Infinite horizon model predictive control for nonlinear periodic tasks

Article

CITATIONS
16

READS
387

3 authors, including:

Yuval Tassa
Google Inc.
42 PUBLICATIONS **5,781** CITATIONS

SEE PROFILE

Infinite-Horizon Model Predictive Control for Nonlinear Periodic Tasks

Tom Erez*,Yuval Tassa†, and Emanuel Todorov‡
*Computer Science and Engineering
Washington University in St. Louis, Missouri
etom@cse.wustl.edu
†Interdisciplinary Center for Neural Computation
Hebrew University, Jerusalem, Israel
tassa@alice.nc.huji.ac.il
‡Applied Mathematics and Computer Science & Engineering
University of Washington, Seattle, USA
todorov@cs.washington.edu

***Abstract*—We present a method for generating robust controllers for complex periodic behavior in nonlinear domains. We use an offline optimization technique to find the limit-cycle solution of an infinite-horizon average-cost optimal-control task. By minimizing the Bellman residual around this limit-cycle, we compute a local quadratic approximation of the Value function. We then use this Value approximation as the *terminal cost* of an online receding-horizon (Model Predictive Control) optimizer.**

This combination of an offline solution of the infinite-horizon problem with an online MPC controller is known as Infinite Horizon Model Predictive Control, and has previously been applied only to simple stabilization objectives. We demonstrate the power of our approach by synthesizing *hopping* behavior in a 10-dimensional simulated robot. Even for very short (and computationally cheap) MPC horizons, the combined controller is shown to recover from very large disturbances, and return the system to the optimal hopping gait. As further evidence of robustness, we introduce modeling errors by altering the morphology of the robot, and show that the same controller remains effective.

I. Introduction

Methods of optimal control derive control policies from cost functions that penalize undesirable states. This is an appealing paradigm, because it offers the system designer an intuitive scheme — it is easier to specify which states are desirable than to directly craft the policy that realizes these goals.

The *finite-horizon* criterion seeks to minimize the future cumulative cost over some predefined planning horizon. The repeated online solution of the finite-horizon problem for a perpetually receding horizon is called Model Predictive Control (MPC). Online optimization is possible because this class of problems is relatively easy to solve, but may result in undesirable "myopic" behavior.

The *infinite-horizon* criterion poses the optimization problem in terms of the average cost over an infinitely-distant horizon. This formulation is applicable to domains of gait generation, where the solution is periodic and forms a limit cycle. This class of optimization problems is usually too computationally-intensive to be solved online.

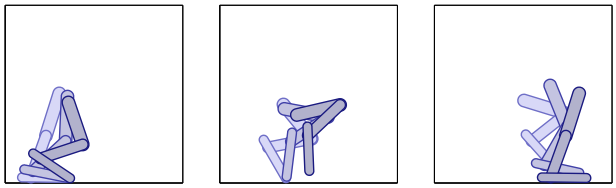

Fig. 1. The hopping robot's limit cycle.

The strengths of these two formulations can be combined by using the cost-to-go of an infinite-horizon solution as the terminal cost of the finite-horizon problem. This scheme is called Infinite-Horizon MPC (IHMPC), and in the past has been applied only to tracking problems (see section II-C).

In this paper, we tackle the more general problem where the optimal behavior results in a limit cycle. We use an offline optimization scheme to construct a locally-quadratic approximation of the infinite-horizon cost-to-go around the optimal limit-cycle. We then use this approximation as a terminal cost for online IHMPC.

This approach is shown to efficiently solve a domain of gait design and control for a 10-dimensional hopping robot (figure I). The limit cycle is found through offline optimization, and the infinite-horizon average-cost value function (section II-B) is fitted around the closed trajectory. This approximation is used for IHMPC, and yields robust behavior that can effectively recover from any perturbation, as illustrated by a movie demonstrating our results (which is available at goo.gl/jW9Rm). This computation is robust enough to yield competent hopping even when the planning horizon of MPC is very short, which allows efficient online computation using a standard laptop computer. Furthermore, the robustness afforded by MPC extends to effective recovery from modeling errors, as we were able to generate robust hopping even when the simulated plant is incongruent with the model used for planning (figure 5).

II. BACKGROUND

We consider systems with non-linear, discrete-time dynamics of the form:

$$\mathbf{x}(i+1) = \mathbf{f}(\mathbf{x}(i), \mathbf{u}(i)) \tag{1}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the system's state, and $\mathbf{u} \in \mathbb{R}^m$ is the control signal. The optimization criterion is specified by the cost function $\ell(\mathbf{x}, \mathbf{u})$.

Local algorithms of optimal control fall into two broad categories [1], according to the representation of the search space: *simultaneous* methods explicitly represent the trajectory in state space, treating the dynamics as constraints; in contrast, *sequential* methods represent only the control sequence, and use numerical integration to evaluate the resulting trajectory. The sequential approach often employs less variables, and the resulting trajectories are guaranteed to be dynamically consistent. The simultaneous approach allows the optimization to consider infeasible trajectories; this may prevent getting stuck in some local minima, but requires machinery for enforcing dynamical consistency.

A. Trajectory optimization and MPC

Given a planning horizon N, we seek the open-loop control sequence $\mathbf{U} = \{\mathbf{u}(1), \ldots, \mathbf{u}(N-1)\}$ that minimizes the cumulative cost:

$$V\big(\mathbf{x}(1), 1\big) = \min_{\mathbf{U}} \sum_{i=1}^{N-1} \ell\big(\mathbf{x}(i), \mathbf{u}(i)\big) + \ell_N\big(\mathbf{x}(N)\big) \tag{2}$$

where $\mathbf{x}(i)$ is defined by (1) for $i > 1$, and ℓ_N is an optional state-dependent cost function which is applied to the terminal state of the trajectory.

Bellman's equation for this case yields a *time-dependent* value function, defined recursively as the optimal cost-to-go:

$$V(\mathbf{x}, i) = \min_{\mathbf{u}} \left[\ell(\mathbf{x}, \mathbf{u}) + V\big(\mathbf{f}(\mathbf{x}, \mathbf{u}), i+1\big)\right]. \tag{3}$$

Since the value function is time-dependent, the effective planning horizon of states at the latter part (tail end) of the trajectory is very short, which may cause myopic behavior in these states. The terminal cost function ℓ_N mitigates that problem, since it can effectively inform the controller about all the events which lie beyond its planning horizon. However, crafting a terminal cost function that yields the intended result can be hard.

Finite-horizon optimization problems can be solved by both simultaneous and sequential approaches; we focus here on the sequential approach. Algorithms in this category (see section II-C) share a common structure: given a fixed first state, and initialized with some control sequence, every iteration seeks an improved control sequence. The basic structure of a single iteration consists of two phases: first, a new nominal trajectory is found by integrating the current control sequence forward in time using equation (1). Then, the trajectory is swept backwards, and a modification to the control sequence is computed. These two passes are repeated until no improvement is found.

The computation of the modified control sequence often involves the integration of the time-dependent value function along the nominal trajectory. In such cases, every step of the backwards pass computes a local approximation of the value function around a certain state by integrating backwards the local approximation around the next state, following equation (3). Often, a locally-quadratic model of the value function is used, parametrized by a linear component $V_{\mathbf{x}}(i)$ and a quadratic component $V_{\mathbf{xx}}(i)$. In such cases the backwards step has the form:

$$\big\{V_{\mathbf{x}}(i), V_{\mathbf{xx}}(i)\big\} = B\big(\mathbf{x}(i), V_{\mathbf{x}}(i+1), V_{\mathbf{xx}}(i+1)\big). \tag{4}$$

We include an explicit description for such an algorithm in the appendix, where the backwards step B is computed by equations (10)-(12). These algorithms are fast enough to be used for planning in Model-Predictive Control (MPC).

The method of MPC is designed to avoid the problem of myopic behavior: while an entire trajectory is planned, only the first action is executed, and planning starts again from the resulting new state. In order to be applicable for robotic domains, the process of re-planning must be very rapid. This is achieved by warm-starting the optimization with the entire optimized trajectory from the previous iteration, with a single time-shift. The effectiveness of MPC rests on the assumption that the solution at the previous timestep is similar to the solution at this timestep, and therefore the warm-start will allow for efficient optimization.

This assumption is disrupted if the tail of the trajectory falls into a local optimum. This will have no immediate effect, because only the first action is actually executed by the MPC, but as the planning horizon recedes, more and more of the trajectory is pushed into that local minimum. If at some stage the local optimality vanishes, the trajectory used to initialize the re-planning is no longer almost-optimal. This usually leads to the failure of MPC, since there is not enough time for re-optimization.

In section IV-B, we describe a one-legged hopping robot whose task is to maintain a fixed horizontal velocity. Intuitively, the optimal behavior we expect in such a domain is a hopping gait. However, applying regular MPC to this domain results in catastrophic failure. To understand why, consider a trajectory where the last few states could involve ground collision. Such a collision would necessarily slow down the hopper, which would impede the performance of the task in the short term. As such, in the absence of an adequately long planning horizon, the optimal solution is myopic, causing the hopper to retract its leg; by avoiding the ground, the hopper tries to maintain its air velocity just a few timesteps longer. However, as the planning horizon of the MPC recedes, the locally-optimal avoidance maneouver becomes more complicated, and eventually ground impact becomes inevitable. At that stage, the MPC must plan the foot landing, but the optimization is initialized with a suboptimal trajectory that involves a bizarre contortion towards the end. Unwinding this suboptimal behavior is probably impossible

within the time constraints of the robotic application, leading to the failure of MPC.

Such local minima can be avoided by using an appropriate terminal cost function ℓ_N. The central idea of this paper is to use the solution of the inifinite-horizon problem (section II-B) as the terminal cost for MPC. Section IV-B shows that this allows MPC to generate robust hopping through large external perturbations without falling into local minima.

B. Limit-Cycle Optimization

In this case, we seek a policy $\mathbf{u} = \pi(\mathbf{x})$ that minimizes the average cost along the trajectory in the limit of infinitely-long horizon:

$$c = \min_{\pi} \lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} \ell\Big(\mathbf{x}(i), \pi\big(\mathbf{x}(i)\big)\Big). \tag{5}$$

Given the optimal policy, the value function of the infinite-horizon problem is defined as the deviation of the future cumulative cost from the average:

$$\tilde{V}(\mathbf{x}(1)) = \lim_{N\to\infty} \left[\sum_{i=1}^{N} \ell\Big(\mathbf{x}(i), \pi\big(\mathbf{x}(i)\big)\Big) - Nc\right]. \tag{6}$$

Bellman's equation for this formulation is simpler, as it is no longer time dependent:

$$c + \tilde{V}(\mathbf{x}) = \min_{\mathbf{u}} \Big[\ell(\mathbf{x}, \mathbf{u}) + \tilde{V}\big(\mathbf{f}(\mathbf{x}, \mathbf{u})\big)\Big] \tag{7}$$

A sequential approach is inapplicable to solve infinite-horizon problems. Instead, we apply a simultaneous approach to the optimization of limit cycles, which takes the general form:

$$\min_{\mathbf{X},\mathbf{U}} \sum_{i=1}^{N} \ell\big(\mathbf{x}(i), \mathbf{u}(i)\big) \quad \text{s.t. } \forall i : \psi\big(\mathbf{x}(i), \mathbf{u}(i), \mathbf{x}(i+1)\big) = 0$$

where $\mathbf{X} = \{\mathbf{x}(i)\}_{i=1}^{N}$ is a sequence of states, $\mathbf{U} = \{\mathbf{u}(i)\}_{i=1}^{N}$ is a sequence of controls, and the function

$$\psi(\mathbf{x}, \mathbf{u}, \mathbf{x}') = \mathbf{x}' - \mathbf{f}(\mathbf{x}, \mathbf{u})$$

imposes dynamical consistency between every consecutive pair of states, treating equation (1) as a constraint. Since we focus on limit cycles, we define $N + 1 \equiv 1$ so that the last state $\mathbf{x}(N)$ is followed by the first state $\mathbf{x}(1)$.

The optimization algorithm searches in the space of all rings $\mathbf{X}$ for a limit cycle that minimizes the cumulative cost. In order to reduce the dimensionality of the search space, we build a model of the plant that solves for *inverse dynamics*: given a pair of states $\mathbf{x}, \mathbf{x}'$, we compute the control signal $\mathbf{u}$ that minimizes the dynamical error $||\psi(\mathbf{x}, \mathbf{u}, \mathbf{x}')||^2$. This allows us to maintain only a representation of the state-space trajectory $\mathbf{X}$, and have it implicitly define $\mathbf{U}$.

In practice, we can handle the constraints ψ by using a penalty method, yielding the unconstrained optimization problem

$$\min_{\mathbf{X}} \sum_{i=1}^{N} \Big[\ell\big(\mathbf{x}(i), \mathbf{u}(i)\big) + \lambda \big|\big|\psi\big(\mathbf{x}(i), \mathbf{u}(i), \mathbf{x}(i+1)\big)\big|\big|^2\Big]$$

which can be solved using standard optimization methods.

C. Related work

The basic algorithm for sequential trajectory optimization in a nonlinear system is Pontryiagin's minimum principle [2], a generalization of the Euler-Lagrange equations that provides a first-order criterion for identifying a locally-optimal trajectory. More elaborate proposals include algorithms by Jacobson and Mayne [3] (see appendix), Bryson and Ho [4] and Bertsekas and Dunn [5].

MPC has been studied extensively in the past few decades (see reviews by Morari and Lee [6], Bertsekas [7], Mayne et al. [8], Diehl et al. [1], and references therein) and its application to robotic domains is gaining popularity [9]. Chen and Allgöwer [10] seem to be the first to suggest the combination of MPC with an infinite-horizon optimization problem, and this approach has been studied extensively in the past decade [11]–[13]. However, current IHMPC algorithms make the strong assumption that the task is specified in terms of reaching a goal state. In such a case, only a finite amount of time is spent away from this target, and so the conditions around the goal state dominate the infinite-horizon considerations. Even if the domain exhibits non-linear dynamics, a linear approximation can be used effectively in some small region around the goal state. Together with a quadratic approximation of the cost function, the infinite-horizon problem takes the familiar form of a Linear-Quadratic Regulator (LQR), and can be solved using Ricatti equations.

The optimization criterion of infinite-horizon average-cost has been extensively studied in the past two decades [14]–[17]; it was used for policy search [18], and successfully applied to gait optimization [19].

Local methods of optimization that use a simultaneous representation include multiple shooting [20] and space-time constraints [21], and this approach has been applied to gait design [22], [23]. Popović and Wu [24] presented a method where offline optimization (through Evolutionary Computation) was complemented by Quadratic Programming during runtime to generate robust locomotion behavior from motion-capture data.

III. Approximating the Infinite-Horizon Value Function

Given an optimized closed trajectory $\mathbf{X}$ and the corresponding open-loop sequence $\mathbf{U}$, we approximate the value function of the infinite-horizon problem locally as a quadratic function. This means that for each of the points $\mathbf{x}(i) \in \mathbf{X}$ we seek the coefficients $\tilde{v}(i), \tilde{V}_{\mathbf{x}}(i), \tilde{V}_{\mathbf{xx}}(i)$ so that for small $\delta\mathbf{x}$,

$$\tilde{V}(\mathbf{x}(i) + \delta\mathbf{x}; i) \approx \tilde{v}(i) + \delta\mathbf{x}^{\mathsf{T}}\tilde{V}_{\mathbf{x}}(i) + \frac{1}{2}\delta\mathbf{x}^{\mathsf{T}}\tilde{V}_{\mathbf{xx}}(i)\delta\mathbf{x}.$$

The constant terms $\tilde{v}(i)$ can be computed directly, while the coefficients $\tilde{V}_{\mathbf{x}}(i), \tilde{V}_{\mathbf{xx}}(i)$ can be estimated using least-squares.

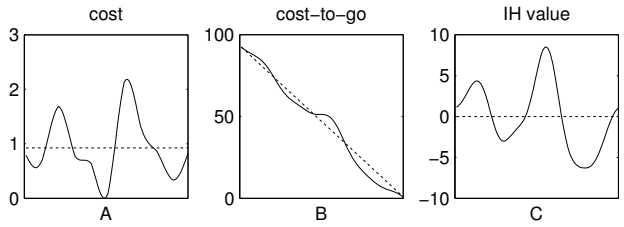

Fig. 2. An illustration of the calculation of the infinite-horizon value function along a limit cycle (described in section III-A). **A.** the cost ℓ at every point along a limit cycle (solid), and the average cost (dashed). **B.** the finite horizon cost-to-go (solid) and the average cost-to-go (dashed), integrated along one cycle of the closed trajectory. **C.** the resulting infinite-horizon average-cost value function, shifted so that its mean over the entire limit cycle is zero.

A. The Constant Terms $\tilde{v}$

Since we can measure the cost ℓ along every point of the limit cycle, we can calculate the average cost per step

$$c = \frac{1}{N}\sum_{i=1}^{N} \ell\big(\mathbf{x}(i), \mathbf{u}(i)\big).$$

We define the cost-to-go along the limit cycle

$$V(i) = \sum_{k=i}^{N} \ell\big(\mathbf{x}(i), \mathbf{u}(i)\big)$$

and the average cost-to-go

$$\bar{V}(i) = (N-i)c.$$

The infinite-horizon value function is the deviation of the cost-to-go from the average cost-to-go

$$\tilde{V}(i) = V(i) - \bar{V}(i) + b$$

(see figure 2), shifted by the scalar b to enforce

$$\sum_{i=1}^{N} \tilde{V}(i) = 0.$$

This computation is illustrated in figure 2.

B. Estimating the Quadratic Model

Equation (4) describes a relationship between two consecutive quadratic models of the time-dependent value function (3). Equation (7) describes a very similar relationship between two parts of the same value function around two temporally-consecutive states. However, although equations (3) and (7) differ only in the constant term, a straightforward backward integration of equation (4) along the limit cycle cannot always be used to evaluate the value function of the infinite-horizon problem. The backward integration used in the finite-horizon case relies on the dynamic-programming principle that the past can affect the future, but not vice-versa. This principle does not hold in limit cycles, where every state is visited both before and after every other state. We found that in practice, simple backward integration of equation (4) along several rounds of the limit cycle can be effective for some domains (section IV-A), but fails to produce robust behavior in others (section IV-B).

As an alternative, we can pose equation 4 in terms of least-squares minimization. Given some values of a quadratic model $\tilde{V}_{\mathbf{x}}(i), \tilde{V}_{\mathbf{xx}}(i)$ and its successor $\tilde{V}_{\mathbf{x}}(i), \tilde{V}_{\mathbf{xx}}(i)$, we can define the backward step mismatch as

$$\Psi(i) = \{\tilde{V}_{\mathbf{x}}(i), \tilde{V}_{\mathbf{xx}}(i)\} - B\big(\tilde{V}_{\mathbf{x}}(i+1), \tilde{V}_{\mathbf{xx}}(i+1)\big).$$

The difference between two quadratic models can be defined in several ways; here we take the simple approach of calculating the difference between corresponding coefficients. We seek a sequence of quadratic models that minimize the backward step mismatch:

$$\min_{\{\tilde{V}_{\mathbf{x}}(i), \tilde{V}_{\mathbf{xx}}(i)\}_{i=1}^{N}} ||\Psi(i)||^2.$$

This is a nonlinear sum-of-squares optimization problem whose variables are the coefficients of the quadratic models around the ring. Although the number of variables in this problem is quadratic in the dimensionality of the domain, note that only variables of consecutive pairs interact. Therefore, the Hessian of this optimization problem is sparse, and it can be efficiently solved using standard optimization algorithms.

IV. Results

We demonstrate the efficacy of these methods by presenting results from two domains. First, we analyse a nonlinear 2D model, which can be solved globally through discretization, and compare our approximations to the ground truth. We then apply our method to a simulated domain of a one-legged hopping robot, showing how this method can yield robust

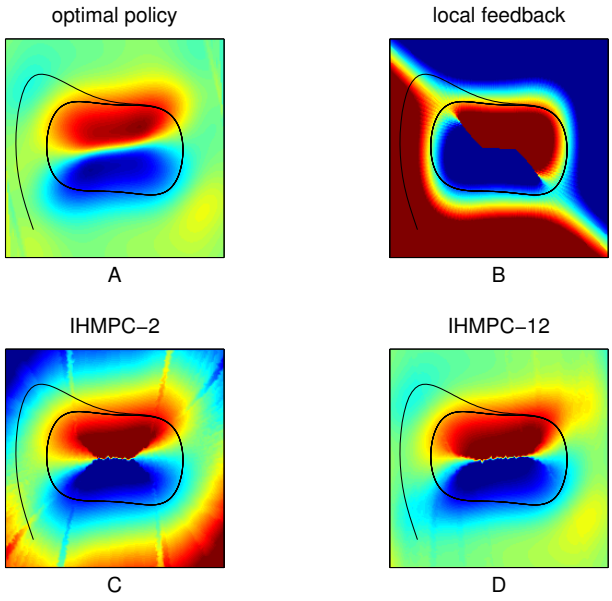

Fig. 3. **A.** the ground-truth policy solved through MDP discretization. **B.** the locally-linear feedback policy computed around the limit cycle. **C.** IHMPC with a lookahead horizon of 2 steps. **D.** IHMPC with a horizon of 12 steps. The complete limit cycle is 150 steps long.

body	length (cm)	radius (cm)	mass (kg)
foot	50	7	8.77
lower leg	50	6	6.33
upper leg	50	5	4.32

TABLE I
MORPHOLOGICAL SPECIFICATION OF THE HOPPING ROBOT

feedback behavior even in the context of discontinuities due to ground impact.

A. 2D Problem

A variant of this non-linear problem was first proposed by Todorov [25], and was further studied by Tassa et al. [26]. We discretized the state space with a 200x200 grid, and solved the resulting MDP that corresponds to the infinite-horizon problem to obtain ground-truth for the optimal policy.

We found the optimal limit cycle using the algorithm in section II-B, and confirmed that it matches the limit cycle found by the MDP. We identified quadratic approximations of the value function around every element of the limit cycle according to the algorithm presented in section III. We then used IHMPC to compute the policy over the entire state-space. Figure 3 shows the resulting policies for different lengths of the MDP horizon — with a planning horizon of 12 steps, the IHMPC policy becomes effectively equal to the true policy almost everywhere.

In practice, the 2D problem is simple, and the optimal limit cycle can be identified using finite-horizon optimization without any terminal cost, provided the horizon is long enough. This means that MPC alone could be used to solve the 2D problem, even with no terminal cost. However, such an approach cannot be applied to more complicated domains, like the one discussed in the next section.

B. Planar Hopping Robot

This mechanical system of the hopping robot is composed of three body segments and two joints. The masses and segment lengths are specified in table I. The ground interaction forces are computed using stochastic LCP [27] with $\sigma = 1$. The task requires the hopper's center of mass to maintain a fixed horizontal velocity $\dot{y}_{COM}$ of 1 m/s, while keeping its vertical position x_{COM} around 1 m:

$$\ell(\mathbf{x}, \mathbf{u}) = 10(\dot{y}_{COM} - 1)^2 + 0.1(x_{COM} - 1)^2 + 0.01\,||\mathbf{u}||^2$$

We started by finding an optimal limit cycle with $N = 40$ steps of 20msec. The optimization of the hopping gait was implemented in MATLAB, and took about an hour of computation on a standard dual-core T9300 Intel processor. IHMPC with a planning horizon of 10 steps was able to generate robust behavior over the entire state-space in real-time. The hopping behavior is best illustrated by a movie depicting the hooper in action, which is available online at `goo.gl/jW9Rm`. The effective basin of attraction of the IHMPC effectively covers the entire volume of state space, and the hopping robot can recover from any perturbation and resume its gait (for an illustration, see figure 4).

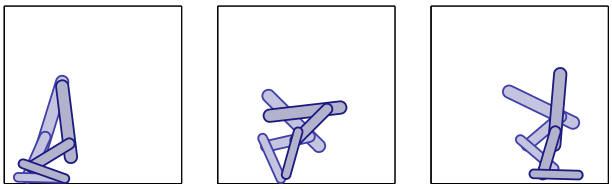

Fig. 5. The limit cycle of a hopping robot with altered morphology (the top body's length is extended by 60% to 80 cm). IHMPC can recover stable hopping even as the planner still uses the original model.

The robustness afforded by IHMPC also extends to model variations. To illustrate that, we modified the model by increasing the length of the top segment by 60% (figure 5). In this case, the original limit cycle is no longer the optimal gait for the modified morphology, and the MPC optimizations are using the wrong model of the dynamics. As the latter part of the movie shows, IHMPC can maintain effective hopping in this case as well.

ACKNOWLEDGMENT

This research was funded by the NSF.

APPENDIX

In the experiments described in section IV we used Differential Dynamic Programming (DDP), an algorithm by Jacobson and Mayne [3] which uses second-order expansion of both dynamics and cost to iteratively find improved trajectories given an initial state. Here we only repeat the main equations (as presented by [28]), and refer the interested reader to [3] for further details.

Every iteration involves a *backward pass* along the current $(\mathbf{x}, \mathbf{u}, i)$ trajectory, recursively constructing a quadratic approximation to $V(\mathbf{x}, i)$, followed by a *forward pass* which applies the new control sequence to form a new trajectory.

The backward pass integrates a quadratic approximation of the value function backward in time along the trajectory. Given a quadratic model around $\mathbf{x}(i+1)$, we compute the quadratic model $V_{\mathbf{x}}(i), V_{\mathbf{xx}}(i)$ around $\mathbf{x}(i)$ as a function of $V_{\mathbf{x}}(i+1), V_{\mathbf{xx}}(i+1)$ and the quadratic models of $\mathbf{f}, \ell$ around $\mathbf{x}(i)$.

This backward integration is initialized by taking a quadratic approximation of the terminal cost ℓ_N and setting the derivatives of the value function:

$$V_{\mathbf{x}}(N) = \ell_{N\mathbf{x}},\ V_{\mathbf{xx}}(N) = \ell_{N\mathbf{xx}}$$

where subscript denotes partial derivatives. In order to compute the integration step, we define the argument of the minimum in (3) (a quantity analogous to the Hamiltonian in continuous time) as a function of perturbations around the i-th $(\mathbf{x}, \mathbf{u})$ pair:

$$\begin{aligned} Q(\delta\mathbf{x}, \delta\mathbf{u}) &= \ell(\mathbf{x} + \delta\mathbf{x}, \mathbf{u} + \delta\mathbf{u}, i) - \ell(\mathbf{x}, \mathbf{u}, i) \\ &\quad + V(\mathbf{f}(\mathbf{x} + \delta\mathbf{x}, \mathbf{u} + \delta\mathbf{u}), i+1) - V(\mathbf{f}(\mathbf{x}, \mathbf{u}), i+1) \end{aligned} \quad (8)$$

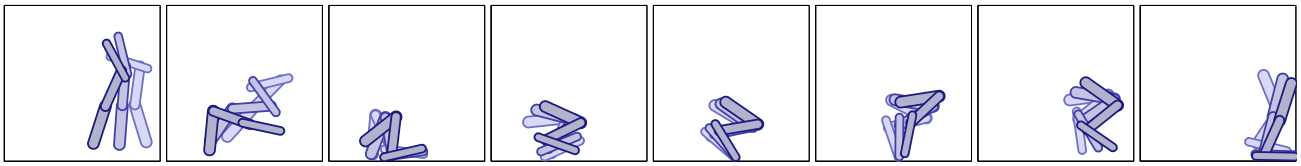

Fig. 4. IHMPC recovering from a starting state that is far from the limit cycle. Note how in the first frame on the right, the hopping robot is flying upside-down, and yet manages to eventually get a foothold and push itself back up.

and expand to second order

$$\approx \frac{1}{2}\begin{bmatrix}1\\ \delta\mathbf{x}\\ \delta\mathbf{u}\end{bmatrix}^{\mathsf{T}}\begin{bmatrix}0 & Q_{\mathbf{x}}^{\mathsf{T}} & Q_{\mathbf{u}}^{\mathsf{T}}\\ Q_{\mathbf{x}} & Q_{\mathbf{xx}} & Q_{\mathbf{xu}}\\ Q_{\mathbf{u}} & Q_{\mathbf{ux}} & Q_{\mathbf{uu}}\end{bmatrix}\begin{bmatrix}1\\ \delta\mathbf{x}\\ \delta\mathbf{u}\end{bmatrix}. \tag{9}$$

The expansion coefficients are[1]

$$Q_{\mathbf{x}} = \ell_{\mathbf{x}} \ + \mathbf{f}_{\mathbf{x}}^{\mathsf{T}} V'_{\mathbf{x}} \tag{10a}$$
$$Q_{\mathbf{u}} = \ell_{\mathbf{u}} \ + \mathbf{f}_{\mathbf{u}}^{\mathsf{T}} V'_{\mathbf{x}} \tag{10b}$$
$$Q_{\mathbf{xx}} = \ell_{\mathbf{xx}} + \mathbf{f}_{\mathbf{x}}^{\mathsf{T}} V'_{\mathbf{xx}} \mathbf{f}_{\mathbf{x}} + V'_{\mathbf{x}} \cdot \mathbf{f}_{\mathbf{xx}} \tag{10c}$$
$$Q_{\mathbf{uu}} = \ell_{\mathbf{uu}} + \mathbf{f}_{\mathbf{u}}^{\mathsf{T}} V'_{\mathbf{xx}} \mathbf{f}_{\mathbf{u}} + V'_{\mathbf{x}} \cdot \mathbf{f}_{\mathbf{uu}} \tag{10d}$$
$$Q_{\mathbf{ux}} = \ell_{\mathbf{ux}} + \mathbf{f}_{\mathbf{u}}^{\mathsf{T}} V'_{\mathbf{xx}} \mathbf{f}_{\mathbf{x}} + V'_{\mathbf{x}} \cdot \mathbf{f}_{\mathbf{ux}}. \tag{10e}$$

Note that the last terms in (10c, 10d, 10e) denote contraction with a tensor. Minimizing (9) WRT $\delta\mathbf{u}$ we have

$$\delta\mathbf{u}^*(i) = \underset{\delta\mathbf{u}}{\operatorname{argmin}}\, Q(\delta\mathbf{x}, \delta\mathbf{u}) = -Q_{\mathbf{uu}}^{-1}(Q_{\mathbf{u}} + Q_{\mathbf{ux}}\delta\mathbf{x}), \tag{11}$$

giving us an open-loop term $\mathbf{k} = -Q_{\mathbf{uu}}^{-1}Q_{\mathbf{u}}$ and a feedback gain term $\mathbf{K} = -Q_{\mathbf{uu}}^{-1}Q_{\mathbf{ux}}$. Plugging the result back in (9), we have a quadratic model of the Value at time i:

$$\Delta V(i) = \quad -\tfrac{1}{2}Q_{\mathbf{u}}Q_{\mathbf{uu}}^{-1}Q_{\mathbf{u}} \tag{12a}$$
$$V_{\mathbf{x}}(i) = Q_{\mathbf{x}} \ - Q_{\mathbf{u}}Q_{\mathbf{uu}}^{-1}Q_{\mathbf{ux}} \tag{12b}$$
$$V_{\mathbf{xx}}(i) = Q_{\mathbf{xx}} - Q_{\mathbf{xu}}Q_{\mathbf{uu}}^{-1}Q_{\mathbf{ux}}. \tag{12c}$$

Recursively computing the local quadratic models of $V(i)$ and the control modifications $\{\mathbf{k}(i), \mathbf{K}(i)\}$, constitutes the backward pass. The main complication stems from the inversion of $Q_{\mathbf{uu}}$ in (11): while the maximum principle guarantees it to be positive definite at the optimal trajectory, it is often not the case throughout the optimization. This can be solved by applying regularization: $\bar{Q}_{\mathbf{uu}} = Q_{\mathbf{uu}} + \lambda\mathbf{I}$.

Once the backward pass is completed, the *forward pass* computes a new trajectory and control sequence:

$$\hat{\mathbf{x}}(1) = \mathbf{x}(1) \tag{13a}$$
$$\hat{\mathbf{u}}(i) = \mathbf{u}(i) + \mathbf{k}(i) + \mathbf{K}(i)\big(\hat{\mathbf{x}}(i) - \mathbf{x}(i)\big) \tag{13b}$$
$$\hat{\mathbf{x}}(i{+}1) = \mathbf{f}(\hat{\mathbf{x}}(i), \hat{\mathbf{u}}(i)). \tag{13c}$$

Note that $\mathbf{K}(i)$ serves as a time-dependent linear feedback term, using the second-order information to ensure a better forward pass

[1]Dropping the index i, primes denoting the next time-step: $V' \equiv V(i{+}1)$.

REFERENCES

[1] M. Diehl, H. Ferreau, and N. Haverbeke, "Efficient numerical methods for nonlinear MPC and moving horizon estimation," *Nonlinear Model Predictive Control*, pp. 391–417, 2009.

[2] L. S. Pontryagin, V. G. Boltyanskii, and R. V. Gamkrelidze, "The Theory of Optimal Processes," vol. 110, no. 7, 1956.

[3] D. H. Jacobson and D. Q. Mayne, *Differential Dynamic Programming*. Elsevier, 1970.

[4] A. E. Bryson and Y. C. Ho, *Applied optimal control*. Wiley New York, 1975.

[5] J. C. Dunn and D. P. Bertsekas, "Efficient dynamic programming implementations of Newton's method for unconstrained optimal control problems," *Journal of Optimization Theory and Applications*, vol. 63, no. 1, pp. 23–38, 1989.

[6] M. Morari and J. H. Lee, "Model predictive control: past, present and future," *Computers & Chemical Engineering*, vol. 23, no. 4-5, pp. 667–682, 1999.

[7] D. P. Bertsekas, "Dynamic programming and suboptimal control: A survey from ADP to MPC," *European Journal of Control*, vol. 11, no. 4-5, pp. 310–334, 2005.

[8] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. Scokaert, "Constrained model predictive control: Stability and optimality," *Automatica*, vol. 36, pp. 789–814, 2000.

[9] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng, "An application of reinforcement learning to aerobatic helicopter flight," in *Advances in Neural Information Processing Systems 19: Proceedings of the 2006 Conference*, 2007, p. 1.

[10] H. Chen and F. Allgwer, "A Quasi-Infinite horizon nonlinear model predictive control scheme with guaranteed stability," *Automatica*, vol. 34, no. 10, pp. 1205–1217, 1998.

[11] G. Pannocchia, S. J. Wright, and J. B. Rawlings, "Existence and computation of infinite horizon model predictive control with active steady-state input constraints," *IEEE Transactions on Automatic Control*, vol. 48, no. 6, pp. 1002–1006, 2003.

[12] F. A. Almeida, "Waypoint navigation using constrained infinite horizon model predictive control," 2008.

[13] B. Hu and A. Linnemann, "Toward infinite-horizon optimality in nonlinear model predictive control," *IEEE Transactions on Automatic Control*, vol. 47, no. 4, pp. 679–682, 2002.

[14] A. Schwartz, "A reinforcement learning method for maximizing undiscounted rewards," in *Proceedings of the Tenth International Conference on Machine Learning*, vol. 298, 1993, p. 305.

[15] A. Gosavi, "A reinforcement learning algorithm based on policy iteration for average reward: Empirical results with yield management and convergence analysis," *Machine Learning*, vol. 55, no. 1, pp. 5–29, 2004.

[16] S. Mahadevan, "Average reward reinforcement learning: Foundations, algorithms, and empirical results," *Machine Learning*, vol. 22, pp. 159–196, 1996.

[17] S. P. Singh, "Reinforcement learning algorithms for average-payoff Markovian decision processes," in *Proceedings of the twelfth national conference on Artificial intelligence*, 1994, pp. 700–705.

[18] J. Baxter and P. L. Bartlett, "Infinite-horizon policy-gradient estimation," *Journal of Artificial Intelligence Research*, vol. 15, no. 4, pp. 319–350, 2001.

[19] R. Tedrake, T. W. Zhang, and H. S. Seung, "Stochastic policy gradient reinforcement learning on a simple 3D biped," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 3, 2004, pp. 2849–2854.

[20] H. G. Bock and K. J. Plitt, "A multiple shooting algorithm for direct solution of optimal control problems," *Proceedings of the 9th IFAC world congress*, 1984.

[21] A. Witkin and M. Kass, "Spacetime constraints," in *Proceedings of the 15th annual conference on Computer graphics and interactive techniques - SIGGRAPH '88*, 1988, pp. 159–168.
[22] K. Wampler and Z. Popović, "Optimal gait and form for animal locomotion," *ACM Transactions on Graphics (TOG)*, vol. 28, no. 3, pp. 1–8, 2009.
[23] K. Mombaur, "Using optimization to create self-stable human-like running," *Robotica*, vol. 27, no. 03, pp. 321–330, 2009.
[24] J.-c. Wu and Z. Popović, "Terrain-adaptive bipedal locomotion control," *ACM Trans. Graph.*, vol. 29, pp. 72:1–72:10, July 2010.
[25] E. Todorov, "Eigenfunction approximation methods for linearly-solvable optimal control problems," in *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*. IEEE, 2009, pp. 161–168.
[26] Y. Tassa, T. Erez, and E. Todorov, "Optimal limit-cycle control recast as bayesian inference."
[27] Y. Tassa and E. Todorov, "Stochastic complementarity for local control of discontinuous dynamics," in *Proceedings of Robotics: Science and Systems (RSS)*, 2010.
[28] Y. Tassa, T. Erez, and E. Todorov, "Fast model predictive control for complex robotic behavior."