u}_{d'}$, however, does result in finding the high-reward sequence, because it is part of the subtree that has $x_{d'}$ as its root. Since the high-reward path starts at d_0, replanning later loses a value of $\frac{\gamma^{d_0}}{1-\gamma}$. ■

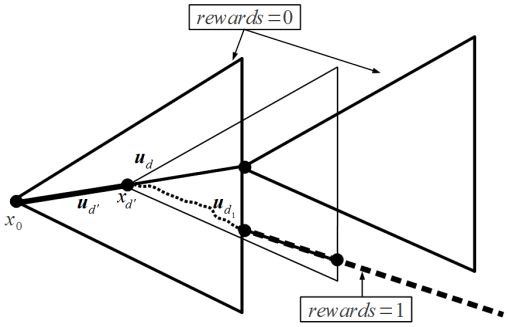

Fig. 2. Worst-case example of performance-loss when applying longer sequences. Sequence superscripts have been dropped for readability.

Intuitively, the first inequality of Theorem 5 says that, when compared to applying the full-length sequence, the shorter sequence cannot *gain* more than $\frac{\gamma^{d_0}}{1-\gamma}$, i.e. the uncertainty of the full-length sequence; and the second inequality says that the shorter sequence followed by replanning cannot *lose* more than $\frac{\gamma^{d'+d_1}}{1-\gamma}$, i.e. the uncertainty of the combined sequence after replanning.

IV. Experimental Study

This section describes experiments conducted to analyze the practical performance of RTOPS on two systems: an acrobot simulation and a real inverted pendulum. For the simulated acrobot, a part of the experiments have been done with the addition of simulated model mismatches. In a real system such as the inverted pendulum setup, it is of course inevitable that model errors are present.

A. Acrobot simulation

The acrobot is a two-link robot arm, where one joint is fixed and only the middle joint is actuated, see Figure 3. The aim is to swing up both links to the upright position, and the problem gets its name from the similarity to a gymnast (acrobat) on the horizontal bar. The acrobot is a commonly used nonlinear control example.

The acrobot has a four-dimensional state-space, $x = [\theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2]$, where: θ_1 is the angle in radians of the first link (the "body") with respect to the vertical axis, θ_2 is the angle in radians of the second link (the "legs"), also with respect to the vertical axis, and $\dot{\theta}_1$, $\dot{\theta}_2$ are the angular velocities of the first and second link, respectively. The control action u is a torque applied to the middle joint.

The dynamics of the system are described as follows [17]:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \cdot \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

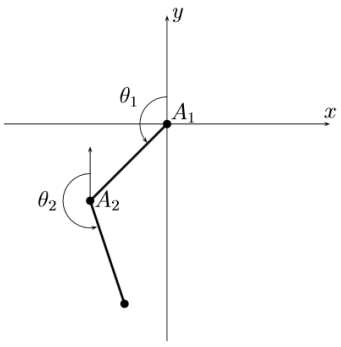

Fig. 3. Acrobot system. The figure is taken from [17].

where

$$a_{11} = (4/3 \cdot m_1 + 4m_2){l_1}^2, a_{22} = 4/3 \cdot m_2{l_2}^2$$
$$a_{11} = (4/3 \cdot m_1 + 4m_2){l_1}^2, a_{22} = 4/3 \cdot m_2{l_2}^2$$
$$a_{12} = a_{21} = 2m_2 l_1 l_2 \cos(\theta_1 - \theta_2)$$
$$b_1 = 2m_2 l_2 l_1 \dot{\theta}_2^2 \sin(\theta_2 - \theta_1) + (m_1 + 2m_2) l_1 g \sin(\theta_1) - \mu_1 \dot{\theta}_1 - u$$
$$b_2 = 2m_2 l_2 l_1 \dot{\theta}_1^2 \sin(\theta_1 - \theta_2) + m_2 l_2 g \sin(\theta_2) - \mu_2 \dot{\theta}_2 + u$$

Here, $l_1 = 0.5$ m is the half-length of the first link, $l_2 = 0.5$ m is the half-length of the second link, $m_1 = 1$ kg and $m_2 = 1$ kg are the mass of the first and second link, respectively, $g = 9.81$ m/s^2 is the gravitational acceleration and $\mu_1 = 0.05$ and $\mu_2 = 0.05$ are the friction coefficients of the links. The sampling time was set to $T_s = 0.1$ s. The action is discretized into a set of size $K = 3$ containing values $-2, 0, 2$.

The reward function used to express the goal of swinging up the acrobot is:

$$r_k = 1 - \frac{\sqrt{[y_k - (l_1 + l_2)]^2 + x_k^2}}{2(l_1 + l_2)}$$

with the cartesian coordinates of the "feet":

$$x_k = l_1 \sin(\theta_1) + l_2 \sin(\theta_2), \quad y_k = l_1 \cos(\theta_1) + l_2 \cos(\theta_2)$$

and discount factor $\gamma = 0.99$. This form was chosen in [18] to provide information about the progress towards the goal state of pointing up, while keeping rewards in the interval $[0, 1]$.

Theorem 2 with the case in equation (11) is used to find the feasible combinations of n and d', which allow for real-time application of RTOPS. The possible combinations are shown in Figure 4. Next we report simulation results in two scenarios: one where the model was identical to the system, and another where a model mismatch was introduced.

Perfect model: All feasible lengths for the applied subsequence have been used, in combination with a large budget n for each choice of d' (it might be possible to shrink the budget while maintaining performance, but our aim is to use all the allowable computation time). The results are summarized in Table I. The performance is generally good for short sequences and poor for long ones, however, this is not an exact relationship and has exceptions (e.g. $d' = 4$ leads to bad performance while $d' = 3$ and 5 do not). So for the acrobot, even when the model is accurate, applying long sequences is not favorable, likely because closing the loop sooner does help to react better to the evolution of the state.

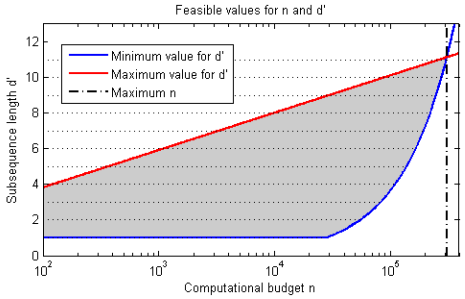

Fig. 4. The gray area shows all possible combinations of n and d' for which the worst-case condition for real-time applicability of RTOPS holds. Being integer, d' must lie on a dotted line in the gray area. Note that the horizontal axis is logarithmic.

TABLE I
DISCOUNTED SUM OF REWARDS (OVER 100 STEPS) FOR THE ACROBOT SIMULATION. THE COLUMNS ARE SORTED IN DESCENDING ORDER OF THE SUM, AND THE BEST PERFORMING SETTINGS ARE IN BOLD.

n	d'	Discounted sum of rewards
82950	**3**	**46.34372**
138300	**5**	**46.34372**
193600	7	46.30856
55300	2	46.26664
221200	8	46.25057
165900	6	46.14592
248900	9	45.28947
110600	4	40.41503
276600	10	40.22135
304200	11	39.12224

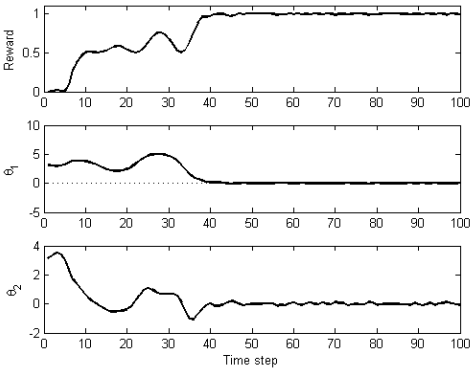

Fig. 5. Acrobot simulation results with $d' = 3$ and $n = 82950$.

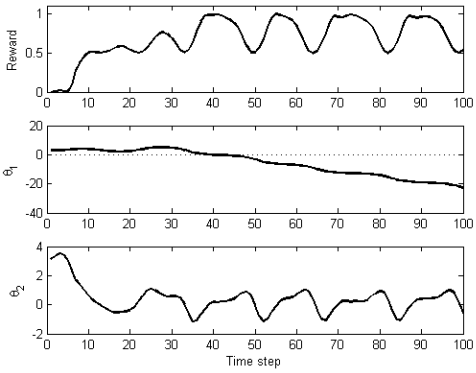

Fig. 6. Acrobot simulation results with $d' = 4$ and $n = 110600$.

As an illustration, the reward and angle trajectories in the experiments with the best performing setting ($d' = 3$) and the atypical poorly-performing short $d' = 4$ are shown in Figures

5 and 6, respectively. The swing-up succeeds for $d' = 3$ but fails for $d' = 4$, reflecting the numerical differences in return.

Model mismatch:

In practice, models will of course never be perfect, so we also study the performance of RTOPS when model errors are present. Several experiments for different kinds of parameter errors in the acrobot model have been conducted. The errors are introduced on the mass and length of the links. Four different settings are used for the parameters of the first link/second link: modeled too low/low, too low/high, too high/low, too high/high. For the first link "too low" means the modeled mass is $m_1 \cdot 0.95$ (the real mass is m_1) and $l_1 \cdot 0.99$, and "too high" means $m_1 \cdot 1.02$ and $l_1 \cdot 1.02$. For the second link "low" is $m_2 \cdot 0.97$ and $l_2 \cdot 0.6$ and "high" $m_2 \cdot 1.03$ and $l_2 \cdot 1.01$. Table II shows the results for $d' = 2$ and $d' = 3$. Interestingly, neither of these values of d' performs consistently better. The model mismatch seems to determine which of the settings outperforms the other (or if they give the same performance). We include trajectories for the model mismatch type for which the performance differs most between the two settings (namely "high/low"), in Figures 7 and 8. The performance with $d' = 2$ is not as good at first, but it is able to keep both links positioned upright after one "miss".

TABLE II
COMPARISON OF PERFORMANCE BETWEEN $d' = 2$ AND $d' = 3$ FOR DIFFERENT MODEL PARAMETER ERRORS. FOR EACH TYPE OF ERROR, THE BEST SETTING IS HIGHLIGHTED IN BOLD.

d' \ $(m_1, l_1/m_2, l_2)$	low/low	low/high	high/low	high/high
2 ($n = 55310$)	**48.2293**	46.2644	44.8735	**45.9687**
3 ($n = 82950$)	48.0277	**46.2731**	**47.4749**	**45.9687**

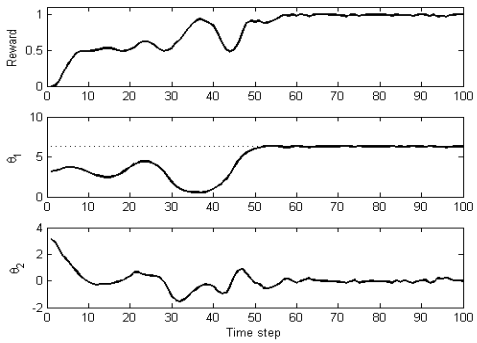

Fig. 7. Acrobot simulation results with $d' = 2$ and $n = 55310$, for "high/low" model mismatch.

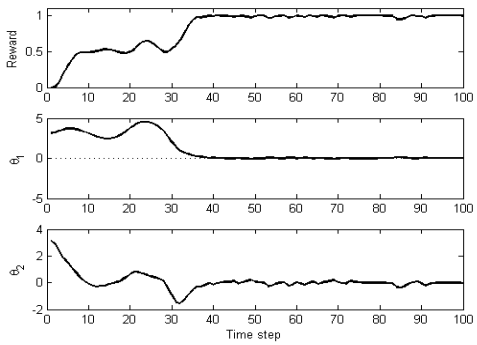

Fig. 8. Acrobot simulation results with $d' = 3$ and $n = 82950$, for the same mismatch.

B. Real inverted pendulum

A DC motor setup developed at TU Delft is used to create an inverted pendulum, by attaching an additional mass at the edge of the rotating disc. A picture of the setup is shown in Figure 9. The state is two-dimensional, $x = [\theta, \dot{\theta}]$, where θ is the angle in radians of the (virtual) pole with respect to the vertical axis, and $\dot{\theta}$ its angular velocity. Both θ and $\dot{\theta}$ are provided directly by the interface software. The control action is the armature voltage and we choose to limit it to a maximum of $u_{\max} = 0.9$ V. The action is discretized into $K = 3$ discrete values, $-u_{\max}, 0.0, u_{\max}$. The goal is to bring the weight to the top and then stabilize it there. Since the maximum voltage is small, several swings are required to accumulate energy before the pendulum can be pointed up.

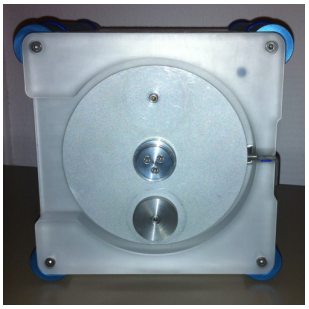

Fig. 9. Experimental setup: DC motor as inverted pendulum. The aim is to position the weight at the top, opposite to where it is located in the picture.

The dynamics of this inverted pendulum are:

$$\ddot{\theta} = 1/J(mgl\sin(\theta) - (b + \frac{K^2}{R})\dot{\theta} + K_m u) \qquad (17)$$

where $m = 0.03$ kg is the mass of the attached weight, $g = 9.81$ m/s^2 is the gravitational acceleration, $l = 0.042$ m is the distance between the center of the disc and the center of the weight, $b = 3.0 \cdot 10^{-6}$ Nms/rad is the damping coefficient, $K = 53.6 \cdot 01^{-3}$ Nm/A is the torque constant, $R = 9.50\ \Omega$ is the rotor resistance, $K_m = \frac{K}{R}$ and $J = 10 \cdot 10^{-5}$ kg·m^2 is the moment of inertia of the rotor. The sampling time in the experiments was set to $T_s = 0.05$ seconds. The numerical integration method used to simulate the model is Runge-Kutta 4^{th} order, which in preliminary tests gave results close to the actual system. The reward function is [18]:

$$r_k = 0.5(\cos(\theta) + 1.0) \qquad (18)$$

which again provides progress information towards the goal $\theta = 0$, while remaining in $[0, 1]$; and $\gamma = 0.99$.

When using RTOPS on a real system, real-time synchronization must be taken into account, as well as the fact that planning and applying actions are done concurrently. To this end, we run RTOPS and the I/O-interface with the system (including the actuation with the sequence of actions) in two separate processes that communicate through a UDP socket.

We choose the settings $n = 1666$ and $d' = 2$, which are feasible due to Theorem 2. Moreover, d' is small, which gave good results for the acrobot. Since this is a real-life system, a perfect model does not exist.

Our experiments show good performance: Figure 10 illustrates that RTOPS can indeed bring the weight to the upright

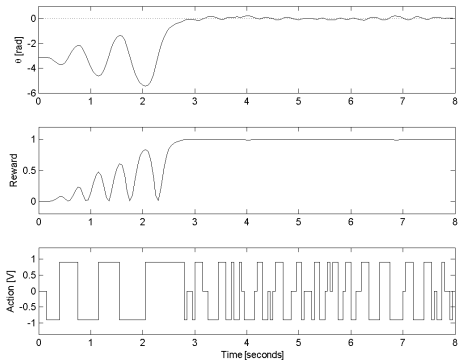

Fig. 10. The controlled trajectory produced by RTOPS on the real inverted pendulum system.

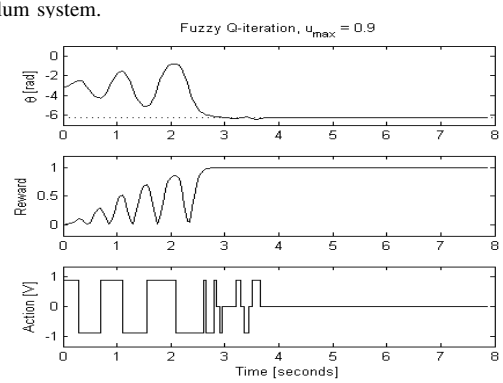

Fig. 11. Results with fuzzy QI policy; compare to Figure 10. The system trajectories are mirrored with respect to the horizontal axis, but this is not relevant since there is no preference between the two directions in the optimal solution.

position, through as much as five swings. Thus RTOPS is able to search over a long enough horizon to plan these five swings. We also verify how close the RTOPS solution is to the optimal one. Approximate value iteration with a very accurate approximator is used to find a provably near-optimal control policy [1], and this policy is then used to control the system. Note that value iteration is applicable due to the low dimensionality of the problem, and quickly becomes intractable for higher dimensions; in contrast, OPD is independent of the state dimensionality (although it is limited to a few action dimensions). The results with this near-optimal policy are shown in Figure 11, and they are close to those of RTOPS, except for the elimination of chattering around the equilibrium. Looking at the discounted sum of rewards obtained (truncated at 1200 steps, which induces an error of less than $\frac{\gamma^{1200}}{1-\gamma} < 0.0006$ compared to the infinite-horizon value), fuzzy Q-iteration obtained 68.8578, compared to RTOPS at 68.3578; the difference is small.

V. Conclusions and Future Work

We introduced an extension to real-time control of an optimistic planning algorithm for deterministic near-optimal control. Our approach applies sequences of actions in parallel with running OP to find the *next* sequence from the predicted state at the end of the current sequence. We provided conditions under which the algorithm is guaranteed to be feasible in real-time, and we analyzed its performance by extending results in [9]. Experiments were reported for two nonlinear systems: a simulation of an acrobot, and a real inverted pendulum.

The most important extension of this work is to provide performance guarantees while taking model mismatch into account. Further, the same idea of using long-horizon solutions to allow more time for computation should also work for other OP algorithms, such as OP for stochastic systems [6] or for continuous actions [8].

References

[1] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, 4th ed. Athena Scientific, 2012, vol. 2.

[2] 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.

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

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

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

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

[7] L. Buşoniu, E. Páll, and R. Munos, "An analysis of optimstic, best-first search for minimax sequential decision making," in *2014 IEEE International Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL-14)*, Orlando, 10–12 December 2014.

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

[9] L. Buşoniu, R. Postoyan, and J. Daafouz, "Near-optimal strategies for nonlinear networked control systems using optimistic planning," in *Proceedings American Control Conference 2013 (ACC-13)*, Washington, DC, 17–19 June 2013, pp. 3020–3025.

[10] 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.

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

[12] ——, "Intermittent model predictive control," *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, vol. 221, no. 7, pp. 1007–1018, 2007.

[13] 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.

[14] 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.

[15] H. van Ekeren, R. Negenborn, P. van Overloop, and B. De Schutter, "Time-instant optimization for hybrid model predictive control of the Rhine-Meuse delta." *Journal of Hydroinformatics*, vol. 15, no. 2, pp. 271–292, 2013.

[16] L. Buşoniu, A. Daniels, R. Munos, and R. Babuška, "Optimistic planning for continuous–action deterministic systems," in *2013 IEEE International Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL-13)*, Singapore, 16–19 April 2013.

[17] R. Coulom, "Reinforcement learning using neural networks, with applications to motor control," Ph.D. dissertation, Institut National Polytechnique de Grenoble, 2002.

[18] J.-F. Hren, "Planification optimiste pour systèmes déterministes," Ph.D. dissertation, Lille 1 University - Science and Technology, 2012.