Regularized Hierarchical Differential Dynamic Programming

Mathieu Geisert, Andrea del Prete, Nicolas Mansard, Francesco Romano, Francesco Nori

Regularized Hierarchical Differential Dynamic Programming

Mathieu Geisert, Andrea Del Prete, Nicolas Mansard, Francesco Romano, and Francesco Nori

***Abstract*—This paper presents a new algorithm for optimal control (OC) of nonlinear dynamical systems. The main feature of this algorithm is that it allows the specification of the control objectives as a hierarchy of tasks. Each task is described by a cost function that the algorithm tries to minimize, while not affecting the tasks of higher priority. The concept of strict priority allows for an easier and more robust specification of the control objectives, without hand-tuning of task weights. The hierarchy also makes it possible to properly regularize the behavior of each task independently. For the first time, we properly define the problem of regularizing the task cost functions in the presence of a hierarchy and propose an algorithm to compute an approximate solution. Several simulated scenarios with different robots compare our solution with other state-of-the-art methods, validating the interest of the hierarchy in OC and empirically demonstrating the importance of regularization to generate safe behaviors.**

***Index Terms*—Optimal control, Hierarchy of Tasks.**

I. Introduction

This paper deals with the problem of motion generation for nonlinear, possibly underactuated, dynamical systems, such as mobile robot manipulators, quadrotors and legged robots. A well-known and effective mathematical tool to tackle this problem is optimal control. Optimal control algorithms allow the user to generate motion for arbitrary systems by specifying a cost function to minimize. While solving an optimal control problem is intractable in most realistic cases, algorithms exist that compute approximate local optima, which are often good enough in practice [1], [2]. However, designing a cost function that results in the desired behavior is much more complex than it may look like. Especially for systems with many degrees of freedom (DoFs) a cost function is typically a weighted sum of several elementary costs, each one representing a task that the robot should perform [3], [4]. For instance, to make a humanoid robot walk, we can control the trajectory of its center of mass and its swinging foot, its angular momentum and its whole-body posture. If we also add manipulation objectives, it is clear that the number of tasks rapidly grows. In such cases, finding the right weights for the different terms of the cost function may be extremely time consuming. Moreover, weights are typically not robust to task variations, e.g. the weights used for walking may be very different from the weights needed for running. Often, rather than using weights, it is simpler to specify strict priorities between tasks. This means that in case of conflict between two tasks, we might require the most important task to be achieved at the expenses of the other. For instance, the task of avoiding collisions has clearly higher priority than the task of minimizing the motor commands. The concept of strict priorities is indeed widespread in robotics for inverse-kinematics [5], [6] and inverse-dynamics [7], [8] controllers. In optimal control strict priorities are typically approximated using much larger weights for the high-priority tasks. However this approach does not scale well when the number of priority levels grows because it can lead to poor numerical conditioning. This motivated our first work on Hierarchical Optimal Control (HOC) [9], in which we introduced strict priorities in the optimal control problem formulation. Later [10], we proposed another algorithm to solve the HOC problem, Hierarchical Differential Dynamic Programming (HDDP). In this paper we propose an improved version of HDDP, which properly handles the regularization of the tasks. While the problem of regularizing the cost function is often overlooked in the literature, we show that it is actually paramount when using strict priorities.

A. The Role of Regularization

The problem of regularization (or damping) is well-known in robotics [11], [12] and optimization [13], but for different reasons. Optimization problems are regularized to avoid poor numerical conditioning, which leads to large numerical errors due to the finite precision of computer's arithmetic. In this context, the regularization parameters typically take very small values (e.g. 10^{-9}). This regularization, which we refer to as *algorithm regularization*, modifies the original problem, thus introducing a small error in the resulting solution. This error is however largely compensated for by the improved behavior of the numerical solver.

In inverse-dynamics/inverse-kinematics control instead, regularization is used to prevent large motor commands, which occur e.g. in the neighborhood of kinematic singularities [11]. In this context, regularization parameters take much larger values (e.g. 10^{-3}). This regularization, which we refer to as *task regularization*, is a core part of the cost functions describing the tasks—rather than a parameter of the algorithm. Task regularization is even more critical in optimal control: without regularization the resulting control would overexploit the robot actuation capabilities (e.g. saturating the motor limits). This behavior is undesirable in most situations in autonomous robots.

The difference between task regularization and algorithm regularization is part of the know-how, but yet not well defined

Geisert, Del Prete and Mansard are with the CNRS, LAAS, 7 avenue du colonel Roche, Univ de Toulouse, LAAS, F-31400 Toulouse, France. e-mail: mgeisert@laas.fr, adelpret@laas.fr, nmansard@laas.fr.

Romano and Nori are with the iCubFacility department, Istituto Italiano di Tecnologia, Genoa, Italy. e-mail: francesco.romano@iit.it, francesco.nori@iit.it.

Manuscript submitted on Aug 4, 2016.

in robotics. This is because in inverse-kinematics/dynamics they are both implemented through the damping factor of the pseudo-inverses [11].

Despite its importance, the problem of task regularization is often not explicitely mentioned, or relegated to a small paragraph towards the end of the paper. This is because typically task regularization does not affect the mathematical developments that are the subject of the publication. However, this is no longer the case when considering a hierarchical (or lexicographic) optimization [14], [15]. Indeed we will show that the hierarchical minimization of least-squares functions—which has been extensively studied in robotics [13], [16] and is at the core of our algorithm—becomes nonconvex when introducing task regularization. Moreover, while the unregularized hierarchical problem can be defined as the limit of the weighted problem for the ratio of the weights going to infinity, this is no longer the case if we introduce regularization. This implies that the solution of the regularized hierarchical problem cannot be approximated with a classic optimization using large weights. These two properties highlight the fact that task regularization should be taken into account from the very beginning when dealing with hierarchical optimization. The contribution of this paper goes thus in this direction, by introducing a regularized version of HDDP, called Regularized HDDP (RHDDP).

B. State of the Art

To the best of our knowledge, there is only another work in the literature that deals with the problem of hierarchical trajectory optimization [14]. With respect to our approach we can find several differences. The main advantage of their algorithm [14] is that it can handle inequality constraints. However, this prevented them from exploiting the sparsity of the optimal control problem, which we do thanks to the Differential Dynamic Programming (DDP) formulation. Moreover, they did not deal with the problem of task regularization, which is the main focus of this paper.

This work is based on two previous conference publications [9], [10]. In our first work [9] we introduced strict task prioritization in the optimal control formulation. This algorithm did not exploit the intrinsic sparsity of the optimal control problem, which can lead to a reduction of the computational complexity from cubic to linear in the number of time steps. In our second work [10] we addressed this problem by extending the DDP algorithm to account for strict priorities between the cost functions. In this paper we present an improved version of the HDDP algorithm, together with extensive simulations with several robotic systems to validate our approach.

C. Paper Overview

This paper has three contributions. First, we propose the first well-founded definition and resolution of a hierarchy of quadratic objectives with regularization. Second, we use this solution to derive the first algorithm to solve HOC with regularization, while exploiting the sparsity of the problem. Finally, we demonstrate the importance of our approach by several case studies on various simulated robot models.

Before discussing about HOC, Section II treats the problem of Hierarchical Quadratic Programming (HQP), which is strongly related to HDDP. The subject of Hierarchical Least-Squares Programming (HLSP, a special case of HQP) is well-known in robotics and has been extensively studied [5], [16], [17] and applied [18] in recent years. However, the problem of task regularization has never been properly addressed. In particular, we show that the regularized HLSP problem is not convex in general—while HLSP always is. We then propose a convex relaxation of regularized HQP, which gives a (possibly) suboptimal solution that is guaranteed to satisfy the priority constraints.

Section III introduces the problem of Parametric HQP (PHQP), which consists in minimizing a set of quadratic cost functions with respect to (w.r.t.) a subset of their variables, treating the other variables as problem parameters. This is exactly what happens in the DDP algorithm, where the optimization is performed w.r.t. the control variables, while the state variables are treated as problem parameters. This allows DDP to compute feedback control laws rather than open-loop control trajectories.

Then, Section IV and V present the Regularized HDDP algorithm, exploiting the results already presented for PHQP. Section VI reports numerical simulations on different robotic systems, comparing RHDDP with HDDP and DDP. Finally, Section VII discusses the results and Section VIII draws the conclusions.

D. Notation

The following notation is used throughout the paper:

- (A, B) is a short form for $\begin{bmatrix} A^\top & B^\top \end{bmatrix}^\top$.
- $\mathcal{N}(A)$ is the null-space projector of the matrix A.
- $A^\dagger$ is the Moore-Penrose pseudo-inverse of the matrix A.
- $\partial_y g$ is the partial derivative of a multivariable function $g(\cdot)$ with respect to one of its variables y; $\partial_{yz} g$ is the partial second-order derivative with respect to y and z.
- y is a generic variable while x and u are respectively the state and control variable in an optimal control problem. We also denote by X and U the state and control sequences (i.e. $X = (x_0 \dots x_N)$)

II. Hierarchical Quadratic Programming (HQP)

We first recall the Hierarchical Quadratic Programming problem without regularization, as it is typically presented in the literature [16]. For applications in robot control, most of the time this problem is not interesting because it results in large/discontinuous motor commands. For this reason, we present then the regularized HQP problem, which does not suffer from this issue, and we discuss its properties.

A. Problem Statement

Suppose to have n_l quadratic functions:

$$g^{(l)}(y) = \frac{1}{2} y^\top H^{(l)} y + h^{(l)\top} y, \quad l = 1, \dots, n_l$$

which we want to minimize in a hierarchical way:

$$\begin{aligned} g^{(l)*} = \underset{y}{\text{minimize}} \quad & g^{(l)}(y) \\ \text{subject to} \quad & g^{(j)}(y) = g^{(j)*} \quad \forall j < l \end{aligned} \tag{1}$$

Clearly, we can expect the Hessians of all the functions $g^{(j)}(y)$ to be singular—except for the last one. If that was not the case, all the functions of priority lower than a function with a full-rank Hessian could not be optimized at all. In this form, the priority constraints are quadratic, which make problem (1) nonconvex. However, it is well-known that, for the least-squares case, we can replace them with linear constraints (as we will recall in Section II-C), making (1) convex. We will show that this is no longer the case if we introduce regularization.

B. Regularizing the Problem

Suppose to have a regularized version of each objective function:

$$\hat{g}^{(j)}(y) = \frac{1}{2} y^\top \hat{H}^{(j)} y + \hat{h}^{(j)\top} y, \quad j = 1, \dots, n_l$$

We assume that the regularized Hessians are positive-definite. A typical case of regularization consists in adding a scaled identify matrix to the Hessians, i.e. $\hat{H}^{(j)} = H^{(j)} + \lambda I$, while leaving the gradients unvaried, i.e. $\hat{h}^{(j)} = h^{(j)}$. The regularized HQP problem is then:

$$\begin{aligned} \hat{y}^{(l)*} = \underset{y}{\text{argmin}} \quad & \hat{g}^{(l)}(y) \\ \text{subject to} \quad & g^{(j)}(y) = g^{(j)}(\hat{y}^{(j)*}) \quad \forall j < l \end{aligned} \tag{2}$$

Note that we do not use the regularized functions in the priority constraints because that would leave no null space to optimize the secondary objectives. Again, because of the quadratic equality constraints, problem (2) is not convex. We show now how to replace them with linear constraints that are sufficient (but not necessary) to guarantee the satisfaction of the original quadratic constraints, resulting thus in a convex relaxation of (2).

C. Reformulating the Priority Constraints

Let us use the first two objectives of the hierarchy to illustrate this technique, which can be then easily generalized to the following objectives. The minimization of the first objective is unconstrained, so we can compute $\hat{y}^{(1)*}$ as:

$$\hat{y}^{(1)*} = -(\hat{H}^{(1)})^{-1} \hat{h}^{(1)}$$

The priority constraint for the optimization of the second objective is then:

$$\frac{1}{2} y^\top H^{(1)} y + h^{(1)\top} y = g^{(1)}(\hat{y}^{(1)*})$$

We introduce now a simple change of variable to simplify the derivation: $y = \hat{y}^{(1)*} + y^{(2)}$. This leads us to:

$$\frac{1}{2} y^{(2)\top} H^{(1)} y^{(2)} + \hat{y}^{(1)*\top} H^{(1)} y^{(2)} + h^{(1)\top} y^{(2)} = 0 \tag{3}$$

When the HQP problem is unregularized and the functions $g^{(l)}(y)$ are least-squares functions we can replace (3) with an equivalent linear constraint, which makes the HQP convex:

$$\frac{1}{2} y^{(2)\top} H^{(1)} y^{(2)} = 0 \quad \Longleftrightarrow \quad y^{(2)} = \mathcal{N}(H^{(1)}) z,$$

where z is an arbitrary variable, and we exploited the fact that for an unregularized HQP $\hat{y}^{(1)} = -(H^{(1)})^\dagger h^{(1)}$ and that for least-squares functions $h^{(1)\top} \mathcal{N}(H^{(1)}) = 0$. However, this is not the case for the regularized HQP, which is in general nonconvex. We suggest to replace the quadratic priority constraints (3) with linear constraints that are sufficient but not necessary to ensure (3). By doing so we get a convex optimization problem whose solution is in general suboptimal for the original problem, but it is guaranteed to satisfy the priority constraints. A sufficient condition for (3) to hold is to select $y^{(2)}$ in the null space of $H^{(1)}$ (to nullify the first two terms) and $h^{(1)\top}$ (to nullify the last term), that is:

$$y^{(2)} = \mathcal{N}((H^{(1)}, h^{(1)\top})) z = N^{(1)} z, \tag{4}$$

Note that for a hierarchy of least-squares functions being in the null space of $H^{(1)}$ would be sufficient [16] (but still not necessary) because the gradient would always be zero in the null space of the Hessian, i.e. $h^{(1)\top} \mathcal{N}(H^{(1)}) = 0$.

D. Solving the Second Minimization

The minimization of the second objective is then a QP:

$$\begin{aligned} \underset{y,z}{\text{minimize}} \quad & \hat{g}^{(2)}(y) \\ \text{subject to} \quad & y = \hat{y}^{(1)*} + N^{(1)} z \end{aligned}$$

Eliminating the constraints and setting the gradient of the cost to zero we get:

$$\hat{y}^{(2)*} = \hat{y}^{(1)*} - (N^{(1)} \hat{H}^{(2)} N^{(1)})^\dagger (\hat{h}^{(2)} + \hat{H}^{(2)} \hat{y}^{(1)*})$$

E. Solving the Whole Hierarchy

Generalizing this to an arbitrary number of functions n_l, we get the following recursive solution:

$$\hat{y}^{(l)*} = \hat{y}^{(l-1)*} - \bar{H}^{(l)\dagger} \bar{h}^{(l)},$$

where:

$$\begin{aligned} \bar{H}^{(l)} &\triangleq N^{(l-1)} \hat{H}^{(l)} N^{(l-1)} \\ \bar{h}^{(l)} &\triangleq \hat{h}^{(l)} + \hat{H}^{(l)} \hat{y}^{(l-1)*} \\ N^{(l)} &\triangleq N^{(l-1)} \mathcal{N}((H^{(l)}, h^{(l)\top}) N^{(l-1)}), \end{aligned}$$

The recursion is initialized with $N^{(0)} = I$ and $\hat{y}^{(0)*} = 0$.

F. A Simple Example

We can look at a simple example to give some insights about the proposed convex relaxation of the regularized HQP problem. Let us minimize in a hierarchical way two quadratic functions $g^{(1)}$ and $g^{(2)}$ of the 3-dimensional variable $y = (y_1, y_2, y_3)$. The function $g^{(1)}$ depends only on y_1 and y_2, and its Hessian has rank 2. The function $g^{(2)}$ instead has a full-rank Hessian. The regularized versions of $g^{(1)}$ and $g^{(2)}$ differ

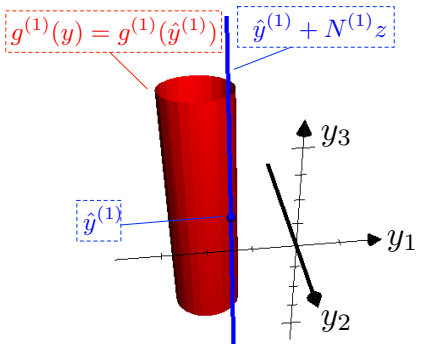


Fig. 1. 3D example depicting the difference between the nonconvex priority constraints of the regularized HQP problem (red cylinder) and their convex relaxation (blue line) proposed in this paper.

only for their Hessians, which are regularized in this way: $\hat{H}^{(l)} = H^{(l)} + \lambda I$.

In this case, the set of solutions of the priority constraint for the second level (i.e. $g^{(1)}(y) = g^{(1)}(\hat{y}^{(1)*})$) is described by the surface of a 3d cylinder (see Fig. 1). This cylinder has an axis that is parallel to y_3, and passes through the minimizer of $g^{(1)}$. The diameter of the cylinder is proportional to the regularization parameter λ. For $\lambda = 0$ the cylinder collapses to a line, and hence the priority constraint becomes linear, making the HQP convex. For $\lambda > 0$ the problem is not convex because the constraint of lying over the surface of a cylinder is clearly nonlinear. Rather than looking for the minimizer of $\hat{g}^{(2)}$ over the surface of this cylinder, our convex relaxation looks only over a line, which is parallel to y_3 and passes through $\hat{y}^{(1)}$.

In general, it is difficult to quantify the suboptimality of the resulting solution. Depending on the problem data we could find the global optimum, or we may be significantly suboptimal. Another approach to solve the regularized HQP could be to linearize the quadratic priority constraints and use Sequential Quadratic Programming [19]. However, the linear approximation of the quadratic constraints is in general rather poor, and leads the algorithm to take very small steps, slowing down convergence[1]. Alternatively, we could consider the interior of the nonconvex set defined by the priority constraints (which is a convex set) and use an Interior-Point method [19]. However, even this approach would require solving several QPs for each level of the hierarchy. Our convex relaxation instead allows us to compute an approximate solution by solving a single QP for each hierarchy level.

III. Parametric Hierarchical Quadratic Programming (PHQP)

The problem of HQP becomes more complex if we are not optimizing w.r.t. all the decision variables, but only w.r.t. a subset of them (as it happens in DDP). Let us split the decision variables into two subsets $y = (x, u)$:

$$g^{(l)}(x,u) = \frac{1}{2}x^\top H^{(l)}_{xx}x + \frac{1}{2}u^\top H^{(l)}_{uu}u + x^\top H^{(l)}_{xu}u + h^{(l)\top}_x x + h^{(l)\top}_u u, \qquad j = 1,\dots,n_l$$

We also have a regularized version of each cost function:

$$\hat{g}^{(l)}(x,u) = \frac{1}{2}x^\top \hat{H}^{(l)}_{xx}x + \frac{1}{2}u^\top \hat{H}^{(l)}_{uu}u + x^\top \hat{H}^{(l)}_{xu}u + \hat{h}^{(l)\top}_x x + \hat{h}^{(l)\top}_u u, \qquad j = 1,\dots,n_l$$

Rather than optimizing w.r.t. y, we want to optimize w.r.t. u only, treating x as a problem parameter:

$$\begin{aligned}\hat{u}^{(l)*}(x) = \underset{u}{\text{argmin}} \quad & \hat{g}^{(l)}(x,u) \\ \text{subject to } & g^{(j)}(x,u) = g^{(j)}(x,\hat{u}^{(j)*}) \quad \forall j < l\end{aligned} \tag{5}$$

1) Solving the First Minimization: For the first objective we have an unconstrained optimization, which we can solve by computing the cost gradient and setting it equal to zero:

$$\hat{u}^{(1)*} = \underbrace{-(\hat{H}^{(1)}_{uu})^{-1}\hat{h}^{(1)}_u}_{\hat{k}^{(1)}} - \underbrace{(\hat{H}^{(1)}_{uu})^{-1}\hat{H}^{(1)}_{ux}}_{\hat{K}^{(1)}}x$$

2) Solving the Second Minimization: The priority constraint for the optimization of the second objective is then:

$$g^{(1)}(x,u) = g^{(1)}(x,\hat{u}^{(1)*})$$

As before, we introduce a change of variable to simplify the derivation: $u = \hat{u}^{(1)*} + u^{(2)}$. This leads us to:

$$\frac{1}{2}u^{(2)\top}H^{(1)}_{uu}u^{(2)} + (\hat{u}^{(1)*\top}H^{(1)}_{uu} + x^\top H^{(1)}_{xu} + h^{(1)\top}_u)u^{(2)} = 0 \tag{6}$$

Similarly to HQP, if the problem were unregularized and the cost functions were least-squares, we could replace this quadratic constraint with an equivalent linear constraint (as we did in [10]). Contrary to the regularized HQP, selecting $u^{(2)}$ in the null space of $H^{(1)}_{uu}$ and $h^{(1)\top}_u$ is not sufficient to ensure the satisfaction of (6), because of the term $x^\top H^{(1)}_{xu}u^{(2)}$. A sufficient condition is to select $u^{(2)}$ to be also in the null space of $H^{(1)}_{xu}$:

$$u^{(2)} = \mathcal{N}((H^{(1)}_{uu}, h^{(1)\top}_u, H^{(1)}_{xu}))z = N^{(1)}z \tag{7}$$

This new constraint is linear and it is a sufficient condition for the original quadratic constraint. We propose to relax problem (5) by replacing (6) with (7). The solution of the second objective minimization is then:

$$\hat{u}^{(2)*} = -(\bar{H}^{(2)}_{uu})^\dagger \bar{h}^{(2)}_u - (\bar{H}^{(2)}_{uu})^\dagger \bar{H}^{(2)}_{ux}x,$$

where:

$$\begin{aligned}\bar{H}^{(2)}_{uu} &\triangleq N^{(1)}\hat{H}^{(2)}_{uu}N^{(1)} \\ \bar{h}^{(2)}_u &\triangleq \hat{h}^{(2)}_u + \hat{H}^{(2)}_{uu}\hat{k}^{(1)} \\ \bar{H}^{(2)}_{ux} &\triangleq \hat{H}^{(2)}_{ux} - \hat{H}^{(2)}_{uu}\hat{K}^{(1)}\end{aligned}$$

Note that for least-squares functions, being in the null space of H_{uu} would still be sufficient.

[1]We performed some simple tests, which showed that the SQP algorithm can find better solutions than our convex relaxation, but it usually takes tens of iterations just to find a solution of the same quality. Since this is not the main focus of the paper, for the sake of conciseness we do not report these results here.

Algorithm 1 Parametric Hierarchical Quadratic Programming

function PHQP($\{H_{uu}^{(l)}, H_{xu}^{(l)}, h_u^{(l)}, \hat{H}_{uu}^{(l)}, \hat{H}_{xu}^{(l)}, \hat{h}_u^{(l)}\}_l$)
  $N \leftarrow I, \bar{k} \leftarrow 0, \bar{K} \leftarrow 0$
  for l=1:n_l **do**
    $\bar{k} \leftarrow \bar{k} - (N\hat{H}_{uu}^{(l)}N)^\dagger(\hat{h}_u^{(l)} + \hat{H}_{uu}^{(l)}\bar{k})$
    $\bar{K} \leftarrow \bar{K} + (N\hat{H}_{uu}^{(l)}N)^\dagger(\hat{H}_{ux}^{(l)} - \hat{H}_{uu}^{(l)}\bar{K})$
    $N \leftarrow N\mathcal{N}((H_{uu}^{(l)}, h_u^{(l)\top}, H_{xu}^{(l)})N)$
  return $(\bar{k}, \bar{K})$

3) Solving the Whole Hierarchy: Generalizing this to an arbitrary number of functions n_l, we get the following solution:

$$\hat{u}^* = \bar{k} - \bar{K}x,$$

which can be computed using Algorithm 1.

IV. Hierarchical Dynamic Programming

A. Problem Statement

Let us consider a discrete-time nonlinear dynamical system:

$$x_{i+1} = f(x_i, u_i), \quad \text{for } i = 0, \dots, N-1, \tag{8}$$

where $f(\cdot) : \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}^n$ is the dynamics function, $x_i \in \mathbb{R}^n$ is the state at time step i, and $u_i \in \mathbb{R}^m$ is the control at time step i. Assume that we want the system to perform n_l tasks, with task 1 having the highest priority, and task n_l the lowest. The l-th task is represented by an arbitrary cost function:

$$c^{(l)}(X, U) := \sum_{i=0}^{N-1} \phi_i^{(l)}(x_i, u_i) + \phi_N^{(l)}(x_N)\ , \tag{9}$$

where $\phi_i^{(l)}$ is the running cost and $\phi_N^{(l)}$ is the final cost. We also have a regularized version of the cost functions:

$$\hat{c}^{(l)}(X, U) := \sum_{i=0}^{N-1} \underbrace{\phi_i^{(l)}(x_i, u_i) + \frac{\lambda^{(l)}}{2}||u_i||^2}_{\hat{\phi}_i^{(l)}(x_i, u_i)} + \phi_N^{(l)}(x_N)\ ,$$

where $\lambda^{(l)} \in \mathbb{R}$ is a regularization parameter. Other regularizations may be used if needed, as long as the Hessian of the regularization function with respect to U is positive definite. Our problem consists in finding the control and state sequences (U^*, X^*) that solve the following hierarchical optimal control problem, denoted as HOC$^{(l)}$:

$$\begin{aligned} \underset{X,U}{\text{minimize}} \quad & \hat{c}^{(l)}(X, U) \\ \text{subject to} \quad & x_{i+1} = f(x_i, u_i), \quad \text{for } i = 0, \dots, N-1 \\ & x_0 \text{ fixed} \\ & c^{(j)}(X, U) = c^{(j)}(X^{(j)*}, U^{(j)*}) \quad \forall j < l\ , \end{aligned} \tag{10}$$

for $l = 1$ to n_l, where $(X^{(j)*}, U^{(j)*})$ is the optimum obtained by solving the HOC$^{(j)}$.

B. Dynamic Programming with Regularization

We tackle problem (10) by applying the dynamic programming algorithm [20]. The principle of dynamic programming states that optimizing over the whole trajectory is equivalent to performing a sequence of optimizations over a single time step, starting from the end of the horizon and moving backwards in time. Let us start by defining the regularized and unregularized *cost-to-go* at step i for task l as:

$$c_i^{(l)}(x_i, U_i) \triangleq \sum_{j=i}^{N-1} \phi_j^{(l)}(x_j, u_j) + \phi_N^{(l)}(x_N)$$
$$\hat{c}_i^{(l)}(x_i, U_i) \triangleq \sum_{j=i}^{N-1} \hat{\phi}_j^{(l)}(x_j, u_j) + \phi_N^{(l)}(x_N)$$

The total cost corresponds to the cost-to-go for $i = 0$.

1) First Task: For task 1, we define the value function (i.e. the optimal cost-to-go), for the regularized and unregularized cost:

$$V_i^{(1)}(x_i) \triangleq \underset{U_i}{\text{minimize}}\ c_i^{(1)}(x_i, U_i)$$
$$\hat{V}_i^{(1)}(x_i) \triangleq \underset{\hat{U}_i}{\text{minimize}}\ \hat{c}_i^{(1)}(x_i, \hat{U}_i)$$

We call $U_i^{(1)}(x_i)$ and $\hat{U}_i^{(1)}(x_i)$ the optimal control laws resulting from these minimizations, and $\hat{X}^{(1)}$ the state sequence resulting from applying $\hat{U}_0^{(1)}(x_0)$. By applying Bellman's principle of optimality [21] we can reformulate these minimizations over the whole future control sequence into minimizations over a single control:

$$V_i^{(1)}(x_i) = \underset{u_i}{\text{minimize}}\ \overbrace{\phi_i^{(1)}(x_i, u_i) + V_{i+1}^{(1)}(f(x_i, u_i))}^{\mathcal{V}_i^{(1)}(x_i, u_i)} \tag{11}$$
$$\hat{V}_i^{(1)}(x_i) = \underset{\hat{u}_i}{\text{minimize}}\ \underbrace{\hat{\phi}_i^{(1)}(x_i, \hat{u}_i) + \hat{V}_{i+1}^{(1)}(f(x_i, \hat{u}_i))}_{\hat{\mathcal{V}}_i^{(1)}(x_i, \hat{u}_i)} \tag{12}$$

We call $u_i^{(1)}(x_i)$ and $\hat{u}_i^{(1)}(x_i)$ the optimal control laws resulting from this minimization.

C. Introducing the Hierarchy

For a task $l > 1$, we define the unregularized value function as the minimum cost-to-go, subject to the constraint of not affecting the cost-to-go of the higher-priority tasks:

$$\begin{aligned} V_i^{(l)}(x_i) \triangleq \underset{U_i}{\text{minimize}}\ & c_i^{(l)}(x_i, U_i) \\ & \text{subject to } c_i^{(j)}(x_i, U_i) = c_i^{(j)}(x_i, U_i^{(j)}) \quad \forall j < l \end{aligned}$$

Again, applying Bellman's principle of optimality we can reformulate this optimization as:

$$\begin{aligned} V_i^{(l)}(x_i) = \underset{u_i}{\text{minimize}}\ & \mathcal{V}_i^{(l)}(x_i, u_i) \\ & \text{subject to } \mathcal{V}_i^{(j)}(x_i, u_i) = V_i^{(j)}(x_i) \quad \forall j < l \end{aligned} \tag{13}$$

The regularized value function is instead the minimum regularized cost-to-go, subject to the constraint of not affecting the cost-to-go of the higher-priority tasks:

$$\hat{V}_i^{(l)}(x_i) \triangleq \underset{\hat{U}_i}{\text{minimize}}\ \hat{c}_i^{(l)}(x_i, \hat{U}_i) \quad \text{subject to } c_i^{(j)}(x_i, \hat{U}_i) = c_i^{(j)}(\hat{x}_i^{(j)}, \hat{U}_i^{(j)}) \quad \forall j < l \tag{14}$$

Note the difference w.r.t. the constraints of the unregularized value function: here the desired cost-to-go is a function of the optimal state $\hat{x}_i^{(j)}$ rather than the current state x_i. This is due to the fact that $U_i^{(j)}$ is the minimizer of $c_i^{(j)}$, whereas $\hat{U}_i^{(j)}$ is not. In particular, using x_i instead of $\hat{x}_i^{(j)}$ in (14) would prevent the secondary tasks from modifying the state sequence found by the first task, that is $\hat{X}^{(1)}$. This is because the control law $\hat{U}^{(1)}$ minimizes the regularized cost, so applying it from a state that does not belong to $\hat{X}^{(1)}$ would result in a different value of the unregularized cost $c_i^{(1)}$. This difference prevents us to directly apply the same reduction as in (13).

D. Reformulation of the Regularized Problem

In order to apply Bellman's principle to reformulate (14) as a problem of a single control input, we need to introduce another control law. This control law tries to maintain the unregularized cost-to-go at the same value given by the regularized control law:

$$\tilde{U}_i^{(l)}(x_i) \triangleq \text{find}\, U_i \quad \text{s.t.} \quad c_i^{(l)}(x_i, U_i) = c_i^{(l)}(\hat{x}_i^{(l)}, \hat{U}_i^{(l)})$$

Intuitively, $\tilde{U}^{(l)}$ should behave like the regularized control law in feedforward, but like the unregularized control law in feedback. As long as the state sequence follows $\hat{X}^{(l)}$ we have $\tilde{U}^{(l)} = \hat{U}^{(l)}$. However, if the state sequence is modified by other tasks, $\tilde{U}^{(l)}$ uses the feedback action to maintain the same value of the unregularized cost-to-go. By definition, this new control law allows us to reformulate (14) as:

$$\hat{V}_i^{(l)}(x_i) \triangleq \underset{\hat{U}_i}{\text{minimize}}\ \hat{c}_i^{(l)}(x_i, \hat{U}_i) \quad \text{subject to } c_i^{(j)}(x_i, \hat{U}_i) = c_i^{(j)}(x_i^{(j)}, \tilde{U}_i^{(j)}) \quad \forall j < l$$

Now we can apply Bellman's principle of optimality to reformulate the minimization of the cost function over the whole control sequence as a cascade of minimizations over a single control input. However, we can not do the same for the priority constraints: since $\tilde{U}_i^{(j)}$ is not the minimizer of $c_i^{(j)}$ the principle of optimality does not apply. This prevents us from directly reformulating the priority constraints as functions of a single control input.

E. Final HDP Formulation

We propose then to replace the priority constraints with stricter constraints that are functions of a single control input so as to benefit from the computational efficiency of dynamic programming. The new constraints state that the cost-to-go of the higher-priority tasks at each time step must stay the same—which is sufficient but not necessary to guarantee that the total cost stay the same:

$$\hat{V}_i^{(l)}(x_i) \triangleq \underset{\hat{u}_i}{\text{minimize}}\ \hat{\mathcal{V}}_i^{(l)}(x_i, \hat{u}_i) \quad \text{subject to } \tilde{\mathcal{V}}_i^{(j)}(x_i, \hat{u}_i) = \tilde{V}_i^{(j)}(x_i) \quad \forall j < l, \tag{15}$$

where $\tilde{V}_i^{(j)}(x_i)$ is the value function associated to the new control law (i.e. $\tilde{V}_i^{(j)}(x_i) = c_i^{(j)}(x_i^{(j)}, \tilde{U}_i^{(j)})$), and:

$$\tilde{\mathcal{V}}_i^{(l)}(x_i, u_i) = \phi_i^{(l)}(x_i, u_i) + \tilde{V}_{i+1}^{(l)}(f(x_i, u_i))$$

Solving this sequence of optimization problems for $l = 1, \ldots, n_l$ and for $i = N-1, \ldots, 0$, initialized with $\hat{V}_N^{(l)}(x_N) := \phi_N^{(l)}(x_N)$, we could solve (10).

V. Hierarchical Differential Dynamic Programming

The previous section derived the optimality conditions along samples of the trajectory, resulting in several difficult (typically nonconvex) optimization problems. We devised a discretized version of the conditions to improve the paper clarity, however they could be quite directly extended to the continuous case as differential conditions. This section proposes a complete algorithm to compute the solution satisfying conditions (15). Direct resolution is not tractable in general. We rather follow the route proposed initially by the DDP approach [22]. We build at each time step a quadratic approximation of the condition, which we can then solve using the proposed PHQP algorithm. The presentation of our method is self contained, but is best understood if the reader has in mind the classic DDP formulation. A concise and modern presentation of it can be found e.g. in [3].

A. Quadratic Differential Approximation

We start by considering a nominal control sequence $\bar{U} \triangleq (\bar{u}_0, \cdots, \bar{u}_{N-1})$ and the corresponding state sequence $\bar{X} \triangleq (\bar{x}_1, \cdots, \bar{x}_N)$ resulting by applying the former control to the system (8). We introduce the new variables of our optimization problem, which are the variation of control and state with respect to their nominal values: $\delta u_i \triangleq u_i - \bar{u}_i$ and $\delta x_i \triangleq x_i - \bar{x}_i$. We can now approximate $\mathcal{V}_i^{(l)}$, $\hat{\mathcal{V}}_i^{(l)}$ and $\tilde{\mathcal{V}}_i^{(l)}$ with their second-order Taylor's expansions and optimize the resulting quadratic functions with the PHQP algorithm. Let us define the local least-squares approximation of $\hat{\mathcal{V}}_i^{(l)}$:

$$\hat{\mathcal{V}}_i^{(l)}(x_i, u_i) \approx \hat{\mathcal{V}}_i^{(l)}(\bar{x}_i, \bar{u}_i) + \frac{1}{2} ||\hat{A}_{x,i}^{(l)\top} \delta x_i + \hat{A}_{u,i}^{(l)\top} \delta u_i + \hat{a}_i||^2$$

In the following, for coherence with our previous works [9], [10] and the standard DDP algorithm [3], we prefer to represent the local approximation of $\hat{\mathcal{V}}_i^{(l)}$ with a quadratic form[2]:

$$\hat{\mathcal{V}}_i^{(l)}(x_i, u_i) \approx \hat{\mathcal{V}}_i^{(l)}(\bar{x}_i, \bar{u}_i) + \begin{bmatrix} \hat{Q}_{x,i}^{(l)\top} & \hat{Q}_{u,i}^{(l)\top} \end{bmatrix} \begin{bmatrix} \delta x_i \\ \delta u_i \end{bmatrix} + \frac{1}{2} \begin{bmatrix} \delta x_i^\top & \delta u_i^\top \end{bmatrix} \begin{bmatrix} \hat{Q}_{xx,i}^{(l)} & \hat{Q}_{xu,i}^{(l)} \\ \hat{Q}_{ux,i}^{(l)} & \hat{Q}_{uu,i}^{(l)} \end{bmatrix} \begin{bmatrix} \delta x_i \\ \delta u_i \end{bmatrix} \tag{16}$$

[2]We derive the algorithm without exploiting the least-squares structure, so that all the developments are still valid for a general quadratic approximation. The only exception is the hybrid control law (see Appendix), which we derive only for the least-squares case.

The coefficients of the quadratic approximation of $\hat{\mathcal{V}}_i^{(l)}$ can be recursively computed as:

$$\begin{aligned}
\hat{Q}_{x,i}^{(l)} &\triangleq \partial_x \phi_i^{(l)} + \partial_x f^\top \partial_x \hat{V}_{i+1}^{(l)} \\
\hat{Q}_{u,i}^{(l)} &\triangleq \partial_u \phi_i^{(l)} + \partial_u f^\top \partial_x \hat{V}_{i+1}^{(l)} + \lambda^{(l)} \bar{u}_i \\
\hat{Q}_{xx,i}^{(l)} &\triangleq \partial_{xx} \phi_i^{(l)} + \partial_x f^\top \partial_{xx} \hat{V}_{i+1}^{(l)} \partial_x f + \partial_x \hat{V}_{i+1}^{(l)} \partial_{xx} f \\
\hat{Q}_{uu,i}^{(l)} &\triangleq \partial_{uu} \phi_i^{(l)} + \partial_u f^\top \partial_{xx} \hat{V}_{i+1}^{(l)} \partial_u f + \partial_x \hat{V}_{i+1}^{(l)} \partial_{uu} f + \lambda^{(l)} I \\
\hat{Q}_{xu,i}^{(l)} &\triangleq \partial_{xu} \phi_i^{(l)} + \partial_x f^\top \partial_{xx} \hat{V}_{i+1}^{(l)} \partial_u f + \partial_x \hat{V}_{i+1}^{(l)} \partial_{xu} f
\end{aligned} \tag{17}$$

The local quadratic approximation of $\mathcal{V}_i^{(l)}$ and $\tilde{\mathcal{V}}_i^{(l)}$ are defined similarly, removing the symbol $\hat{}$ or replacing it with the symbol $\tilde{}$ (respectively) and setting $\lambda^{(l)}$ to zero. All the derivatives in (17) are computed for $x_i = \bar{x}_i$ and $u_i = \bar{u}_i$.

B. Backward Pass

The computation of (17) is initialized with $\hat{V}_N^{(l)}(x_N) = \phi_N^{(l)}(x_N)$. Then we can minimize our quadratic model of $\hat{\mathcal{V}}_i^{(l)}$ using PHQP, which gives us the locally-optimal feedforward and feedback terms for task l at time i (starting from $i = N-1$). Finally, to compute the solution for time step $i-1$ we need to compute $\hat{V}_i^{(l)}(\delta x_i)$ to update our quadratic approximation of $\hat{\mathcal{V}}_{i-1}^{(l)}$. We can do it by substituting the locally-optimal control $\delta \hat{u}_i^{(l)} = \hat{k}_i^{(l)} - \hat{K}_i^{(l)} \delta x_i$ into (16), which gives us:

$$\hat{V}_i^{(l)}(\delta x_i) \approx \hat{V}_{s,i}^{(l)} + \hat{V}_{x,i}^{(l)\top} \delta x_i + \frac{1}{2} \delta x_i^\top \hat{V}_{xx,i}^{(l)} \delta x_i \ ,$$

where[3]:

$$\begin{aligned}
\hat{V}_{x,i}^{(l)} &= \hat{Q}_{x,i}^{(l)} - \hat{K}_i^{(l)\top} \hat{Q}_{u,i}^{(l)} - \hat{K}_i^{(l)\top} \hat{Q}_{uu,i}^{(l)} \hat{k}_i^{(l)} + \hat{Q}_{xu,i}^{(l)} \hat{k}_i^{(l)} \\
\hat{V}_{xx,i}^{(l)} &= \hat{Q}_{xx,i}^{(l)} + \hat{K}_i^{(l)\top} \hat{Q}_{uu,i}^{(l)} \hat{K}_i^{(l)} - \hat{Q}_{xu,i}^{(l)} \hat{K}_i^{(l)} - \hat{K}_i^{(l)\top} \hat{Q}_{ux,i}^{(l)}
\end{aligned} \tag{18}$$

For the unregularized value function $V_i^{(l)}$ we have exactly the same equations, but using the unregularized control law rather than the regularized one. For the hybrid value function $\tilde{V}_i^{(l)}$ we need to use the following control law (see proof in the Appendix):

$$\delta \tilde{u}_i^{(l)} = \hat{k}_i^{(l)} - \hat{K}_i^{(l)} \delta \hat{x}_i^{(l)} - K_i^{(l)} (\delta x_i - \delta \hat{x}_i^{(l)}) \tag{19}$$

The feedback term of this control law is the same as for the unregularized control law, whereas its feedforward term is:

$$\tilde{k}_i^{(l)} = \hat{k}_i^{(l)} + (K_i^{(l)} - \hat{K}_i^{(l)}) \delta \hat{x}_i^{(l)}$$

In our tests, to speed-up the algorithm, we neglected the terms depending on the second-order derivatives of the dynamics. This is the same approximation that distinguishes the iterative LQR algorithm [3], [23] from DDP [24]. This allows us also to avoid the computation of some terms, since we have: $\tilde{V}_{xx,i}^{(l)} = V_{xx,i}^{(l)}$, $\tilde{Q}_{uu,i}^{(l)} = Q_{uu,i}^{(l)}$, $\tilde{Q}_{ux,i}^{(l)} = Q_{ux,i}^{(l)}$, $\tilde{Q}_{xx,i}^{(l)} = Q_{xx,i}^{(l)}$.

[3]We do not report here the value of $\hat{V}_{s,i}^{(l)}$ because it does not affect the computation.

C. Regularizing the Optimization

So far we extensively discussed the issue of regularizing the cost functions describing the robot tasks, which we refer to as *task* regularization. The main goal of this regularization is to prevent the use of large control inputs. However, *algorithm* regularization remains necessary in general for a number of reasons that we describe now. The backward pass operates on a local approximation of the original problem (10). This model is only valid in the neighborhood of the current solution, so we must avoid taking too large steps. The regularization has a damping effect on near-singular directions, which tends to limit unreasonable step lengths. Moreover the Hessian Q_{uu} may become nonpositive, because the initial problem is nonconvex or because of propagation of numerical errors along the backward pass. Algorithm regularization ensures the soundness of the inversion of the Hessian.

We can achieve all of these goals by adding to all $\hat{Q}_{uu,i}^{(l)}$'s the identity matrix scaled by a sufficiently large scalar parameter $\mu^{(l)}$. Initial values for $\mu^{(l)}$ are given by the user (and may affect a lot the speed of convergence, as shown in our tests). At each iteration the algorithm decides whether to increase or decrease the current value of $\mu^{(l)}$ depending on the eigenvalues of $\hat{Q}_{uu}^{(l)}$ (after projection in the null space of previous tasks, see Algorithm 2) and the line-search parameters (more details on this in Section V-E). The null-space projectors must be computed with the unregularized pseudo-inverses to ensure the proper hierarchy propagation.

D. Order of the Operations

Now that we have defined the backward propagation of the value functions, we need to decide how to use them. In particular, we have two options:

1) Solve task l (starting from $l = 1$) for $i = N-1, \ldots, 0$ and then move on to task $l+1$.
2) Solve all tasks for time step i (starting from $i = N-1$) and then move on to time step $i-1$.

We decided to use the first option. To understand the reason behind this choice we need to look at what happens after the backward pass, that is the forward pass. During the forward pass we simulate the system forward in time using the locally-optimal control policy; if needed, we gradually reduce the magnitude of the feedforward control term until the associated cost does not improve. This is equivalent to the line-search procedure used in Sequential Quadratic Programming for nonlinear optimization [19]. During the backward pass of task $l+1$ we assume that the feedforward term of task l will be completely applied, while this may not be the case because of the potential reduction occurring in the forward pass. It could be then beneficial to perform the forward pass of task l before the backward pass of task $l+1$, so that we could account for this reduction. This is only possible if we select the first one of the two options mentioned above.

Following this choice, Algorithm 2 summarizes the backward pass. Its inputs are: the derivatives of the dynamics and the cost function $\partial_\cdot f, \{\partial_\cdot \phi_i\}_i$; the null-space projector, feedforward and feedback terms of the previous tasks $\{N_i, \bar{k}_i, \bar{K}_i\}_i$; the task regularization parameter λ; the nominal

Algorithm 2 Regularized HDDP: Backward Pass

function BACKPASS($\partial f, \{\partial \phi_i, N_i, \bar{k}_i, \bar{K}_i\}_i, \lambda, \bar{U}, \mu$)
Initialize $V^{(l)}_{x,N}$, $V^{(l)}_{xx,N}$, $\hat{V}^{(l)}_{x,N}$, $\hat{V}^{(l)}_{xx,N}$
for $i = N-1$ to 0 **do**
▷ Compute regularized control law
$(\hat{Q}^{(l)}_{x,i}, \hat{Q}^{(l)}_{u,i}, \hat{Q}^{(l)}_{xx,i}, \hat{Q}^{(l)}_{uu,i}, \hat{Q}^{(l)}_{xu,i}) \leftarrow$ (17)
$\bar{Q}_{uu} \leftarrow N_i(\hat{Q}^{(l)}_{uu,i} + \mu I)N_i$
if $\min(\text{eigenvalues}(\bar{Q}_{uu,i})) < 0$ **then**
Increase μ and repeat from line 2
$\hat{k}_i \leftarrow -\bar{Q}^{\dagger}_{uu}(\hat{Q}^{(l)}_{u,i} + \hat{Q}^{(l)}_{uu,i}\bar{k}_i)$
$\hat{K}_i \leftarrow \bar{Q}^{\dagger}_{uu}(\hat{Q}^{(l)}_{ux,i} - \hat{Q}^{(l)}_{uu,i}\bar{K}_i)$
$(\hat{V}^{(l)}_{x,i}, \hat{V}^{(l)}_{xx,i}) \leftarrow$ (18) with $\bar{k}_i + \hat{k}_i$ and $\bar{K}_i + \hat{K}_i$
▷ Compute unregularized control law
$(Q^{(l)}_{x,i}, Q^{(l)}_{u,i}, Q^{(l)}_{xx,i}, Q^{(l)}_{uu,i}, Q^{(l)}_{xu,i}) \leftarrow$ (17)
$\bar{Q}_{uu} \leftarrow N_i Q^{(l)}_{uu,i} N_i$
$k_i \leftarrow -\bar{Q}^{\dagger}_{uu}(Q^{(l)}_{u,i} + Q^{(l)}_{uu,i}\bar{k}_i)$
$K_i \leftarrow \bar{Q}^{\dagger}_{uu}(Q^{(l)}_{ux,i} - Q^{(l)}_{uu,i}\bar{K}_i)$
$(V^{(l)}_{x,i}, V^{(l)}_{xx,i}) \leftarrow$ (18) with $\bar{k}_i + k_i$ and $\bar{K}_i + K_i$
return $\{\hat{k}_i, \hat{K}_i, K_i, \mu\}_i$

control sequence $\bar{U}$ and the Hessian regularization parameter μ (see Section V-C). The central part of the algorithm is identical to the PHQP algorithm presented in Section III.

E. Forward Pass (Line Search)

As usual, the forward pass consists in a forward simulation of the system using the locally-optimal control policy computed in the backward pass. It is used as a line search, i.e. at the end of the forward simulation we check whether the cost of all tasks decrease in a lexicographic order, i.e. a decrease of the cost of task l must not lead to an increase of the cost of any task $j < l$. In practice, we allow for a small increase of the higher-priority costs to speed-up convergence: a step is validated if it leads to a decrease of the current cost that is *much larger* than the increase of the higher-priority costs. A user-defined parameter defines how much larger this improvement should be (we used 10^3 for all our tests).

If this is not the case we reduce the magnitude of the feedforward term of the control policy and repeat again. This is the control policy used during the forward pass of task j:

$$\begin{aligned}\delta\hat{u}^{(l)}_i =& \delta\hat{u}^{(l-1)}_i + \nu^{(l)}\hat{k}^{(l)}_i - \hat{K}^{(l)}_i(x_i - \bar{x}_i) \\ & - \bar{K}^{(l-1)}_i(x_i - \bar{x}_i - \delta\hat{x}^{(l-1)}_i),\end{aligned}$$

where $\delta\hat{u}^{(0)}_i = 0$, $\nu^{(l)}$ is the line-search parameter of task l and $\delta\hat{x}^{(l-1)}$ is the state deviation corresponding to the control deviation $\delta\hat{u}^{(l-1)}$. The term $\delta\hat{x}^{(l-1)}$ for the feedback of the higher-priority tasks comes from the hybrid control law (19). In this way, the unregularized feedback gains of all higher-priority tasks help not modifying the associated unregularized costs. Note that, by doing so, if $\hat{k}^{(l)}_i$ and $\hat{K}^{(l)}_i$ are void we get exactly $\delta\hat{u}^{(l)}_i = \delta\hat{u}^{(l-1)}_i$. We always initialize $\nu^{(l)} = 1$ and we decrease it with an exponential update rule:

$$\nu^{(l)} := a^l \nu^{(l)},$$

where $a \in [0, 1]$ is a parameter of the algorithm and l is the current iteration number of the line-search procedure. If the line-search does not converge after a predefined number of iterations, we increase the regularization parameters $\mu^{(l)}$ and repeat the backward pass. The pseudo-code to compute one complete step of RHDDP is reported in Algorithm 3.

F. Improving the algorithm

Because of the regularization, all tasks try to reduce the control inputs while not affecting the higher-priority tasks. This means that each task is likely to undo the action taken by the lower-priority tasks at the previous iteration of the algorithm (assuming that this action increased the control inputs). This effect should not disturb the convergence because in the backward pass each task can "see" what has been done by the higher-priority tasks, and compensate for it if that does not affect their unregularized costs. However, whenever the line search of a task converges to a small value of ν (e.g. < 0.1), its control action is drastically reduced, and so it is this compensation. In practice, the result of this phenomenon is that the low-priority tasks converge very slowly.

To avoid this effect, we modify the regularization term in the cost functions. Rather than minimizing the norm of all the control inputs, each task only minimizes the norm of the control inputs computed by itself and the higher-priority tasks. This is exactly the content of the variable $\bar{U}^{(l)}$ in Algorithm 3, which is updated in line 16.

G. Algorithm Summary

We now summarize the proposed algorithm. Each iteration is composed of the following phases:

1) Problem approximation.
2) Local control computation, or backward pass.
3) Line search, or forward pass.
4) Computation of the null space

Convergence is tested at the end of each iteration. The convergence criteria consists in an absolute criterion and a relative one. We assume that the algorithm has converged if the cost is lower than the absolute tolerance value. Alternatively, the relative improvement between two successive iterations must be smaller than a relative tolerance value.

VI. SIMULATIONS

This section presents several simulations on different robotic systems to validate the proposed algorithm (RHDDP) and compare it to the classic DDP algorithm and to the previous version of HDDP [10]. In all tests we use weights to approximate strict priorities with the DDP algorithm and we compare this approximation with our method. The main result is that obtaining a safe hierarchy behavior with optimal control is nearly impossible with weighted DDP, while it is straightforward with RHDDP. All the results of the simulations can be seen in the accompanying video.

Section VI-A presents simulations with the PR2 robot that stress the difficulty of tuning the cost weights with DDP.

Algorithm 3 Regularized Hierarchical Differential Dynamic Programming: 1 Step

1: **function** RHDDP($x_0, \bar{U}, f(.), \{\bar{U}^{(l)}, c^{(l)}(.), \lambda^{(l)}, \mu^{(l)}\}_l$)
2: $N \leftarrow I$, $\bar{k} \leftarrow 0$, $\bar{K} \leftarrow 0, \delta\hat{u} \leftarrow 0, \delta\hat{x} \leftarrow 0, \bar{U}^{(l)} \leftarrow \bar{U}$
3: $\bar{X} \leftarrow$ FORDWARDDYNAMICS$(x_0, \bar{U}, \bar{K})$
4: **for** $l = 1$ to n_l **do**
5: $(\{\hat{k}_i, \hat{K}_i, K_i\}_i, \mu^{(l)}) \leftarrow$ BACKPASS$(\partial f, \{\partial \phi_i^{(l)}, N_i, \bar{k}_i, \bar{K}_i\}_i, \lambda^{(l)}, \bar{U}^{(l)}, \mu^{(l)})$
6: $(\delta\hat{u}_i, \delta\hat{x}_i, \nu) \leftarrow$ LINESEARCH$(\{\delta\hat{u}_i, \hat{k}_i, \hat{K}_i, \bar{K}_i, \delta\hat{x}_i\}_i)$
7: **if** $\nu = 0$ **then**
8: Increase $\mu^{(l)}$ and go back to line 5
9: **if** $\nu < \nu^{thr}$ **then**
10: Increase $\mu^{(l)}$
11: **else if** $\nu = 1$ **then**
12: Decrease $\mu^{(l)}$
13: **for** $i = 0$ to $N - 1$ **do**
14: $\bar{k}_i \leftarrow \delta\hat{u}_i + K_i\delta\hat{x}_i$
15: $\bar{K}_i \leftarrow \bar{K}_i + K_i$
16: $\bar{U}_i^{(l)} \leftarrow \bar{U}_i^{(l)} + \delta\hat{u}_i$
17: Initialize $\tilde{V}_{x,N}^{(l)}$
18: **for** $i = N - 1$ to 0 **do**
19: $(\tilde{Q}_{x,i}^{(l)}, \tilde{Q}_{u,i}^{(l)}) \leftarrow$ (17)
20: $(\tilde{V}_{x,i}^{(l)}) \leftarrow$ (18) with $\bar{k}_i$ and $\bar{K}_i$
21: $N_i \leftarrow N_i\mathcal{N}((Q_{uu,i}^{(l)}, \tilde{Q}_{u,i}^{(l)\top}, Q_{xu,i})N_i)$
22: $\bar{U} \leftarrow \bar{U} + \delta\hat{u}$
23: $\bar{X} \leftarrow \bar{X} + \delta\hat{x}$
24: **return** $(\bar{U}, \bar{X}, \{\bar{U}^{(l)}, \mu^{(l)}\}_k)$

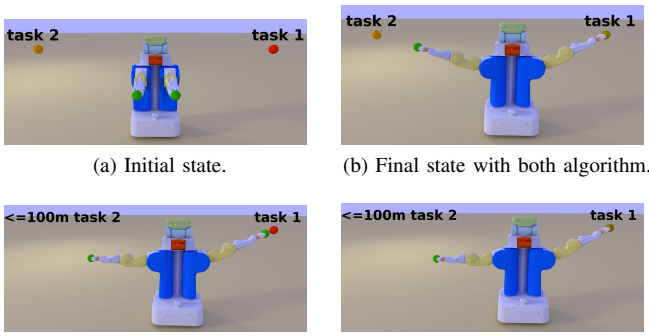
Moreover it analyzes the convergence of the RHDDP algorithm. Section VI-B (always with PR2) shows that without a hierarchy we cannot properly regularize the tasks: either regularization is too large and takes over the low-priority task, or regularization is not enough and so commands are too large. In Section VI-C we use a simple cart-pole system to highlight again—but in a different way—the benefits of the hierarchy when regularizing the tasks. Finally, Section VI-D presents a more complex set of simulations with the UR5 robot, which is asked to execute a sequence of tasks distributed in time. RHDDP allows us to trade-off accuracy and efficiency by using the regularization without compromising the hierarchy (i.e. low-priority tasks deteriorate first when regularization increases). In this test we also show that RHDDP is capable of faster convergence with respect to the old version of the algorithm (HDDP [10]).

The tests have been executed on a computer with two processors Intel Xeon CPU E5-2620V2 (6 cores 2.10GHz) and 64GB of RAM. The tested algorithms were coded in Matlab (v2013b) but the simulation of the systems is computed thanks to the C++ dynamic engine MuJoCo [25].

A. Test 1: PR2 - Final Cost

This simulation compared RHDDP and DDP in a grasping task with the PR2 robot. The goal was to perform the following four tasks (in order of priority):

1) have zero joint velocity at final time $i = N$
2) grasp left (red) ball with left gripper (final cost only)
3) grasp right (orange) ball with right gripper (final cost only)
4) minimize 2-norm of control trajectory



(a) Initial state.
(b) Final state with both algorithm.
(c) Final state with DDP and second ball 100 m away.
(d) Final state with RHDDP and second ball 100 m away.

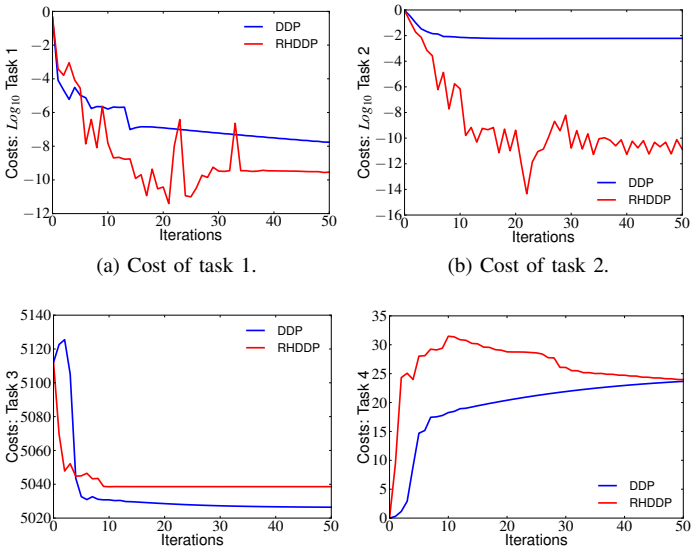
Fig. 2. Test 1: snapshots of the simulations when both balls are close to the robot (top row) and when one of the balls is far away (bottom row).

In this test we did not need to use any regularization for the tasks because they used only a final cost—the robot was not asked to reach the goal as fast as possible. Results are summarized by Fig. 2 to 4 and Table I.

When using DDP we used the following weights to approximate strict priorities: $10^6, 10^3, 1, 10^{-6}$. First we set the two balls slightly too far away from each other to be reached at the same time (see Fig. 2a). In this scenario both RHDDP and DDP managed to reach the first ball and to minimize the distance to the second ball (see Fig. 2b). Then we moved

TABLE I
TEST 1: COST FUNCTION VALUES.

Algorithm	Scenario	Task 2	Task 3	Effort
DDP	Balls close	$1.23\ 10^{-7}$	$1.01\ 10^{-1}$	$3.51\ 10^{4}$
RHDDP	Balls close	$3.65\ 10^{-13}$	$1.12\ 10^{-1}$	$2.84\ 10^{4}$
DDP	Balls far	$6.03\ 10^{-3}$	$5.03\ 10^{3}$	$2.64\ 10^{4}$
RHDDP	Balls far	$2.39\ 10^{-10}$	$5.04\ 10^{3}$	$2.73\ 10^{4}$



(a) Cost of task 1. (b) Cost of task 2.

(c) Cost of task 3. (d) Cost of task 4.

Fig. 3. Test 1: cost of the different tasks over the iterations of the algorithm when second ball is 100 m away from the robot.

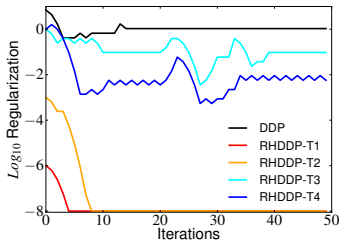


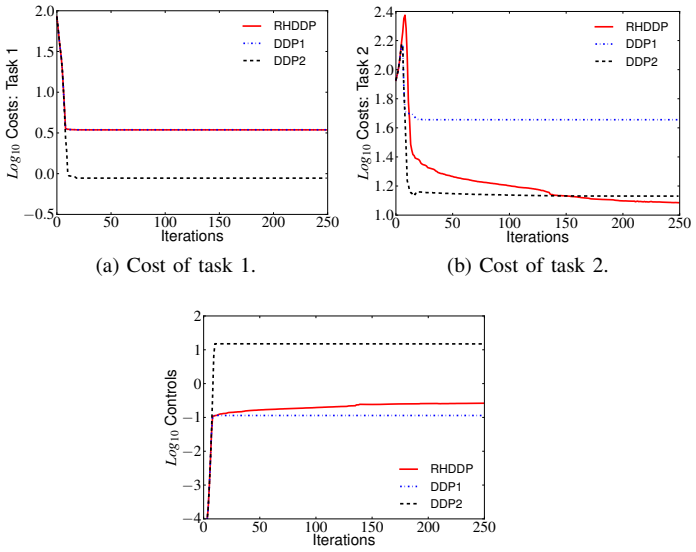
Fig. 4. Test 1: value of the regularization parameters throughout the iterations of the RHDDP (one value for each task) and DDP algorithm.

the second ball 100 meters away from the robot. In this case DDP did not manage to reach the first ball (see Fig. 2c), while RHDDP did (see Fig. 2d). This clearly shows that weights are task dependent, and using strict priorities provides more robust behaviors. Table I shows the values of the cost functions with RHDDP and DDP.

1) Convergence of the Algorithm: Fig. 3 shows the value of the cost functions throughout the iterations of RHDDP and DDP. Fig. 4 instead shows the values taken by the regularization parameters $\mu^{(l)}$. With RHDDP the regularization quickly converged to its minimum value (10^{-8}) for the first two tasks, which allows a fast convergence to a good approximate solution. On the other hand, DDP convergence is delayed by the algorithm regularization, which never reaches its minimum value due to the artificial ill conditioning introduced by the weight scaling. Convergence of DDP is somehow sequential:

TABLE II
TEST 2: COST FUNCTION VALUES.

Algorithm	Regularization	Task 1	Task 2	Effort Max
DDP	10^{-5}	3.46	45.3	238
DDP	10^{-8}	0.881	13.5	3325
RHDDP	10^{-5}	3.45	12.2	319



(a) Cost of task 1. (b) Cost of task 2.

(c) Cost of task 3.

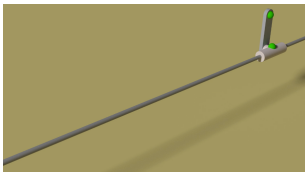
Fig. 5. Test 2: cost of the different tasks over the iterations of the algorithm. In the legend, DDP1 is DDP with $w_r = 10^{-5}$, while DDP2 is DDP with $w_r = 10^{-8}$.

low-priority tasks improve only after high-priority tasks have converged (e.g. see Fig. 3b and 3c).

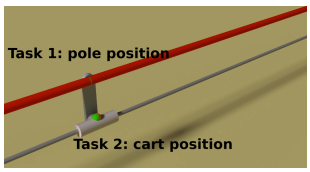
B. Test 2: PR2 - Integral Cost

This test is based on the same scenario of the previous test (with the second ball almost reachable), but i) it uses integral cost functions rather than final costs only, and ii) it does not include the final zero-velocity task. Since we are using integral costs, we need to regularize the tasks to avoid too large commands. For DDP we used $w_1 = 1$ (weight of the first task), $w_2 = 10^{-3}$ (weight of the second task) and we tried two different values of w_r (weight of the regularization, i.e. minimum-effort task). Results are summarized by Fig. 5 and Table II. In brief, it was not possible to find a correct value of w_r for DDP, while it was immediate for RHDDP.

Using $w_r = 10^{-5}$ was enough to avoid too large control inputs, but resulted in a large cost for the second task. Using $w_r = 10^{-8}$ allowed us to improve the execution of the second task, but resulted in large control inputs. With RHDDP we used $w_r = 10^{-5}$ for the regularization, which was large enough to avoid large commands; at the same time, also the second task was executed at its best. This is thanks to the fact that RHDDP allows us to keep same ratio between task and regularization weights, which is not the case for DDP. With DDP, when using $w_r = 10^{-5}$, the ratio between w_2 and w_r was only 10^{-2}, so the performance of the second task was negatively affected.

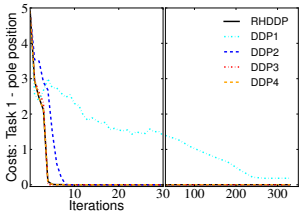


(a) Initial state.

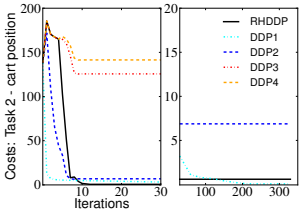


(b) Goal state.

Fig. 6. Test 3: initial and goal state of the cart-pole.

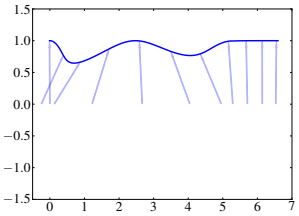


(a) Cost of task 1. After iteration 30 the scale of the x axis changes.

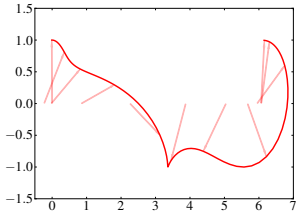


(b) Cost of task 2. After iteration 30 the scale of the x and y axes changes.

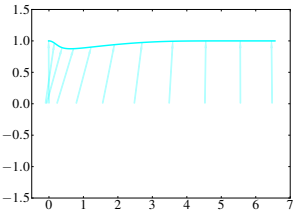
Fig. 7. Test 3: value of the cost functions throughout the iterations of the algorithm. In the legends, DDP1 to DDP4 represent DDP with $w_2 = 1$ to 10^{-3}, respectively.



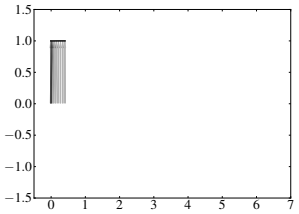
(a) RHDDP.



(b) DDP with $w_2 = 1$.



(c) DDP with $w_2 = 0.1$.



(d) DDP with $w_2 = 0.01$.

Fig. 8. Test 3: trajectory of the cart-pole system obtained with the different algorithms. The cart start at position 0 and should move to position 6.

RHDDP and DDP with $w_r = 10^{-5}$ resulted in almost identical costs for task 1 and 3 (see Table II), but RHDDP performed better task 2. Increasing w_2 would improve task 2, but at the expenses of task 1 and 3. Increasing w_1 and w_2 is equivalent to decreasing w_r and would worsen task 3. This suggests that RHDDP allowed us to get a behavior that we could not achieve by using DDP, no matter the values of w_1, w_2 and w_r.

C. Test 3: Cart-Pole

In this simulation we tested RHDDP and DDP with a simple underactuated system: the cart-pole. In order of priority, the tasks to perform were:

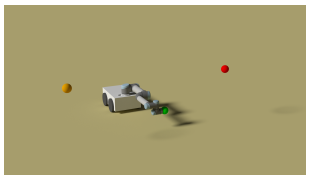
1) keep the pole high (integral cost between $0.7\,N$ and N)
2) reach a goal position with the cart (integral cost between $0.7\,N$ and N)

TABLE III
TEST 3: COST FUNCTION VALUES.

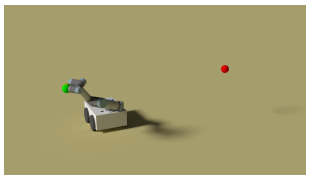
Algorithm	w_2	w_r	Task 1	Task 2	Effort
RHDDP	N.A.	0	$1\ 10^{-5}$	$5\ 10^{-5}$	$4.55\ 10^{2}$
RHDDP	N.A	10^{-2}	$3.65\ 10^{-4}$	$5.93\ 10^{-1}$	$1.60\ 10^{2}$
DDP	1	10^{-2}	$1.84\ 10^{-1}$	$6.66\ 10^{-2}$	$1.11\ 10^{2}$
DDP	10^{-1}	10^{-2}	$2.55\ 10^{-4}$	6.88	$1.60\ 10^{1}$
DDP	10^{-2}	10^{-2}	$3.77\ 10^{-6}$	$1.25\ 10^{2}$	$7.83\ 10^{-2}$
DDP	10^{-3}	10^{-2}	$1.25\ 10^{-7}$	$1.41\ 10^{2}$	$1.25\ 10^{-3}$

TABLE IV
TEST 4: COST FUNCTION VALUES.

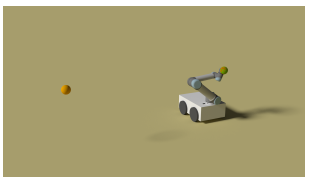
w_r	Task 1	Task 2	Task 3	Task 4	Task 5	Effort
5 10^{-5}	1.4 10^{-8}	3.3 10^{-6}	4.4 10^{-6}	8.3 10^{-5}	1.1 10^{-4}	2.8 10^{-1}
5 10^{-4}	4.7 10^{-12}	7.5 10^{-7}	1.7 10^{-5}	1.0 10^{-2}	4.6 10^{-3}	3.6 10^{-1}
5 10^{-3}	2.2 10^{-9}	3.7 10^{-4}	4.6 10^{-3}	9.8 10^{-1}	2.8 10^{-1}	2.7 10^{-1}
5 10^{-2}	1.4 10^{-9}	6.5 10^{-2}	3.7 10^{-2}	26.5	16.1	8.0 10^{-2}



(a) Initial state.



(b) Task 4: reach ball at time step $i = N/2$.



(c) Task 2: reach ball at time step $i = N$.

Fig. 9. Test 4: snapshots of the simulation.

The two tasks are compatible only if we allow the system to use large control inputs (see first line of Table III). With DDP, we kept constant $w_1 = 1$ (weight of the first task) and $w_r = 10^{-2}$ (weight of the regularization), whereas we varied w_2 (weight of the second task). Using $w_2 = 10^{-1}$ or less, only task 1 was achieved (see corresponding lines in Table III) because w_r was too large w.r.t. w_2. Using $w_2 = 1$ allowed the system to execute the second task, but it deteriorated the performance of the first task (see third line of Table III). With RHDDP we used $w_r = 10^{-2}$ as well, but the hierarchy allowed for the execution of the second task while not deteriorating the first task (see second line of Table III). Fig. 8 depicts some of the trajectories found by DDP and RHDDP.

D. Test 4: UR5 - Sequential Tasks

In this test we used the UR5 robot to reach two balls one after the other. The first goal of this test is to show how, with RHDDP, increasing the regularization affects first the

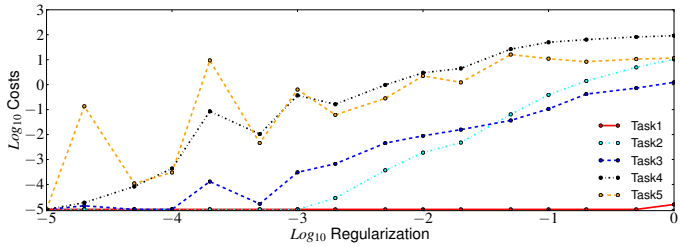
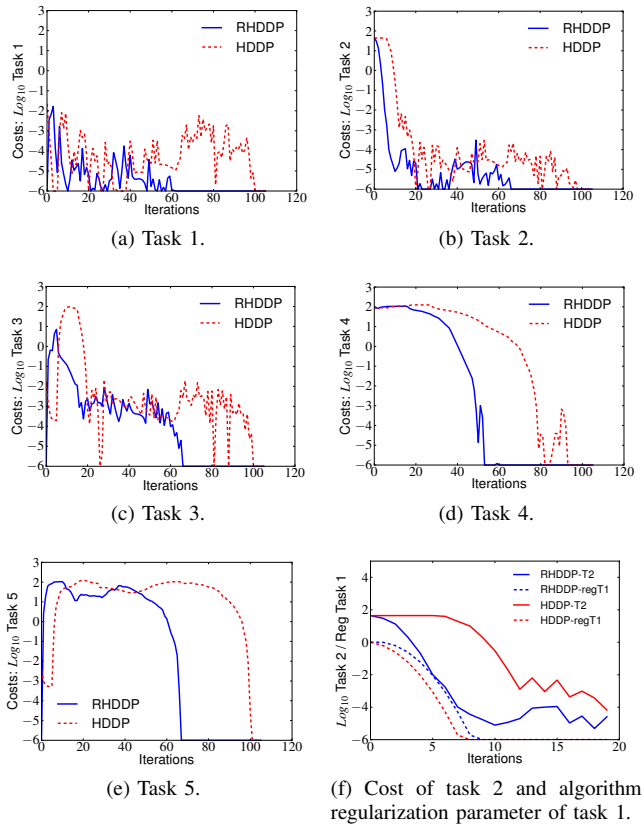


Fig. 10. Test 4: values of the cost functions obtained by using different values of the regularization parameter λ.



(a) Task 1. (b) Task 2.

(c) Task 3. (d) Task 4.

(e) Task 5. (f) Cost of task 2 and algorithm regularization parameter of task 1.

Fig. 11. Test 4: convergence of the cost functions using the algorithm presented in this paper (RHDDP) and its previous version (HDDP [10]).

low-priority tasks. In order of priority, the tasks to perform were:

1) have zero velocity at time step N
2) reach second ball at time step N (see Fig. 9c)
3) have zero velocity for the gripper between $0.9\,N$ and N
4) reach first ball at time step $0.45\,N$ (see Fig. 9b)
5) have zero velocity for the gripper between $0.4\,N$ and $0.5\,N$

All the tasks are compatible if we allow the system to use large control inputs. Table IV and Fig. 10 summarize the results. We observed that w_r can be increased to force the system to use smaller control inputs, and it progressively deteriorates the low-priority tasks.

1) Comparison with HDDP: We also solved this problem with the previous version of our algorithm (HDDP [10]), but without using any task regularization because this was not allowed by HDDP. Fig. 11 shows that, thanks to the improvements proposed in this paper, convergence is faster with RHDDP. Moreover, Fig. 11f shows that HDDP starts decreasing the cost of task 2 only after the algorithm regularization of task 1 has decreased. This is due to an incorrect handling of the algorithmic regularization in HDDP, which has been addressed in RHDDP. As a result, RHDDP is capable of decreasing the cost of task 2 while optimizing task 1.

TABLE V
TEST 4: CONVERGENCE OF DDP/HDDP/RHDDP. TIME IS EXPRESSED IN SECONDS.

μ_{init}	DDP0		DDP2		DDP3		HDDP		RHDDP	
	It.	Time	It.	Time	It.	Time	It.	Time	It.	Time
10^{-6}	7	2.8	9	3.8	65	28	245	299	157	197
10^{-4}	7	2.9	35	14.3	>500	>250	83	105	58	77
10^{-2}	8	3.4	259	116	>500	>250	98	130	58	77
10^{0}	12	5.3	393	179	>500	>250	100	129	63	82.3
10^{2}	14	5.8	>500	>250	>500	>250	113	141	89	111
10^{4}	16	6.7	>500	>250	>500	>250	116	146	72	90.8
10^{6}	21	9.0	>500	>250	>500	>250	118	148	70	88

2) Comparison with DDP: Finally, we solved the same problem also with DDP, trying three different sets of values for the task weights and different initial values of the algorithmic regularization parameter μ. Due to the lack of task regularization all the tasks are compatible, so the choice of the weights does not affect the final results, but it affects the convergence of the algorithm. Table V summarizes the results.

Let us define the weight distance Δw as the ratio between the weights of the neighbor tasks, e.g. w_1/w_2. When $\Delta w = 1$ (DDP0) the algorithm converged quickly regardless of the value of μ_{init}. With $\Delta w = 10^2$ (DDP1), the artificial ill-conditioning introduced by the weights slows down the convergence, especially for large values of μ_{init}. With $\Delta w = 10^3$ (DDP3) the ill-conditioning has an even stronger effect on the convergence, which is much slower for all values of μ_{init}. More in details, the reason why DDP2 and DDP3 converged faster for small values of μ_{init} lies in the line search. For instance, with DDP3 task 5 has a weight of 10^{-6}, so starting with $\mu_{init} = 10^{-2}$ the regularization has a much larger weight than task 5. The result is that, during the first steps, the other tasks converge to a local minimum (because compared to them, the regularization is not so large) and μ reduces until task 5 can be solved. At this point, the algorithm needs to reduce $c^{(5)}$ without worsening the other costs, e.g. for a step that reduces $c^{(5)}$ of 1, $c^{(1)}$ should increase no more than 10^{-12} (because task 1 has a weight of 10^6). This is almost impossible because of the nonlinearity of the problem, so the algorithm can only take little steps to reduce the last tasks, which leads to a slow convergence. On the contrary, using a smaller value of μ_{init}, all tasks can converge during the first iterations. Since task 1 decreases, even if task 5 affects a bit task 1, it is tolerated as long as the weighted sum of the costs decreases.

RHDDP and HDDP seems more robust to variations of μ_{init}. The reason behind this is that the line search of (R)HDDP is not negatively affected by the scaling introduced by the weights. In our tests, (R)HDDP validates a step if it leads to an improvement of the current cost that is 10^3 times larger than the deterioration of the higher-priority costs

(see Section V-E). Moreover, as expected, RHDDP converges always in less iterations than HDDP.

VII. Discussion

There are several benefits of the proposed approach with respect to non-hierarchical trajectory optimization. First, fixing priorities is usually much easier for the user than finding good weights, and it results in more robust behaviors (i.e. weights may need to be retuned if the task is modified, as shown in Section VI-A). Second, RHDDP allows us to properly regularize the tasks. As shown in Section VI-B and VI-C we can arbitrarily increase regularization while still executing the secondary tasks and satisfying the priority constraints. Also compared to other hierarchical trajectory optimization methods [14], our approach presents some benefits. In particular, it exploits the sparsity of the HOC problem and it properly handles the task regularization.

One limitation of our approach is that, in order to get a low computational cost for each iteration we replaced some constraints with stricter constraints. This allows us to guarantee the satisfaction of the original constraints, at the expenses of restricting the space of solutions explored by our algorithm. Let us illustrate this with a simple 1-dimensional dynamical system:

$$x_{i+1} = x_i + \delta t \, u_i$$

Suppose that the system starts at $x_0 = 0$ and its first task is to minimize the cost $(x_N - 5)^2$. If we regularize the task, the system will not reach exactly 5, but it will stop before, e.g. at $x_N = 4.7$. Now let us introduce a second task, of lower priority, which consists in minimizing the cost $(x_{N/2} - 10)^2$. Again, due to the task regularization, the system will not reach $x = 10$, but will stop before, e.g. at $x_{N/2} = 9$. Starting from $x_{N/2} = 9$, the best thing to do to preserve the same cost of task 1 (i.e. $(4.7-5)^2$) would be to reach $x_N = 5.3$. However, RHDDP is not able to do that, and will still reach $x_N = 4.7$, thus using larger control inputs to obtain the same cost. This limitation is due to the relaxation of the priority constraints that we proposed in Section II-C. One way to overcome this issue would be to use inequalities to express the priority constraints as:

$$g^{(j)}(y) \leq g^{(j)}(\hat{y}^{(j)})$$

This would also preserve the convexity of the regularized HQP problem, but it would increase the computational cost of the algorithm due to the need of handling the inequalities (e.g. by using an interior-point method [19]).

Let us now consider another example using the same simple dynamical system. Suppose that the first task consists in minimizing $\sum_{i=0}^{N}(x_i - 5)^2$. Given that the task is regularized, the resulting state sequence is depicted by the blue continuous line in Fig. 12, which results in a cost of task 1 of about 34.8. Clearly, there exist infinitely many other state sequences that result in the same cost of task 1, e.g. the other two lines in Fig. 12. However, RHDDP cannot exploit this redundancy to achieve secondary tasks because of the relaxation introduced in Section IV-B. Rather than constraining the total cost of the higher-priority tasks, RHDDP constrains their cost-to-go

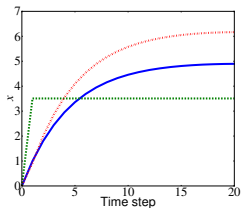


Fig. 12. Example of trajectory redundancy: these three state sequences result in the same value of the cost function: $\sum_{i=0}^{N}(x_i - 5)^2$.

at each time step, which removes part of the redundancy in case of integral costs. To overcome this issue we should abandon dynamic programming and study another algorithm to exploit the sparse structure of the HOC problem. Even if the computational complexity may be similar to RHDDP, the resulting algorithm would be harder to implement. DDP also directly provides the feedback gains for free, which makes close-loop trajectory execution possible. This advantage would be lost with another approach.

VIII. Conclusions

This paper presented a novel algorithm for motion generation of nonlinear dynamical system. The motion is generated by performing a hierarchical minimization of a number of functions of the system state and control sequences. Each function describes a task that the robot should achieve, and its position in the hierarchy defines its priority level w.r.t. the other tasks. This concept of *strict* priorities, as opposed to the *soft* priorities obtained by using a weighted sum, allows for an easier and more robust specification of the motion objectives. In particular, no weight tuning is necessary and numerical ill conditioning is avoided. Moreover, the proposed algorithm properly handles the issue of regularizing the tasks. Our tests clearly show the benefit of the hierarchy, which allows us to properly regularize the problem while preserving the strict priorities. This was not possible with soft priorities.

Another contribution of the paper regards the regularization of a hierarchy of quadratic functions (HQP), the optimization problem that is at the core of our algorithm. Even if the use of regularization is ubiquitous in robotics optimization problems, we are the first to define the regularized HQP problem and to show that it is nonconvex. We then proposed a convex relaxation of this nonconvex problem, which allows us to get an approximate solution in a single iteration. We also showed the importance of both the hierarchy and the regularization to generate safe behavior with autonomous robots in simulation.

Appendix

Here we prove the soundness of the hybrid control law (19). Recalling the definition given in Section IV-D, the hybrid control law $\tilde{U}_i^{(k)}$ must satisfy this equation:

$$c_i^{(l)}(x_i, \tilde{U}_i^{(l)}) = c_i^{(l)}(\hat{x}_i^{(l)}, \hat{U}_i^{(l)})$$

As usual, we can reformulate this as a constraint on the single control input $\tilde{u}_i^{(l)}$:

$$\tilde{\mathcal{V}}_i^{(l)}(x_i, \tilde{u}_i^{(l)}) = \tilde{\mathcal{V}}_i^{(l)}(\hat{x}_i^{(l)}, \hat{u}_i^{(l)})$$

Using the least-squares approximation of $\tilde{\mathcal{V}}$ and dropping the indexes i and l for the sake of simplicity we get:

$$\frac{1}{2}\, y^\top A^\top A y + h^\top y = \frac{1}{2}\, \hat{y}^\top A^\top A \hat{y} + h^\top \hat{y},$$

where $A = \begin{bmatrix} A_x & A_u \end{bmatrix}$, $y = (\delta x, \delta \tilde{u})$ and $\hat{y} = (\delta \hat{x}, \delta \hat{u})$. Let us perform a change of variable $y = \hat{y} + \Delta y$, where $\Delta y = (\Delta x, \Delta u)$:

$$\frac{1}{2}\, \Delta y^\top A^\top A \Delta y + \hat{y}^\top A^\top A \Delta y + a^\top A \Delta y = 0$$

Suppose now that $\Delta u = -K \Delta x$:

$$\left(\frac{1}{2}\, \Delta x^\top \begin{bmatrix} I & -K^\top \end{bmatrix} A^\top + \hat{y}^\top A^\top + a^\top \right) A \begin{bmatrix} I \\ -K \end{bmatrix} \Delta x = 0$$

A sufficient condition to satisfy this equation for any value of Δx is:

$$\begin{aligned} A \begin{bmatrix} I \\ -K \end{bmatrix} &= 0 \\ A_x - A_u K &= 0 \\ K &= A_u^\dagger A_x \\ K &= (A_u^\top A_u)^\dagger A_u^\top A_x \\ K &= Q_{uu}^\dagger Q_{ux} \end{aligned}$$

This gives us the hybrid control law (19):

$$\delta \tilde{u} = \delta \hat{u} - Q_{uu}^\dagger Q_{ux} (\delta x - \delta \hat{x})$$

REFERENCES

[1] D. M. O. Von Stryk, "Numerical solution of optimal control problems by direct collocation," in *Optimal Control*. Springer, 1993, pp. 129–143.

[2] K. Mombaur, R. Longman, H. G. Bock, and J. Schlöder, "Open-loop stable running," *Robotica*, vol. 23, no. 1, pp. 21–33, jan 2005.

[3] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference on*, 2012, pp. 4906–4913.

[4] J. Koenemann, A. Del Prete, Y. Tassa, E. Todorov, O. Stasse, M. Bennewitz, and N. Mansard, "Whole-body Model-Predictive Control applied to the HRP-2 Humanoid," in *Intelligent Robots and Systems (IROS 2015), IEEE International Conference on*, 2015.

[5] A. Escande, N. Mansard, and P.-B. Wieber, "Fast resolution of hierarchized inverse kinematics with inequality constraints," in *Robotics and Automation (ICRA), IEEE International Conference on*, no. 4. IEEE, 2010, pp. 3733–3738.

[6] B. Siciliano and J. J. E. Slotine, "A general framework for managing multiple tasks in highly redundant robotic systems," in *Advanced Robotics, 'Robots in Unstructured Environments', 91 ICAR, Fifth International Conference on*. IEEE, 1991, pp. 1211–1216.

[7] L. Saab, O. E. Ramos, N. Mansard, P. Soueres, and J.-y. Fourquet, "Dynamic Whole-Body Motion Generation under Rigid Contacts and other Unilateral Constraints," *IEEE Transactions on Robotics*, vol. 29, no. 2, pp. 346–362, 2013.

[8] A. Del Prete, F. Nori, G. Metta, and L. Natale, "Prioritized Motion-Force Control of Constrained Fully-Actuated Robots: "Task Space Inverse Dynamics"," *Robotics and Autonomous Systems*, vol. 63, pp. 150–157, 2015.

[9] A. Del Prete, F. Romano, L. Natale, G. Metta, G. Sandini, and F. Nori, "Prioritized Optimal Control," in *Robotics and Automation (ICRA), IEEE International Conference on*, 2014.

[10] F. Romano, A. Del Prete, N. Mansard, and F. Nori, "Prioritized Optimal Control: a Hierarchical Differential Dynamic Programming approach," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, 2015.

[11] S. Chiaverini, B. Siciliano, and O. Egeland, "Review of the Damped Least-Squares Inverse Kinematics with Experiments on an Industrial Robot Manipulator," *Control Systems Technology, IEEE Transactions on*, vol. 2, no. 2, pp. 123–134, 1994.

[12] W. Decré, H. Bruyninckx, and J. De Schutter, "Extending the iTaSC Constraint-based Robot Task Specification Framework to Time-Independent Trajectories and User-Configurable Task Horizons," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, 2013, pp. 1933–1940.

[13] D. Dimitrov, A. Sherikov, and P.-B. Wieber, "Efficient resolution of potentially conflicting linear constraints in robotics," *IEEE Transaction on Robotics (under review)*, 2015.

[14] Y. Tazaki and T. Suzuki, "Constraint-Based Prioritized Trajectory Planning for Multibody Systems," *IEEE Transactions on Robotics*, vol. 30, no. 5, pp. 1227–1234, 2014.

[15] H. Isermann, "Linear lexicographic optimization," *OR Spektrum*, vol. 4, no. 4, pp. 223–228, dec 1982.

[16] A. Escande, N. Mansard, and P.-B. Wieber, "Hierarchical Quadratic Programming: Fast Online Humanoid-Robot Motion Generation," *International Journal of Robotics Research*, vol. 33, no. 7, pp. 1006–1028, 2014.

[17] D. Dimitrov, P.-B. Wieber, and A. Escande, "Multi-Objective Control of Robots," *Journal of the Robotics Society of Japan*, vol. 32, no. 6, pp. 512–518, 2014.

[18] A. Herzog, N. Rotella, S. Mason, F. Grimminger, S. Schaal, and L. Righetti, "Momentum control with hierarchical inverse dynamics on a torque-controlled humanoid," *Autonomous Robots*, vol. 40, no. 3, pp. 473–491, 2016.

[19] S. Wright and J. Nocedal, *Numerical optimization*, 1999.

[20] D. E. Kirk, *Optimal control theory: an introduction*. Courier Dover Publications, 1970.

[21] R. Bellman and S. Dreyfus, *Applied dynamic programming*, 2015.

[22] D. Jacobson and D. Mayne, "Differential dynamic programming," 1970.

[23] E. Todorov and W. Li, "A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems," in *Proceedings of the 2005, American Control Conference*, 2005.

[24] D. Mayne, "A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems," *International Journal of Control*, vol. 3, p. 8595, 1966.

[25] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference on*, 2012.