Prioritized Optimal Control: a Hierarchical Differential Dynamic Programming approach

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

Prioritized Optimal Control: a Hierarchical Differential Dynamic Programming approach

Francesco Romano[1], Andrea Del Prete[2], Nicolas Mansard[2] and Francesco Nori[1]

***Abstract*— This paper deals with the generation of motion for complex dynamical systems (such as humanoid robots) to achieve several concurrent objectives. Hierarchy of tasks and optimal control are two frameworks commonly used to this aim. The first one specifies control objectives as a number of quadratic functions to be minimized under strict priorities. The second one minimizes an arbitrary user-defined function of the future state of the system, thus considering its evolution in time. Our recent work on *prioritized optimal control* merges the advantages of both these methods. This paper reformulates the original prioritized optimal control algorithm with the precise goal of improving its computational speed. We extend the dynamic programming method to work with a hierarchy of tasks. We compared our approach in simulation with both our previous algorithm and classical optimal control. The measured computational improvement represents another step towards the application of prioritized optimal control for online model predictive control of humanoid robots. We believe that this could be the key to unlock the (so far unexploited) dynamic capabilities of these mechanical systems.**

I. Introduction

Control of underactuated nonlinear mechanical systems such as humanoids and legged robots is still a main concern for the control community. Given the high number of degrees of freedom (DoFs), it is practical to formulate the control objective in terms of multiple tasks to achieve at the same time. For instance, to make a humanoid robot walk, we can control the trajectory of its center of mass and its swinging foot, the force exerted by its supporting foot and its whole-body posture. If we also add manipulation objectives, it is clear that the number of tasks rapidly grows. In case of conflict between two or more tasks, we might require the most important task to be achieved at the expenses of the others. This approach — known as prioritized or hierarchical control — has been used in robotics and computer animation since the 80's [1]. Researchers have applied prioritized control in different forms. The basic formalism defines a hierarchy among cartesian velocities of multiple points of a robotic structure [1], [2]. The same applies by considering the dynamic model, while imposing a hierarchy among operational references [3], [4], thus allowing the control of contact forces, besides cartesian and joint-space motion.

*This work was supported by the FP7 EU projects CoDyCo (No. 600716 ICT 2011.2.1 Cognitive Systems and Robotics), Koroibot (No. 611909 ICT-2013.2.1 Cognitive Systems and Robotics) and Entracte (grant ANR 13-CORD-002-01).

[1] RBCS Department, Istituto Italiano di Tecnologia, Genova, Italy. Email: name.surname@iit.it

[2] CNRS, LAAS, 7 avenue du colonel Roche, Univ de Toulouse, LAAS, F-31400 Toulouse, France. Email: adelpret@laas.fr, nmansard@laas.fr

In recent years, new formulations [5], [6] have contributed to improving the computational efficiency of this approach. [7], [8] have studied efficient ways to include inequalities in the problem formulation, which for instance can model joint limits and torque bounds. Research in computer animation [9], [10] has followed a similar path, trying to generate artificial motion by solving one or more Quadratic Programs (QP) (i.e. optimization problems with quadratic cost and linear constraints).

On the other hand, another well-known technique is *optimal control*, which minimizes a function of state and control of the system over a predefined time frame [11]. The control law automatically results as the solution of the associated optimization problem. Depending on the problem structure and the resolution method, the control law can be either a feedback control policy or a pure open-loop trajectory. Differently from prioritized control, optimal control takes decisions in accordance to future predictions, but it does not handle properly the multi-task scenario.

In [12] we introduced strict task prioritization in the optimal control formulation, which we recall in Sections II. Throughout this paper we will refer to it as POC (Prioritized Optimal Control). The developed algorithm gathers the benefits of prioritized control and optimal control. However, the approach in [12] 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. This work revisits the original POC algorithm with the main goal of addressing the above-mentioned issue, i.e. exploiting the sparsity. To accomplish this, in Section III we introduce a hierarchical model into the dynamic programming equation. Then in Section IV we presents the corresponding nonlinear heuristics (i.e. regularization and line search). Comparisons with the previous algorithm show the performance improvement in Section V.

II. Background

A. Notation

The following notation is used throughout the paper:

- $x_i \in \mathbb{R}^n$ is the state variable at the time i.
- $u_i \in \mathbb{R}^m$ is the control variable at the time i.
- $X_j := (x_j, \cdots, x_N)$ and $U_j := (u_j, \cdots, u_{N-1})$ are the partial state and control trajectories, respectively, from time j.
- $X := X_0$ and $U := U_0$ are the entire state and control trajectories.

- $\partial_y g$ 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.
- The superscript $(\cdot)^*$ denotes the minimum value and the corresponding decision variable for a minimization problem: $p^*(z^*) := \min_z p(z)$.
- $A^\dagger \in \mathbb{R}^{m \times n}$ is the Moore-Penrose pseudoinverse of the matrix $A \in \mathbb{R}^{n \times m}$.

B. Problem Formulation

Let us consider a discrete-time nonlinear dynamical system

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

where $f(\cdot) : \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}^n$ is the dynamics function. Assume that the system has to perform K tasks, with task 1 having the highest priority, and task K the lowest. The k-th task is represented with an arbitrary cost function:

$$G^{(k)}(X, U) := \sum_{i=0}^{N-1} \phi^{(k)}(x_i, u_i) + \phi_N^{(k)}(x_N) \ , \tag{2}$$

where $\phi(\cdot) : \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}$ is the running cost and $\phi_N(\cdot) : \mathbb{R}^n \mapsto \mathbb{R}$ is the final cost. The control problem is to find the control input sequence U^* and state trajectory X^* that solve the following hierarchical optimal control problem, denoted by HOC$^{(k)}$:

$$\begin{aligned} \underset{U,X}{\text{minimize}} \quad & G^{(k)}(X, U) \\ \text{subject to} \quad & x_{i+1} = f(x_i, u_i), \quad \text{for } i = 0, \dots, N-1 \\ & x_0 \text{ fixed} \\ & G^{(j)}(X, U) = G^{(j)*} \quad \forall j < k \ , \end{aligned} \tag{3}$$

for $k = 1$ to K, where $G^{(j)*}$ is the optimum obtained by solving the HOC$^{(j)}$. We define the hierarchical optimal control problem composed of the K tasks as the solution of the cascade of HOC$^{(k)}$. This definition of "*cascade*" follows the construction made in [13] for hierarchical quadratic problems.

C. Algorithm Outline

Before delving into the mathematical details, let us introduce the principles of the algorithm. Each iteration consists of the following three phases:

1) Problem approximation.
2) Local control computation, or backward pass.
3) System simulation, or forward pass.

We start by approximating the dynamics and the cost along an initial nominal state-control trajectory. Then in the backward pass — which is the main contribution of this paper — we compute the local control modification as the solution to a hierarchical optimal control problem for the approximated model. Because this control modification is based on a local model, we have to check how it performs on the real system (i.e. we integrate the dynamics with the new control trajectory and compute the new costs). The so-called line-search procedure takes care of reducing the magnitude of the control modification to compensate for the mismatch between approximated and real model. Finally we introduce a regularization procedure to solve two issues commonly arising in this class of algorithms: solutions far from the local validity of the model and ill-conditioning due to finite-precision arithmetic.

III. HDDP ALGORITHM: LINEAR PART

A. Dynamic Programming

We solve the problem (3) by applying the dynamic programming algorithm [11]. Let us start by defining the *cost-to-go* at step i for task k as:

$$G_i^{(k)}(x_i, U_i) := \sum_{j=i}^{N-1} \phi^{(k)}(x_j, u_j) + \phi_N^{(k)}(x_N) \ .$$

Note that the total cost corresponds to the cost-to-go at step $i = 0$. The optimal cost-to-go, or value function, is its minimum value:

$$V_i^{(k)}(x_i) := \min_{U_i} G_i^{(k)}(x_i, U_i) \ .$$

By applying Bellman's principle of optimality [14] we get the recurrence equation of dynamic programming. This formulation allows you to minimize over a single time step at a time, instead of minimizing over the whole trajectory. The recurrence equation is:

$$V_i^{(k)}(x_i) := \min_{u_i} \phi^{(k)}(x_i, u_i) + V_{i+1}^{(k)}(f(x_i, u_i)) \ , \tag{4}$$

with $i = N - 1, \cdots, 0$ initialized with $V_N^{(k)}(x_N) := \phi_N^{(k)}(x_N)$. To simplify the notation we will use the following definition:

$$\mathcal{V}_i^{(k)}(x_i, u_i) := \phi^{(k)}(x_i, u_i) + V_{i+1}^{(k)}(f(x_i, u_i)). \tag{5}$$

Solving the above equation in the nonlinear context is computationally infeasible even for low-dimensional state and control spaces (Bellman's curse of dimensionality). The route followed in this paper, instead, is to iteratively solve local quadratic approximations of the value function.

B. Hierarchical Dynamic Programming

We start by considering a nominal control policy $(\bar{u}_0, \cdots, \bar{u}_{N-1}) =: \bar{U}$ and the corresponding state trajectory $(\bar{x}_1, \cdots, \bar{x}_N) =: \bar{X}$ resulting by applying the former control to the system (1) with initial condition x_0. We also denote the variation of the control and the state with respect to their nominal values as $\delta u_i := u_i - \bar{u}_i$ and $\delta x_i := x_i - \bar{x}_i$ respectively. With these definitions, we approximate (5) with its Taylor's series expansion truncated at the second order:

$$\begin{aligned} \mathcal{V}_i^{(k)}(x_i, u_i) \approx \ & \mathcal{V}_i^{(k)}(\bar{x}_i, \bar{u}_i) + Q_{x,i}^{(k)\top} \delta x_i + Q_{u,i}^{(k)\top} \delta u_i \\ & + \frac{1}{2} \left(\delta x_i^\top Q_{xx,i}^{(k)} \delta x_i + \delta u_i^\top Q_{uu,i}^{(k)} \delta u_i \right) \\ & + \delta x_i^\top Q_{xu,i}^{(k)} \delta u_i \ , \end{aligned} \tag{6}$$

where the coefficients are defined as:

$$\begin{aligned}
Q_{x,i}^{(k)} &= \partial_x \phi^{(k)} + \partial_x f^\top \partial_x V_{i+1}^{(k)} \\
Q_{u,i}^{(k)} &= \partial_u \phi^{(k)} + \partial_u f^\top \partial_x V_{i+1}^{(k)} \\
Q_{xx,i}^{(k)} &= \partial_{xx} \phi^{(k)} + \partial_x f^\top \partial_{xx} V_{i+1}^{(k)} \partial_x f + \partial_x V_{i+1}^{(k)} \partial_{xx} f \\
Q_{uu,i}^{(k)} &= \partial_{uu} \phi^{(k)} + \partial_u f^\top \partial_{xx} V_{i+1}^{(k)} \partial_u f + \partial_x V_{i+1}^{(k)} \partial_{uu} f \\
Q_{xu,i}^{(k)} &= \partial_{xu} \phi^{(k)} + \partial_x f^\top \partial_{xx} V_{i+1}^{(k)} \partial_u f + \partial_x V_{i+1}^{(k)} \partial_{xu} f.
\end{aligned} \tag{7}$$

All the derivatives are computed for $x_i = \bar{x}_i$ and $u_i = \bar{u}_i$. To speed-up the algorithm, we neglect the terms depending on the second-order derivatives of the dynamics in the last three equations. This is the same approximation that distinguishes the iterative LQR algorithm [15], [16] from differential dynamic programming [17]. In most scenarios and applications there is a trade-off between computational burden and accuracy of the solution [16]. Because our final goal is the real-time control of robots, we prefer fast-and-approximate solutions over slow-and-accurate ones.

Now that we have a quadratic model of $\mathcal{V}_i^{(k)}(x_i, u_i)$, we can find the optimal control variation analytically:

$$\delta u_i^{(k)*} = \underbrace{-Q_{uu,i}^{(k)\dagger} Q_{u,i}^{(k)}}_{\delta \hat{u}_i^{(k)}} \underbrace{- Q_{uu,i}^{(k)\dagger} Q_{ux,i}^{(k)}}_{K_i^{(k)}} \delta x_i + N_i^{(k)} \delta u_i^{(k+1)} \tag{8}$$

where $N_i^{(k)}$ is a projector onto $\mathcal{N}(Q_{uu,i}^{(k)})$ (i.e. the null space of $Q_{uu,i}^{(k)}$) and $\delta u_i^{(k+1)}$ is a free vector that will be chosen to minimize the task $k+1$. Typically in robotics the dimension of a task is much smaller than the number of DoFs of the robot, which ensures the existence of the null space.

1) One task resolution: We start by solving the optimal control problem with only one task. In this simplified scenario the optimal control in (8) consists of the first two terms only, i.e. $\delta u_i^{(k+1)} \equiv 0$ for all i.

To obtain the value function at step i we substitute the control (8) into (6), thus obtaining the following quadratic form for the value function:

$$V_i(\delta x_i) = V_{s,i} + V_{x,i}^\top \delta x_i + \frac{1}{2} \delta x_i^\top V_{xx,i} \delta x_i \ , \tag{9}$$

where the scalar, linear and quadratic terms are defines as:

$$\begin{aligned}
V_{s,i} &= Q_{u,i}^\top \delta \hat{u}_i + \frac{1}{2} \delta \hat{u}_i^\top Q_{uu,i} \delta \hat{u}_i \\
V_{x,i} &= Q_{x,i} - K_i^\top Q_{u,i} - K_i^\top Q_{uu,i} \delta \hat{u}_i + Q_{xu,i} \delta \hat{u}_i \\
V_{xx,i} &= Q_{xx,i} + K_i^\top Q_{uu,i} K_i - 2 Q_{xu,i} K_i
\end{aligned} \tag{10}$$

Equation (9) is solved backward in time starting from $i = N-1$ to 0 and initialized with the quadratic approximation of $\phi_N^{(k)}(x_N)$. As expected, for a single task, the solution of hierarchical differential dynamic programming coincides analytically with the solution of DDP [17].

2) Multiple-task resolution: When we solve the problem for a task $k > 1$ the control variable in no longer free. Indeed $\delta \hat{u}_i^{(j)}$ and $K_i^{(j)}$ have already been chosen for all tasks $j < k$. The control variable must then respect the following form:

$$\begin{aligned}
\delta u_i &= [\delta \hat{u}_i^{(1)} + \cdots + N_i^{(1)} \cdots N_i^{(k-2)} \delta \hat{u}_i^{(k-1)}] \\
&\quad - [K_i^{(1)} + \cdots + N_i^{(1)} \cdots N_i^{(k-2)} K_i^{(k-1)}] \delta x_i \\
&\quad + N_i^{(1)} N_i^{(2)} \cdots N_i^{(k-1)} \delta u_i^{(k)} \\
&:= \delta \bar{\hat{u}}_i^{(k-1)} - \bar{K}_i^{(k-1)} \delta x_i + \bar{N}_i^{(k)} \delta u_i^{(k)} \ .
\end{aligned} \tag{11}$$

Substituting (11) into (6) we get updated values for the coefficients in (7):

$$\begin{aligned}
\bar{Q}_{x,i}^{(k)} &= Q_{x,i}^{(k)} - \bar{K}_i^{(k-1)\top} Q_{u,i}^{(k)} + \bar{Q}_{xu,i}^{(k)} \delta \bar{\hat{u}}_i^{(k-1)} \\
\bar{Q}_{u,i}^{(k)} &= Q_{u,i}^{(k)} + Q_{uu,i}^{(k)} \delta \bar{\hat{u}}_i^{(k-1)} \\
\bar{Q}_{xx,i}^{(k)} &= Q_{xx,i}^{(k)} + \bar{K}_i^{(k-1)\top} Q_{uu,i}^{(k)} \bar{K}_i^{(k-1)} - 2 Q_{xu,i}^{(k)} \bar{K}_i^{(k-1)} \\
\bar{Q}_{uu,i}^{(k)} &= \bar{N}_i^{(k-1)} Q_{uu,i}^{(k)} \bar{N}_i^{(k-1)} \\
\bar{Q}_{xu,i}^{(k)} &= Q_{xu,i}^{(k)} - \bar{K}_i^{(k-1)\top} Q_{uu,i}^{(k)} \ .
\end{aligned} \tag{12}$$

With the above redefinition we can compute the optimal control $\delta u_i^{(k)*}$ using (8) with the substitution $Q \leftarrow \bar{Q}$. Similarly, the value function at step i has the same form as the one in (9) provided that its coefficients in (10) are computed with $\bar{Q}$.

Equations (12) and (11) already outline the correct procedure to compute the control and the value-function update during the backward pass. It is clear that two sweeps on the task hierarchy are required: 1) from $k = 1$ to K to compute the feedforward terms, the feedback gain matrices and the null-space projectors; 2) from $k = K-1$ to 1 to update the value function. Algorithm 1 summarizes the procedure.

Algorithm 1 Hierarchical Linear Solver

```
Initialize V_{x,N}^{(k)} and V_{xx,N}^{(k)} ∀ task k
for i = N − 1 to 0 do
    for k = 1 to K do
        Update δû̄_i^{(k−1)}, K̄_i^{(k−1)} and N̄_i^{(k−1)}
        Compute Q_{x,i}^{(k)}, Q_{u,i}^{(k)}, Q_{xx,i}^{(k)}, Q_{uu,i}^{(k)} and Q_{xu,i}^{(k)}
        Compute Q̄_{x,i}^{(k)}, Q̄_{u,i}^{(k)}, Q̄_{xx,i}^{(k)}, Q̄_{uu,i}^{(k)} and Q̄_{xu,i}^{(k)}
        δû_i^{(k)} = −Q̄_{uu,i}^{(k)†} Q̄_{u,i}^{(k)}
        K_i^{(k)} = Q̄_{uu,i}^{(k)†} Q̄_{ux,i}^{(k)}
        N_i^{(k)} ← projector onto N(Q̄_{uu,i}^{(k)})
    end for
    for k = K − 1 to 1 do
        Compute V_{x,i}^{(k)} and V_{xx,i}^{(k)}
    end for
    δû_i = δû̄_i^{(1)}, K_i = K̄_i^{(1)}
end for
return {δû_i, K_i, i = 0, ..., N − 1}
```

C. Computational Cost Analysis

The benefit of the proposed hierarchical dynamic programming approach with respect to the algorithm described in [12] is its computational complexity. We now analyze and compare the computational cost of the two algorithms.

1) HDDP computational complexity analysis: We start by considering a single iteration of the linear part of the algorithm. At each iteration we compute the coefficients $\bar{Q}^{(k)}_{(\cdot)}$. This step consists of a series of matrix-matrix multiplications and matrix-vector multiplications ($O(r^3)$, where r is the largest dimension of the matrices involved).To compute the pseudoinverse and the null-space projector we have to perform an orthogonal decomposition of the matrix $\bar{Q}^{(k)}_{uu}$, e.g. we chose the SVD decomposition. Depending on the algorithm the cost can vary, but it is proportional to $O(m^3)$. The value-function update step consists of matrix-matrix and matrix-vector multiplications too. Its cost is still in the order of $O(m^3)$. To summarize, the above computations are performed for each of the K tasks and for all the N time steps thus yielding the total cost of $O(N\ K\ m^3)$.

2) Comparison with POC computational complexity: POC did not take advantage of the intrinsic sparsity of the optimal control problem. Indeed at every iteration a matrix $S \in \mathbb{R}^{mN \times mN}$ is formed and then decomposed with the SVD algorithm. Its computational cost amounts at $O(N^3\ m^3)$. The total algorithmic cost is dominated by the above decomposition. We can thus neglect the cost due to matrix multiplications. The total complexity of the linear part of the algorithm is $O(N^3\ K\ m^3)$.

Comparing the two costs the difference in complexity is clear: POC is cubic in the number of time steps while our algorithm is linear.

IV. HDDP ALGORITHM: NONLINEAR HEURISTIC

A. Regularization Procedure

The procedure described in the previous section operates on a local approximation of the original problem (3) and in this context regularization helps attenuating numerical issues arising from the finite-precision arithmetic and the local validity of the approximated model.

We introduce two regularization procedures. The first one, which resembles the Levenberg modification for the nonlinear least-squares problem, penalizes state deviations from the nominal state trajectory. A parameter $\lambda^{(k)}$ regulates the intensity of the regularization, which is applied in (7) to the definitions of $Q^{(k)}_{uu,i}$ and $Q^{(k)}_{xu,i}$:

$$
\begin{aligned}
Q^{(k)}_{uu,i} &= \partial_{uu}\phi^{(k)} + \partial_u f^\top (\partial_{xx} V^{(k)}_{i+1} + \lambda^{(k)} I_n) \partial_u f \\
Q^{(k)}_{xu,i} &= \partial_{xu}\phi^{(k)} + \partial_x f^\top (\partial_{xx} V^{(k)}_{i+1} + \lambda^{(k)} I_n) \partial_u f \ ,
\end{aligned}
$$

where $I_n \in \mathbb{R}^{n \times n}$ is the identity matrix.

The second regularization consists in damping the pseudoinverses [18] with a parameter $\mu^{(k)}$. Note that the null-space projectors must be computed with the undamped pseudoinverses to ensure the proper hierarchy propagation.

B. Line-Search Procedure

Another important feature of the proposed algorithm consists in adopting a custom line-search procedure, which reduces the optimization step computed on the quadratic approximation. More in details, the control-policy update rule (8) consists in a feedback $K_i(x_i - \bar{x}_i)$ and a feed forward $\delta\hat{u}^{(k)}_i$ term. The latter in particular, might significantly degrade the policy optimality and should therefore be carefully chosen by a suitable line-search procedure, complicated by the hierarchical nature of the algorithm.

The line search adopted in our algorithm is the following: a set of constants $\nu^{(k)} \in [0, 1]$, one for each task, is chosen and the control:

$$
u_i = \bar{u}_i + \sum_{k=1}^{K} \nu^{(k)} \delta\hat{u}^{(k)}_i - K_i(x_i - \bar{x}_i)
$$

is applied to the system (1). The policy for selecting the step size $\nu^{(k)}$ consists in initializing $\nu^{(k)} = 0$ for all the tasks. We then start from the highest-priority task traversing the whole hierarchy down to the last task. At a generic task k we set $\nu^{(k)} = 1$ and we progressively decrease it until all the costs decrease in a lexicographic order, i.e. a decrease of the cost for the task k must not lead to an increase of the cost of any tasks $j < k$.

C. Algorithm Summary

We now summarize the algorithm proposed to solve a cascade of hierarchical optimal control problems. An iteration of the main algorithm is composed of the following phases:

1) Problem approximation.
2) Local control computation, or backward pass.
3) System simulation, or forward pass.

Starting from an initial nominal trajectory we approximate the system and the costs and we compute a control modification. In the forward pass, we simulate the system and compute the new cost for every task. In this phase we perform the line-search procedure. In case of no improvements, we increase the regularization parameters and repeat the backward pass. If at least one task has improved, we decrease its regularization parameter and accept the iteration.

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.

Algorithm 2 summarizes in pseudo code the hierarchical dynamic programming algorithm.

V. SIMULATION RESULTS

This section presents two sets of simulations. The first tests validate the expected computational improvements of our new formulation with respect to POC [12]. The second tests instead investigate the issues arising when using weights in iLQR to approximate strict priorities.

A. Experimental Setup

All tests have been conducted on a workstation with an Intel Xeon quad core at 3.2GHz with 8GB of RAM. We tested the three algorithms on a customized version of the Compliant huManoid (CoMan) simulator [19]. The base of the robot was fixed because we used only its upper body, which counts 11 DoFs: 4 in each arm and 3 in the torso.

Algorithm 2 Hierarchical Differential Dynamic Programming

Require: Given x_0 and $\bar{U}$
Ensure: $\bar{U}, K_i^*$
  $K_i^* \leftarrow 0$
  $\bar{X} \leftarrow$ FORDWARDDYNAMICS$(x_0, \bar{U}, K^*)$
  $G^{(k)} \leftarrow$ COMPUTECOST$(\bar{X}, \bar{U})$
  Initialize regularization parameters: $\lambda \leftarrow \lambda_0$, $\mu \leftarrow \mu_0$
  loop
    for $k = 1, \cdots, K$ and $i = 0, \cdots, N$ **do**
      Compute $\partial_{(\cdot)}\phi^{(k)}$, $\partial_x f$ and $\partial_u f$
    end for
    $\{\delta\hat{u}_i^{(k)}, K_i\} \leftarrow$ LINEARSOLVER$(\partial_{(\cdot)}\{\phi^{(k)}, f\}, \mu, \lambda)$
    $\{U_{\text{new}}, G_{\text{new}}^{(k)}\} \leftarrow$ LINESEARCH$(\delta\hat{u}_i^{(k)}, K_i)$
    if all$(G_{\text{new}}^{(k)} - G^{(k)} > 0)$ **then**
      increase λ and μ
      goto backward pass
    else
      decrease $\lambda^{(k)}$ and $\mu^{(k)}$ for improved tasks
    end if
    $\bar{U} \leftarrow U_{\text{new}}$, $G^{(k)} \leftarrow G_{\text{new}}^{(k)}$, $K_i^* \leftarrow K_i$
    if converged **then**
      break
    end if
  end loop
  return $\{\bar{U}, K_i^*\}$

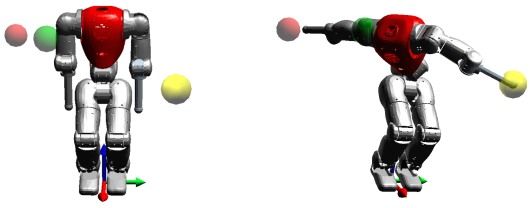

Fig. 1: Screenshots of the simulation. On the left the initial configuration of the robot. On the right the final configuration during one test. Colored balls represent the target. In order of priority: yellow, red and green.

All the algorithms and the dynamics computation have been coded in Matlab 2012b.

B. Test Description

We controlled the humanoid robot in order to reach three cartesian points with three different parts of its body, while minimizing the effort (i.e. the joint torques). In order of decreasing priority, we controlled the position of the left-arm end-effector, right-arm end-effector and the top of the torso. Figure 1 shows the test scenario.

We discretized the robot dynamics with a time step of 5ms for a total of 50 steps. In all the tests, we initialized the control trajectory with gravity compensation torques, so that the robot maintained the initial state for the whole time horizon.

TABLE I: Comparison between POC and HDDP. Task errors are in $[\text{m}^2]$. Values are average of 100 trials.

		Error $[\text{m}^2]$			Iter.	CPU
		Task 1	Task 2	Task 3		Time
Test 1	POC	$4.07\ 10^{-7}$	$1.59\ 10^{-2}$	N.A.	13	139s
	HDDP	$1.87\ 10^{-8}$	$1.22\ 10^{-2}$	N.A.	411	107s
Test 2	POC	$5.03\ 10^{-7}$	$1.39\ 10^{-2}$	$2.87\ 10^{-2}$	28	511s
	HDDP	$1.42\ 10^{-8}$	$1.24\ 10^{-2}$	$2.24\ 10^{-2}$	382	136s
Test 3	POC	$6.73\ 10^{-7}$	$1.44\ 10^{-5}$	N.A.	21	206s
	HDDP	$4.75\ 10^{-9}$	$3.98\ 10^{-9}$	N.A.	281	187s

C. Comparison with Prioritized Optimal Control

We first tested our algorithm against our previous implementation described in [12]. The reaching cost functions contain only a final cost, i.e. it takes the form $G^{(k)}(X, U) \equiv \phi_N^{(k)}(x_N)$, being the squared norm of the distance between the end-effector and the target position:

$$\phi_N^{(k)}(x_N) = ||h(x_N) - \bar{h}||^2 \ ,$$

where $h(x) : \mathbb{R}^n \mapsto \mathbb{R}^3$ is the function mapping the state variable x to a cartesian position, and $\bar{h}$ is its reference value.

We tested three different scenarios: i) two conflicting reaching tasks for the left and right arm end-effectors (Test 1); ii) three conflicting reaching tasks, using both arms and the torso (Test 2); iii) two compatible reaching tasks for the left and right arm end-effectors (Test 3). In all tests we added an effort task at the bottom of the hierarchy, which *removes* any left redundancy. We repeated the tests 100 times changing the original task targets of a value randomly sampled from a uniform distribution between 0 and 5cm. Table I reports the mean value of the final costs and of the total computation time for the three scenarios.

For both algorithms the absolute and relative tolerance for the stopping criteria have been set to $10^{-8}[\text{m}^2]$ and 10^{-4}, respectively.

Table I shows that, as far as the final error is concerned, HDDP manages to achieve a slightly better performance in all three scenarios. Unexpectedly, in Test 3, in which the tasks are compatible, POC yields some errors, especially in the secondary task, while HDDP manages to achieve almost zero errors.

From the computational standpoint HDDP is faster then POC in every test as we expected. Note that the CPU time is not directly proportional to the number of iterations in the HDDP algorithm. This is mainly due to the line-search procedure: big values of $\nu^{(k)}$ mean few forward-dynamics simulations, resulting in faster iterations.

D. Weighted Cost Comparison

This test compares our algorithm with the standard iLQR control in the case of three conflicting reaching tasks (Test 2). To approximate strict priorities we used a weighted sum of the task costs:

$$G(X, U) = w^3 g_1(X, U) + w^2 g_2(X, U) + w g_3(X, U) + g_e(U),$$

where $g_i(\cdot)$ is the i^{th}-task cost function, $g_e(U)$ is the effort task and w is a user-defined weight. Table II shows the results

TABLE II: Task errors ($[\mathrm{m}^2]$) for the three-reaching scenario using a single weighted cost.

	w	Error $[\mathrm{m}^2]$ Task 1	Task 2	Task 3
Weighted HDDP	10	$7.27\ 10^{-4}$	$2.03\ 10^{-2}$	$7.51\ 10^{-2}$
	10^2	$1.38\ 10^{-6}$	$1.43\ 10^{-2}$	$2.86\ 10^{-2}$
	10^3	$1.14\ 10^{-8}$	$1.44\ 10^{-2}$	$2.37\ 10^{-2}$
	10^4	$6.64\ 10^{-10}$	$1.44\ 10^{-2}$	$2.37\ 10^{-2}$
	10^5	$6.64\ 10^{-10}$	$1.44\ 10^{-2}$	$2.37\ 10^{-2}$
	10^6	$4.1\ 10^{-4}$	$1.44\ 10^{-2}$	$2.41\ 10^{-2}$
iLQR	10	$7.2\ 10^{-4}$	$2.0\ 10^{-2}$	$7.5\ 10^{-2}$
	10^2	$1.3\ 10^{-6}$	$1.43\ 10^{-2}$	$2.78\ 10^{-2}$
	10^3	$9.5\ 10^{-9}$	$1.44\ 10^{-2}$	$2.36\ 10^{-2}$
	10^4	$7.56\ 10^{-6}$	$1.47\ 10^{-2}$	$2.32\ 10^{-2}$
	10^5	$3.49\ 10^{-6}$	$1.46\ 10^{-2}$	$2.3\ 10^{-2}$
	10^6	$2.37\ 10^{-3}$	$1.33\ 10^{-2}$	$3.33\ 10^{-2}$

for different weights from $w = 10$ to $w = 10^6$. We used two different implementations to check that the results did not depend on some implementation details: i) *Weighted HDDP* is our algorithm with a single weighted task; ii) *iLQR* is an implementation of [15]. In both cases the cost has been used both as running cost and as final cost.

For the first two weights, i.e. $w = 10$ and $w = 10^2$, both implementations achieved the same result: the task priorities are not respected and the solutions are suboptimal. By increasing the weights the solutions obtained approach the one from multi-task HDDP. Interestingly, from $w = 10^4$ the two implementations yield significantly different results. In both cases increasing the weights over a certain value leads to larger errors.

We can conclude that with the appropriate weights iLQR can find a solution that is very similar to the one of HDDP. However, if the weights are not appropriate iLQR can give either suboptimal or meaningless solutions. Moreover, we believe that the complexity of the weight tuning is proportional to the number of tasks, which could easily be higher than 4 when controlling a full humanoid.

VI. Conclusions and future works

In this paper we proposed an efficient way to compute the solution of the prioritized optimal control (POC) problem that we previously introduced in [12]. We reformulated the original POC method, and solved the problem by means of a modified version of dynamic programming, capable of handling strict task priorities. The resulting algorithm is significantly faster than its previous version, while retaining the capability to correctly handle strict priorities.

Our final goal is to apply this method to a real humanoid robot, but before doing so we need to address some issues. Taking into account torques and joint limits is crucial, and it can be accomplished by introducing inequalities constraints into the problem. Since humanoid robots are almost always in contact with the environment, the controller must consider the contact dynamics. Thanks to the generality of our formulation, it should be easy to integrate any smooth contact dynamics [20] inside our algorithm. Finally, the current implementation has been coded in Matlab, which allowed only "relative" comparisons. To properly measure the computational load of the algorithm, an efficient, multi-threaded C++ implementation is required.

As humanoid robots become more and more complex, the ability to specify simple and clear tasks through strict priorities — thus avoiding the time-consuming and error-prone weight tuning — could greatly simplify the synthesis of complex behavior for these systems.

References

[1] Y. Nakamura, H. Hanafusa, and T. Yoshikawa, "Task-Priority Based Redundancy Control of Robot Manipulators," *The International Journal of Robotics Research*, vol. 6, no. 2, pp. 3–15, June 1987.

[2] B. Siciliano and J.-J. Slotine, "A general framework for managing multiple tasks in highly redundant robotic systems," in *IEEE Fifth International Conference on Advanced Robotics, 1991.*, June 1991, pp. 1211–1216 vol.2.

[3] O. Khatib, "A unified approach for motion and force control of robot manipulators: The operational space formulation," *IEEE Journal on Robotics and Automation*, vol. 3, no. 1, pp. 43–53, Feb. 1987.

[4] L. Sentis, "Synthesis and control of whole-body behaviors in humanoid systems," Ph.D. dissertation, Stanford University, 2007.

[5] L. Righetti, J. Buchli, M. Mistry, and S. Schaal, "Inverse dynamics control of floating-base robots with external constraints: A unified view," *IEEE International Conference on Robotics and Automation*, May 2011.

[6] M. Mistry and L. Righetti, "Operational Space Control of Constrained and Underactuated Systems," in *Robotics: Science and Systems*, June 2011.

[7] L. Saab, N. Mansard, F. Keith, J.-Y. Fourquet, and P. Soueres, "Generation of dynamic motion for anthropomorphic systems under prioritized equality and inequality constraints," *IEEE International Conference on Robotics and Automation*, pp. 1091–1096, May 2011.

[8] A. Escande, N. Mansard, and P.-B. Wieber, "Hierarchical quadratic programming: Fast online humanoid-robot motion generation," *The International Journal of Robotics Research*, vol. 33, no. 7, pp. pp. 1006–1028, Jun 2014.

[9] M. De Lasa and A. Hertzmann, "Prioritized optimization for task-space control," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, Oct. 2009, pp. 5755–5762.

[10] M. De Lasa, I. Mordatch, and A. Hertzmann, "Feature-based locomotion controllers," *ACM Transactions on Graphics*, vol. 29, no. 4, July 2010.

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

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

[13] O. Kanoun, F. Lamiraux, and P.-B. Wieber, "Kinematic control of redundant manipulators: Generalizing the Task-Priority framework to inequality task," *IEEE Transactions on Robotics*, vol. 27, no. 4, 2011.

[14] R. Bellman and S. E. Dreyfus, "Applied dynamic programming," 1962.

[15] E. Todorov and W. Li, "A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems," in *American Control Conference, 2005. Proceedings of the 2005*, June 2005, pp. 300–306 vol. 1.

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

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

[18] A. Maciejewski, "Dealing with the ill-conditioned equations of motion for articulated figures," *Computer Graphics and Applications, IEEE*, vol. 10, no. 3, pp. 63–71, May 1990.

[19] H. Dallali, M. Mosadeghzad, G. Medrano-Cerda, N. Docquier, P. Kormushev, N. Tsagarakis, Z. Li, and D. Caldwell, "Development of a dynamic simulator for a compliant humanoid robot based on a symbolic multibody approach," in *Mechatronics (ICM), 2013 IEEE International Conference on*, Feb 2013, pp. 598–603.

[20] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, Oct 2012, pp. 5026–5033.