2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
Congress Center Hamburg
Sept 28 - Oct 2, 2015. Hamburg, Germany

Unilateral constraints in the Reverse Priority redundancy resolution method

Fabrizio Flacco Alessandro De Luca

Abstract— **Our recently developed Reverse Priority (RP) redundancy resolution method is extended here to the presence of unilateral constraints. The RP method computes the solution to a stack of prioritized tasks starting from the lowest priority one, and adding iteratively the contributions of higher priority tasks. In this framework, unilateral constraints can be added efficiently, while guaranteeing also continuity of joint velocity commands. Since unilateral (hard) constraints are typically placed at the highest priority levels, their treatment within the RP method leads to the least possible modification of the solution computed so far, when analyzing the need to activate or not these constraints. The effectiveness of the approach is shown by simulations on a planar 6R robot and on a humanoid robot, as well as experiments on a KUKA LWR manipulator.**

I. Introduction

One of the most appealing features of robots with many degree of freedom, such as humanoids, is the possibility of executing multiple tasks at the same time [1]. To achieve such result, redundancy is usually resolved at a differential kinematic level [2], [3]. In this framework, a task is often described as a reference trajectory for a geometric component of the robot (the Cartesian pose of the end-effector, the position of a control point on the robot body, a joint angle, etc.). At each time instant, a desired value for this component is specified in the form of a bilateral (equality) constraint.

Such task description is very effective, but sometimes is over-constraining or does not reflect properly the desired robot operation. In fact, in many applications it is not strictly necessary to assign a particular value to some given robot component, but it would be preferable to specify a region to which this component should belong. This requirement can be coded with unilateral (inequality) constraints. Examples of unilateral constraints include structural restrictions on robot motion capabilities, such as joint limits or bounds on maximum velocities or accelerations [4]–[6], the presence of Cartesian obstacles to be avoided [7], [8], the definition of virtual fixtures [9], and the need of keeping some features in the field of view of a camera mounted on the robot [10] or the projection of the center of mass of a humanoid within the support polygon [11].

There are three main approaches to deal with the problem of unilateral constraints in redundant robots. The first one takes an optimization point of view. Unilateral constraints on robot components are transformed into linear inequality constraints at the control differential level, and an optimal solution is obtained by solving a constrained quadratic programming (QP) problem. The drawback of this general approach is that computational costs, when using state-of-the-art QP solvers based on active sets, becomes too large for real-time robot applications as soon as the number of constraints increases [12]. A significant reduction of the computational cost has been obtained with the hierarchical method presented in [13], but its algorithmic implementation is quite complex.

A second approach is based on transforming a unilateral constraint on a robot component into a bilateral (task) constraint at the control differential level [14], [15]. Such task constraint has to be activated only when needed, and a smooth evolution of the joint commands during the activation phase has to be guaranteed. However, the obtained solutions are typically suboptimal, in the sense that constraints may end up being activated even if this was not necessary for accomplishing the desired task. On the other hand, implementations are in general simpler and less computationally demanding than with true active set optimization methods.

Finally, in a third approach the null space of the equality tasks is used for taking into account unilateral constraints. Many such methods have been proposed, mostly based on a projected gradient formulation [16]–[18], or resorting to weighted pseudoinverses [19]. These methods are very common because of their simple implementation. The drawback is that they do not guarantee the actual satisfaction of unilateral constraints, which are typically violated as soon as in contrast with the fulfillment of the equality tasks.

In this paper, we introduce the use of unilateral constraints in the Reverse Priority (RP) redundancy resolution method [20]. The resulting method may be classified in the second category above, since unilateral constraints fulfillment is always guaranteed, but in general only suboptimal solutions are obtained, We exploit here the core feature of the RP method, namely that the contribution of high priority tasks is added only *after* having computed the solution for the lower priority tasks. Since a unilateral constraint is typically placed at the highest priority levels, the reverse order allows checking if such a constraint needs to be activated or not, with the least possible modification of the solution computed so far for the lower priority tasks. The method can indeed insert unilateral constraints at any priority level. Moreover, continuity of the joint velocity commands during activation/deactivation of constraints is guaranteed by minimizing the variation of the joint velocities in discrete time, as proposed in [21].

The authors are with the Dipartimento di Ingegneria Informatica, Automatica e Gestionale, Sapienza Università di Roma, Via Ariosto 25, 00185 Roma, Italy ({fflacco,deluca}@diag.uniroma1.it). This work is supported by the European Commission, within the FP7 ICT-287513 SAPHARI project (www.saphari.eu).

The paper is organized as follows. Background notions are recalled in Sec. II. Section III introduces a characterization of unilateral constraints, while the integration of these constraints in the RP method is presented in Sec. IV. Simulation results on a 6R planar robot and on a humanoid robot, and experimental results on a KUKA LWR manipulator are reported in Sec. V and Sec. VI, respectively. All these results are shown also in the video clip accompanying the paper.

II. Background

Let $\boldsymbol{q}_k \in \mathbb{R}^n$ be the generalized (joint) coordinates of a n-dof robot at time instant $t = kT$, with $k \in \mathbb{N}^+$ and $T > 0$ being the sampling time. The vector $\boldsymbol{x}_k = \boldsymbol{f}(\boldsymbol{q}_k) \in \mathbb{R}^m$ describes a generic m-dimensional task, with $m \leq n$, and $\boldsymbol{J}_k = \boldsymbol{J}(\boldsymbol{q}_k) = (\partial \boldsymbol{f}/\partial \boldsymbol{q})\,|_{\boldsymbol{q}=\boldsymbol{q}_k}$ is the associated $m \times n$ task Jacobian matrix. At a given robot configuration $\boldsymbol{q}_k$, the differential kinematics of the task is

$$\dot{\boldsymbol{x}}_k = \boldsymbol{J}_k \dot{\boldsymbol{q}}_k. \tag{1}$$

Inversion of the differential map (1) provides infinitely many solutions, all of which can be generated as

$$\dot{\boldsymbol{q}}_k = \boldsymbol{J}_k^{\#} \dot{\boldsymbol{x}}_k + \boldsymbol{P}_k \dot{\boldsymbol{q}}_{\mathcal{N},k}, \tag{2}$$

where $\boldsymbol{J}_k^{\#}$ is the (unique) Moore-Penrose pseudoinverse of the task Jacobian [22], and

$$\boldsymbol{P}_k = \boldsymbol{I} - \boldsymbol{J}_k^{\#} \boldsymbol{J}_k \tag{3}$$

is the $n \times n$ orthogonal projector in the Jacobian null space. In (3), $\boldsymbol{I}$ the $n \times n$ identity matrix and $\dot{\boldsymbol{q}}_{\mathcal{N},k} \in \mathbb{R}^n$ is a generic joint velocity. Equation (2) gives the joint velocity $\dot{\boldsymbol{q}}_k$ that satisfies (1) in a least squares sense while minimizing the distance to $\dot{\boldsymbol{q}}_{\mathcal{N},k}$, or

$$\begin{aligned} \dot{\boldsymbol{q}}_k &= \arg\min_{\dot{\boldsymbol{q}} \in \mathcal{S}_k} \frac{1}{2} \|\dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}_{\mathcal{N},k}\|^2 \\ \text{with } \mathcal{S}_k &= \left\{ \arg\min_{\dot{\boldsymbol{q}} \in \mathbb{R}^n} \frac{1}{2} \|\boldsymbol{J}_k \dot{\boldsymbol{q}} - \dot{\boldsymbol{x}}_k\|^2 \right\}. \end{aligned} \tag{4}$$

Thus, setting $\dot{\boldsymbol{q}}_{\mathcal{N},k} = \boldsymbol{0}$ provides the joint velocity solution with minimum norm. When the task is feasible, the *task manifold* $\mathcal{S}_k$ contains the joint velocity subspace of solutions to (1).

The pseudoinverse of a $m \times n$ matrix $\boldsymbol{A}$ can be obtained from its Singular Value Decomposition (SVD) as

$$\boldsymbol{A}^{\#} = \boldsymbol{V} \boldsymbol{\Sigma}^{\#} \boldsymbol{U}^T = \sum_{i=1}^{\rho} \frac{1}{\sigma_i} \boldsymbol{v}_i \boldsymbol{u}_i^T, \tag{5}$$

where $\boldsymbol{V}$ is a $n \times n$ unitary matrix composed by column vectors $\boldsymbol{v}_i$, $\boldsymbol{U}$ is a $m \times m$ unitary matrix composed by column vectors $\boldsymbol{u}_i$, $\boldsymbol{\Sigma}$ is a $m \times n$ matrix containing a diagonal block with the singular values σ_i, $i = 1, \ldots, m$, of $\boldsymbol{A}$ in decreasing order ($\sigma_i \geq \sigma_j$ if $i < j$), and ρ is the rank of $\boldsymbol{A}$.

When the matrix $\boldsymbol{A}$ is singular, or close to a singularity, the pseudoinversion becomes numerically ill-conditioned. This drawback is mitigated by the use of the so-called damped pseudoinverse $\boldsymbol{A}^{\#_d}$, in which $1/\sigma_i$ in (5) is replaced by the approximation

$$\frac{1}{\sigma_i} \approx \frac{\sigma_i}{\sigma_i^2 + \mu^2}, \tag{6}$$

where $\mu \geq 0$ is a damping factor. The higher is μ, the more damped is the joint velocity near a singularity and the larger is the error in executing the desired task. There are different ways to select the damping factor [23]. In this paper, we use

$$\mu^2 = \begin{cases} 0 & \text{when } \sigma_m \geq \epsilon \\ \left(1 - (\sigma_m/\epsilon)^2\right) \mu_{max}^2 & \text{otherwise,} \end{cases} \tag{7}$$

applying the damping action only when the smallest singular value σ_m becomes smaller than a parameter $\epsilon > 0$. The second design parameter μ_{max} is the value of the damping factor at the singularity. We remark that, for the projector in the null space of $\boldsymbol{A}$, it is in general $\boldsymbol{P}_A \neq \boldsymbol{I} - \boldsymbol{A}^{\#_d} \boldsymbol{A}$. Instead, $\boldsymbol{P}_A$ must be computed as

$$\boldsymbol{P}_A = \boldsymbol{I} - \sum_{i=1}^{\rho} \boldsymbol{v}_i \boldsymbol{v}_i^T. \tag{8}$$

In the rest of the paper we shall refer for simplicity to the undamped pseudoinverse (5) and to the null space projector (3). However, in practice the damped pseudoinverse (6–7) and the null space projector (8) have been used.

A. Minimum variation of the joint velocity command

In [21], we have shown how to specify a joint velocity command in discrete time so as to obtain the minimization of the joint acceleration norm. This is obtained by taking the joint velocity at a given time step that solves the task (1) in a least squares sense while minimizing the distance to the previous velocity command, i.e., $\dot{\boldsymbol{q}}_{\mathcal{N},k} = \dot{\boldsymbol{q}}_{k-1}$ in the general first-order solution (2):

$$\dot{\boldsymbol{q}}_k = \boldsymbol{J}_k^{\#} \dot{\boldsymbol{x}}_k + \boldsymbol{P}_k\, \dot{\boldsymbol{q}}_{k-1}. \tag{9}$$

When a redundant robot is controlled using minimum norm commands at the second- or higher-order differential level (i.e., from acceleration upwards), one can typically observe the joints starting to float over time on the self-motion manifold. To remove this undesired behavior, the simplest solution is to introduce a damping action on the null-space motion [3]. This can be obtained with the discrete-time joint velocity control law

$$\dot{\boldsymbol{q}}_k = \boldsymbol{J}_k^{\#} \dot{\boldsymbol{x}}_k + \lambda \boldsymbol{P}_k \dot{\boldsymbol{q}}_{k-1}. \quad \text{for } \lambda \in [0, 1] \tag{10}$$

For further details, we refer to [21] .

B. Multiple tasks using the Reverse Priority approach

Consider a stack of l tasks

$$\dot{\boldsymbol{x}}_k^{\{p\}} = \boldsymbol{J}_k^{\{p\}} \dot{\boldsymbol{q}}_k \qquad p = 1, \ldots, l, \tag{11}$$

each of dimension m_p, that are ordered by their priority, i.e., task i has higher priority than task j if $i < j$. The execution of a task of lower priority should not interfere with the execution of tasks having higher priority. In classical solutions (see [1]), this hierarchy is obtained by projecting

the solution for the pth task in the null space of all higher priority tasks.

In the Reverse Priority method for multiple prioritized tasks [20], a solution is computed first for the lowest priority task and then the contributions of higher priority tasks are added recursively. The correct task hierarchy is obtained if the portion of the lower level tasks that is linearly independent from the higher priority task is preserved (not deformed) when adding the contribution of a higher priority task, while the higher priority task predominates any portion that may be in conflict. Such result is obtained thanks to the RP recursive equation (for $p = l-1, \ldots, 1$)

$$\dot{\boldsymbol{q}}_k^{\{p\}} = \dot{\boldsymbol{q}}_k^{\{p+1\}} + \boldsymbol{T}_k^{\{p\}} \left(\boldsymbol{J}_k^{\{p\}} \boldsymbol{T}_k^{\{p\}} \right)^{\#} \left(\dot{\boldsymbol{x}}_k^{\{p\}} - \boldsymbol{J}_k^{\{p\}} \dot{\boldsymbol{q}}_k^{\{p+1\}} \right), \tag{12}$$

where $\dot{\boldsymbol{q}}_k^{\{p\}}$ is the solution that takes into account all lower level tasks up to the one with priority p. Consider the reverse augmented Jacobian $\boldsymbol{J}_{RA,k}^{\{p+1\}}$, bordered on top with $\boldsymbol{J}_k^{\{p\}}$

$$\begin{aligned} \boldsymbol{J}_{RA,k}^{\{p\}} &= \begin{pmatrix} \boldsymbol{J}_k^{\{p\}^T} & \boldsymbol{J}_k^{\{p+1\}^T} & \ldots & \boldsymbol{J}_l^{\{l\}^T} \end{pmatrix}^T \\ &= \begin{pmatrix} \boldsymbol{J}_k^{\{p\}} \\ \boldsymbol{J}_{RA,k}^{\{p+1\}} \end{pmatrix}. \end{aligned} \tag{13}$$

Then, the $n \times m_p$ matrix $\boldsymbol{T}_k^{\{p\}}$ in (12) is given by

$$\boldsymbol{J}_{RA,k}^{\{p\}\#} = \begin{pmatrix} \boldsymbol{T}_k^{\{p\}} & * \end{pmatrix}. \tag{14}$$

We note that $\boldsymbol{T}_k^{\{p\}}$ is the only extra matrix needed, and can be obtained through an efficient update of the previous $\boldsymbol{J}_{RA,k}^{\{p+1\}\#}$ (see, e.g., [22]).

In this framework, an auxiliary joint velocity command $\dot{\boldsymbol{q}}_{\mathcal{N},k}$ that needs to be projected in the null space of all tasks is introduced by setting simply $\dot{\boldsymbol{q}}_k^{\{l+1\}} = \dot{\boldsymbol{q}}_{\mathcal{N},k}$. This auxiliary joint velocity can be used, e.g., to implement a projected gradient method [16], or to obtain the minimum variation of the joint velocity command (10) choosing

$$\dot{\boldsymbol{q}}_k^{\{l+1\}} = \lambda \dot{\boldsymbol{q}}_{k-1}^{\{1\}}. \tag{15}$$

For more details, please refer to [20].

III. Unilateral Constraints

Consider a generic scalar component $x_C(\boldsymbol{q})$ associated to the robot, as described in the introduction. Depending on the application, we may not wish to specify a task for this component (bilateral constraint), but we need so impose a bound on it (unilateral constraint). A geometric limit can be represented by a simple unilateral constraint in the form

$$x_C \leq \bar{x}_C. \tag{16}$$

Let $\dot{x}_{C,d}$ be the nominal reference velocity of such robot component resulting from the redundancy resolution of all the assigned equality tasks. The inequality constraint (16) can be transformed in the differential kinematic map

$$\dot{x}_C = \begin{cases} \text{free to move} & x_C < \bar{x}_C \text{ or } \dot{x}_{C,d} < 0 \\ 0 & x_C \geq \bar{x}_C \text{ and } \dot{x}_{C,d} \geq 0. \end{cases} \tag{17}$$

In this way, the component x_C is allowed only to have a velocity that will not exceed the limit (16), or that will try to recover it (if ever violated). In our approach, we shall represent the rule (17) for a unilateral constraint by means of a special task having a $1 \times n$ Jacobian $\boldsymbol{J}_C(\boldsymbol{q}) = \frac{\partial x_C(\boldsymbol{q})}{\partial \boldsymbol{q}}$ and a desired zero velocity ($\dot{x}_C = 0$), which needs however to be activated only when $x_C \geq \bar{x}_C$ and $\dot{x}_{C,d} \geq 0$.

In many cases, especially when multiple tasks have to be executed, it is important to consider soft and hard unilateral constraints. A unilateral constraint is *soft* with respect to a set of tasks if the constraint will be fulfilled only when not in contrast with the given tasks. It is *hard* when it can never be relaxed. Thus, a task in contrast with a hard unilateral constraint is bound to be deformed. This global behavior is obtained by giving to the unilateral constraint, respectively a lower or a higher priority than the specified set of tasks.

Summarizing, a redundancy resolution method has to handle unilateral constraints so as to *i)* allow their presence at any priority level, *ii)* manage constraint activation and deactivation, and *iii)* guarantee smooth control commands. We will present next how these features can be efficiently obtained using the RP framework.

IV. The Reverse Priority Solution

One of the nice aspects of the RP framework is the possibility of inserting unilateral constraints at high priority levels, and of processing them only after the evaluation of the solution for lower priority tasks.

Assume that the unilateral constraint (16) has the ith priority in the hierarchy, being thus a soft constraint for all tasks j with $j < i$ and a hard constraint for tasks with $j > i$. The nominal reference velocity for this robot component constrained by the lower priority tasks up to level $i+1$ is given by

$$\dot{x}_{C,d} = \boldsymbol{J}_{C,k}\, \dot{\boldsymbol{q}}_k^{\{i+1\}}. \tag{18}$$

This reference velocity can be used to check whether the constraining task associated to (16) has to be activated or not, according to the rule (17). For, introduce the activation variable h_C defined as

$$h_C = \begin{cases} 0 & x_C < \bar{x}_C \text{ or } \dot{x}_{C,d} < 0 \\ 1 & x_C \geq \bar{x}_C \text{ and } \dot{x}_{C,d} \geq 0, \end{cases} \tag{19}$$

namely, $h_C = 1$ when the constraint hass to be considered active and $h_C = 0$ when it is inactive.

The unilateral constraint (16) is then added to the RP method by having

$$\dot{\boldsymbol{q}}_k^{\{i\}} = \dot{\boldsymbol{q}}_k^{\{i+1\}} - h_C\, \boldsymbol{T}_k^{\{i\}} \left(\boldsymbol{J}_{C,k} \boldsymbol{T}_k^{\{i\}} \right)^{\#} \boldsymbol{J}_{C,k}\, \dot{\boldsymbol{q}}_k^{\{i+1\}}, \tag{20}$$

which is obtained from (12) by setting $\boldsymbol{J}_k^{\{i\}} = \boldsymbol{J}_{C,k}$ and $\dot{\boldsymbol{x}}_k^{\{i\}} = \dot{x}_C = 0$. We have thus the chance to check whether the unilateral constraint needs to be activated *after* the computation of the current nominal solution in the stack of tasks. Note also that our algorithm does not require the fictitious introduction of a repulsive action (in the form $\dot{x}_C = -K x_{C,d}$), differently from other existing methods in the same category (see, e.g., [15]).

A. Smooth activation and deactivation

So far we have shown how to insert a unilateral constraint at any level of the hierarchy by using (20), and how to activate and deactivate it properly by using (18–19). However, joint velocity commands will not be continuous (except in very special cases) when the constraint is activated or deactivated. This discontinuity is due to the variation of the task manifold associated to the currently active constraints.

Consider the situation between steps $k-1$ and k in time. In general, if the unilateral constraint is activated between these two steps ($h_{C,k-1}=0$, $h_{C,k}=1$), the previous joint velocity command $\dot{\boldsymbol{q}}_{k-1}$ will not belong to the manifold associated with the new stack of tasks. Thus, the solution will jump to a new value ($\dot{\boldsymbol{q}}^{\mathcal{A}} \rightarrow \dot{\boldsymbol{q}}^{\mathcal{B}}$ or $\dot{\boldsymbol{q}}^{\mathcal{C}} \rightarrow \dot{\boldsymbol{q}}^{\mathcal{D}}$ in Fig. 1). In a similar way, if the constraint is being deactivated ($h_{C,k-1}=1$, $h_{C,k}=0$), the solution will be discontinuous. However, two different possible behaviors may arise in this case: if the constraint is in contrast with lower priority tasks (i.e., the task defined through (17) is linearly dependent on the lower priority tasks), then the manifold associated to the stack of tasks at step k will not contain the previous joint velocity command (the discontinuity shown as $\dot{\boldsymbol{q}}^{\mathcal{D}} \rightarrow \dot{\boldsymbol{q}}^{\mathcal{C}}$ in Fig. 1); on the other hand, if the constraint is linearly independent from the lower priority tasks, then the previous solution belongs also to the new manifold, but not being in general the *optimal* one, the new solution will also jump ($\dot{\boldsymbol{q}}^{\mathcal{B}} \rightarrow \dot{\boldsymbol{q}}^{\mathcal{A}}$ in Fig. 1).

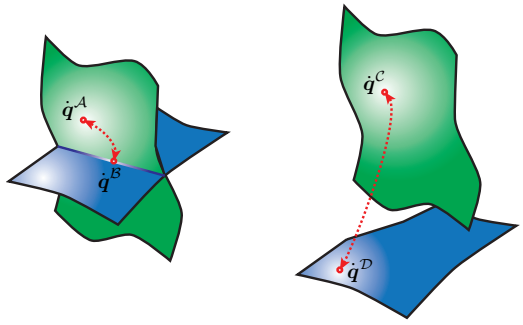

Fig. 1. Illustrative examples of discontinuities due to unilateral constraint activation/deactivation. The velocity commands $\dot{\boldsymbol{q}}^{\mathcal{A}}$ and $\dot{\boldsymbol{q}}^{\mathcal{C}}$ are optimal solutions of the unconstrained case (the task associated to the unilateral constraint is inactive). The velocity commands $\dot{\boldsymbol{q}}^{\mathcal{B}}$ and $\dot{\boldsymbol{q}}^{\mathcal{D}}$ are the optimal solutions when the constraint is active, respectively when it is linearly independent from or linearly dependent on lower priority tasks.

In order to guarantee the continuity of joint velocity commands, two actions have to be taken: first, the solution has to be guided smoothly to the new manifold; then, it should move with continuity inside the manifold. Implementation of the second action is straightforward, thanks to the minimization of the variation of joint velocity commands over successive steps given by the choice (15). For the first action, consider again the illustrative example in Fig. 1, using now the initialization (15) for the RP method. If the solution at the previous step $k-1$ was $\dot{\boldsymbol{q}}^{\mathcal{B}}$ (unilateral constraint active) and the constraint is removed at the current step k, the solution will not jump to $\dot{\boldsymbol{q}}^{\mathcal{A}}$ but will remain by continuity near to $\dot{\boldsymbol{q}}^{\mathcal{B}}$. To guide the solution from the constrained manifold to the unconstrained manifold and vice versa, the activation variable h_C must be smoothly varied. To this end, define an activation buffer $\beta_P > 0$, which acts in proximity of the geometric limit $\bar{x}_C$ in (16), and a deactivation buffer $\beta_V > 0$, which works on the nominal velocity commands. The combination of these two buffers with the activation rule (19) leads to

$$h_{C,P} = \begin{cases} 0 & x_C < \bar{x}_C - \beta_P \\ g\left(\frac{x_C - \bar{x}_C + \beta_P}{\beta_P}\right) & \bar{x}_C - \beta_P \le x_C < \bar{x}_C \\ 1 & \bar{x}_C \le x_C, \end{cases} \tag{21}$$

$$h_{C,V} = \begin{cases} 0 & \dot{x}_{C,d} < -\beta_V \\ g\left(\frac{\dot{x}_{C,d} + \beta_V}{\beta_V}\right) & -\beta_V \le \dot{x}_{C,d} < 0 \\ 1 & 0 \le \dot{x}_{C,d}, \end{cases} \tag{22}$$

$$h_C = h_{C,P} \cdot h_{C,V}. \tag{23}$$

In (21–22), $g(\cdot)$ is a sufficiently smooth *activation* function with boundary conditions $g(0)=0$ and $g(1)=1$. Figure 2 shows a possible shape of this function when using the quintic polynomial

$$g(a) = 6a^5 - 15a^4 + 10a^3. \tag{24}$$

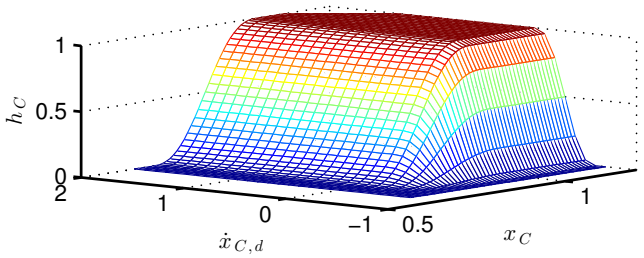

Fig. 2. Activation profile associated to (21–23), with the function (24), $\bar{x}_C = 1$, $\beta_P = 0.3$, and $\beta_V = 0.6$.

V. Simulation Results

To show the effectiveness of the proposed method we present here simulation results for three examples.

In the first two examples, we have considered a planar robot with six revolute joints and having links of unitary length. Simulations were conducted using Matlab. The single primary task ($l=1$) requires the robot end-effector position $\boldsymbol{x}^{\{1\}}$ to go through a sequence of Cartesian points connected by linear paths, and starting from the initial configuration $\boldsymbol{q}_0 = (0,0,0,0,10,-30)$ [deg]. The desired trajectory between two generic Cartesian points $\boldsymbol{X}_A \rightarrow \boldsymbol{X}_B$ starts at time t_A and is described by

$$\begin{aligned} \boldsymbol{X}(t) &= \boldsymbol{X}_A + (\boldsymbol{X}_B - \boldsymbol{X}_A)\,\gamma(t) \\ \gamma(t) &= 6c^5 - 15c^4 + 10c^3 \\ c &= \frac{t - t_A}{T_{AB}} \\ \boldsymbol{v}(t) &= \dot{\boldsymbol{X}}(t) = \frac{\boldsymbol{X}_B - \boldsymbol{X}_A}{T_{AB}} \left(30c^4 - 60c^3 + 30c^2\right), \end{aligned} \tag{25}$$

where $T_{AB} > 0$ is the nominal time to perform the rest-to-rest trajectory. Accordingly, the desired task velocity is defined as

$$\dot{\boldsymbol{x}}^{\{1\}} = \boldsymbol{v}(t) + K_P \left(\boldsymbol{X}(t) - \boldsymbol{x}^{\{1\}} \right), \qquad (26)$$

where $K_P > 0$ is a scalar control parameter. The control law switches to the next desired point in the sequence, as soon as $t - t_A > T_{AB}$.

The considered unilateral constraints are an upper bound $\bar{x}_C = 0.5$ m and a lower bound $\underline{x}_C = -0.5$ m on the Y-component of the tip position of the fourth link (i.e., the location of the fifth joint). The upper bound $\bar{x}_C$ will be a hard constraint for the motion task (26), while the lower bound $\underline{x}_C$ is considered as a soft constraint for the same task. The activation parameters are $\beta_P = 0.1$ and $\beta_V = 0.1$. Damped pseudoinverses are computed with $\epsilon = 0.1$ and $\mu^2 = 0.1$, while the joint velocity command initialization (15) at each step is used with $\lambda = 0.9$. The other time parameters are $T = 1$ ms and $T_{AB} = 2$ s.

Figures 3-4 show the results of the first example, in which the motion task requires just to reach the single point $\boldsymbol{X}_B = [2, -2]$. Three snapshots of the robot evolution are displayed in Fig. 3, illustrating the correct execution of the desired task. The fulfillment of the two unilateral constraints and the evolution of their activation variable h_C are presented in Fig. 4.

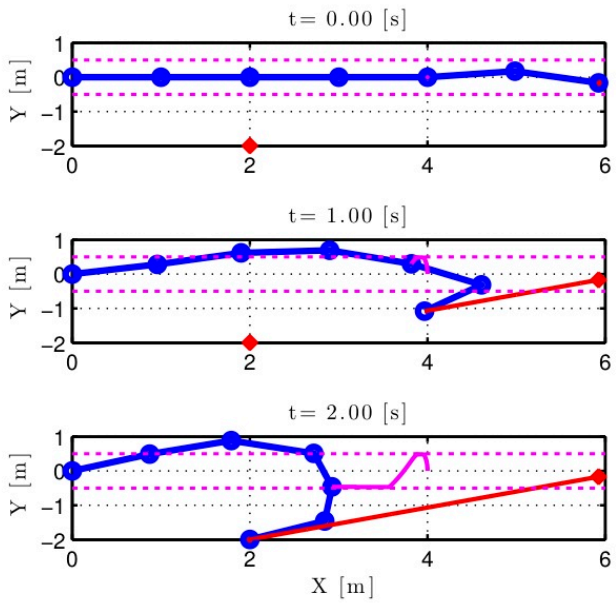

Fig. 3. Simulation 1. Robot snapshots at $t = 0$, $t = 1$, and $t = 2$ s. The primary task is to move the end-effector from the starting position $\boldsymbol{X}_A$ to $\boldsymbol{X}_B = [2, -2]$, taking into account the lower and upper constraints (shown as dashed magenta lines) for the Y-coordinate of the fifth joint. The executed trajectories of the end-effector and of the fifth joint location are shown as continuous lines in red and in magenta, respectively.

We note that, despite the smooth transition designed for the activation variable, the continuity of the joint velocity command is guaranteed only thanks to the damped minimization of the variation in discrete time of the velocity commands.

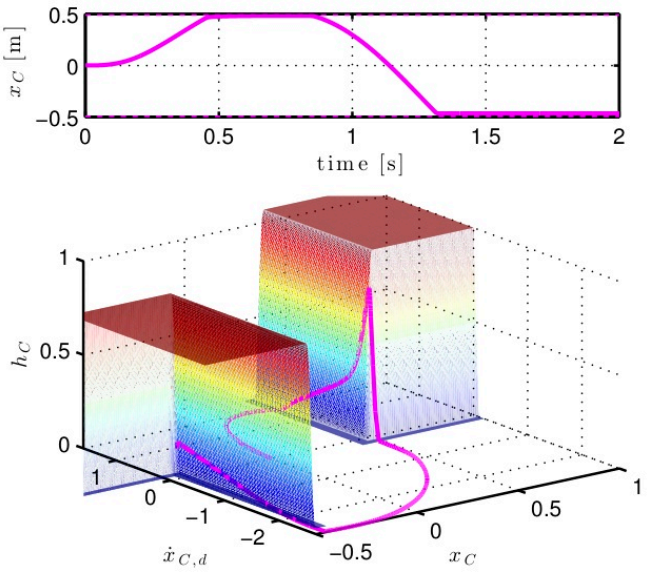

Fig. 4. Simulation 1. Evolution in time of the constrained component x_C (the Y-coordinate of the fifth joint) [top] and of the associated activation variable h_C [bottom].

Figure 5 shows the two different behaviors: when this method is not used, $\lambda = 0$ in (a), discontinuities are encountered, while they are fully removed in (b) using $\lambda = 0.9$ in (15).

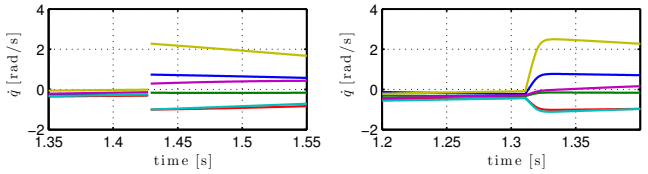

Fig. 5. Simulation 1. Close-up view of the joint velocity commands with $\lambda = 0$ (a) and with $\lambda = 0.9$ (b).

In the second example, the primary task requires to move the robot end-effector from the starting point $\boldsymbol{X}_A$ to the point $\boldsymbol{X}_B = [4, 3]$ and then to $\boldsymbol{X}_C = [2, -3]$. Both Cartesian points cannot be reached within the unilateral constraints imposed on the Y-coordinate of the fifth joint. Results from this simulation are shown in Fig. 6. The different hard and soft characteristics of the two unilateral constraints are quite evident: the upper bound (hard constraint of higher priority than the motion task) is never exceeded, and thus the first Cartesian point $\boldsymbol{X}_B$ cannot be reached; however, the second point $\boldsymbol{X}_C$ is reached, thanks to the relaxation of the lower bound (soft constraint of lower priority).

In the third example, a C++ implementation of the RP method was applied to a kinematic model of a 37-dof humanoid robot. The results of this simulation are shown in Fig. 7. With reference to this figure, the global reference frame is fixed at the right foot (in black). The tasks to be executed, ordered by decreasing priority, are:

1) keep the left foot (in blue) at the initial position [bilateral constraints];

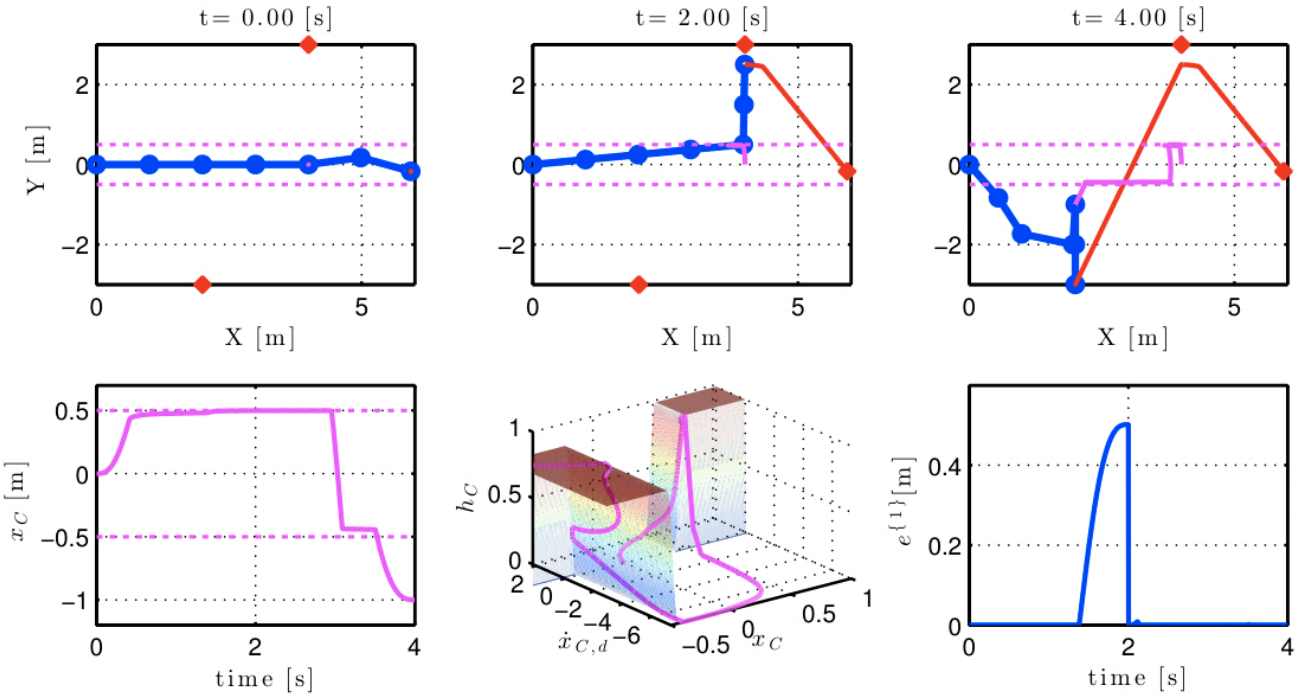

Fig. 6. Simulation 2. Robot snapshots at $t = 0$, $t = 2$, and $t = 4$ s [top]. In the bottom row: evolution in time of the constrained component x_C [left], associated activation variable h_C [center], and norm of the primary task error [right].

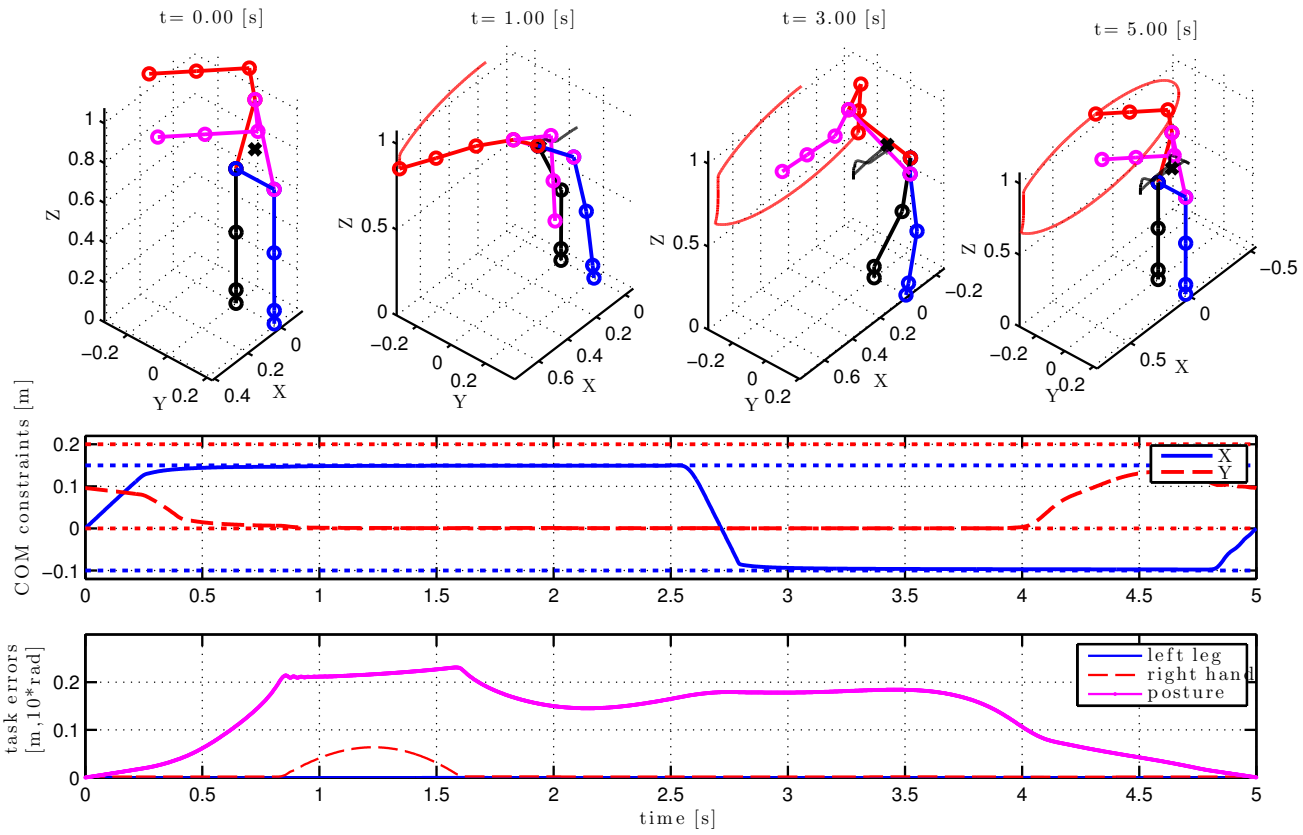

Fig. 7. Simulation 3. Humanoid snapshots at $t = 0$, $t = 1$, $t = 3$, and $t = 5$ s [top], with the right hand trajectory (red) and the COM trajectory (gray). Evolution in time of the two unilaterally constrained components of the COM [center]. Error norms on the equality tasks 1), 4), and 5) [bottom].

2) upper and lower bounds or the X-coordinate of the Center of Mass (COM)[1] [unilateral constraints];
3) upper and lower bounds or the Y-coordinate of the Center of Mass (COM) [unilateral constraints];
4) ellipsoid motion for the right hand (in red) [bilateral constraints];
5) keep the initial configuration as postural task [bilateral constraints].

The constraints on the projected position of the COM in the (X, Y)-plane are necessary to keep the humanoid robot in equilibrium. One can see that the left foot is always kept at the initial position (highest priority) and all unilateral constraints are satisfied, thus guaranteeing the humanoid balance. On the other hand, the motion task assigned to the right hand cannot be fully executed without destabilizing the robot. Thus, this task is partly deformed (right after $t = 1$ s).

VI. Experimental Results

To illustrate the practical feasibility and value of the proposed method, we have performed also experiments on a KUKA LWR IV robot. The LWR is controlled using the Fast Research Interface (FRI), which allows updating desired joint position references at high frequency rates, 500 Hz in our case ($T = 2$ ms). All methods have been coded in C++, using the Eigen library [24] for algebraic computations. Experiments were performed using the ROS environment [25] on a Intel Core i7-2600 CPU 3.4GHz, with 8Gb of RAM. Starting from the initial configuration $\boldsymbol{q}_0 = (0.5236, 1.5708, 0.3491, 1.3963, 0, -0.3491, 0)$ [rad] (first frames in the top and center rows of Fig. 8), the robot end-effector is required to move on a spiral trajectory (as illustrated in full in the last frames of Fig. 8). The unilateral constraints are given by maximum (absolute) variations of the joint positions with respect to the initial robot configuration. In particular, we have considered $\Delta\boldsymbol{q}_{max} = (0.3, 0.4, 0.5, 0.45, 0.5, 0.6, 0.6)$ [rad].

With the proposed approach, the task is executed correctly and all bounds for the joint positions are satisfied (Fig. 8). As already mentioned, at each step only a suboptimal solution is guaranteed. In fact, the behavior obtained with a state-of-the-art QP solver [26] is slightly different (Fig. 9). On the other hand, a redundancy resolution scheme based on a Projected Gradient approach, which tries at each step to bring the robot configuration closer to the initial one (moving in the null space of the primary task), will violate more than once the unilateral constraints (Fig. 10).

Figure 11 shows the time needed for computing the solution to be applied as joint velocity command at each sampling step. The time needed may vary during motion, depending on the number of currently active constraints. We note that the optimal result obtained by the QP solver, which is based on the active set approach, comes at the cost of a much larger computational burden. The PG scheme is indeed the fastest method, but it fails to comply with the

[1]For simplicity, in this simulation we have assumed the COM as a fixed point on the robot structure.

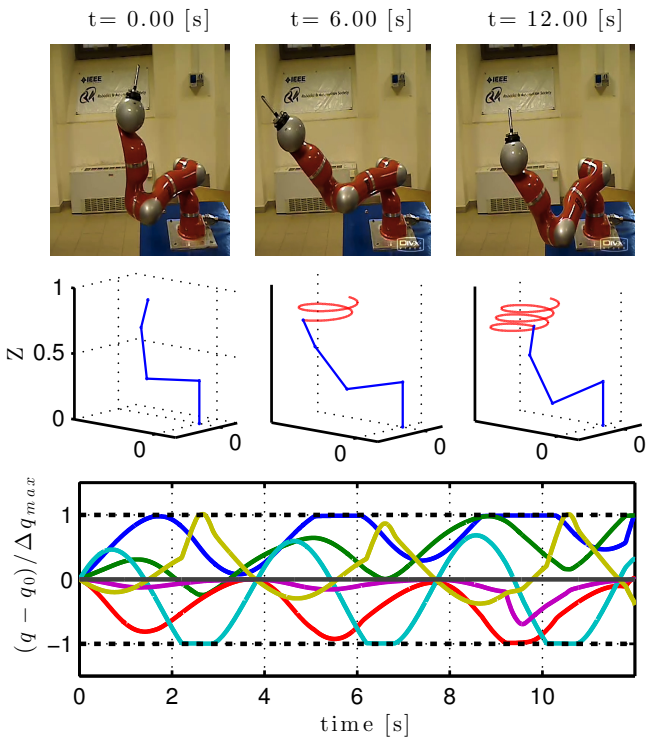

Fig. 8. Experiments on a KUKA LWR robot using the proposed algorithm based on the RP method. Screenshots of the experiment [top]. Evolution of the primary motion task [middle]. Joint position variations with respect to the initial configuration, normalized with respect to the maximum allowed range [bottom].

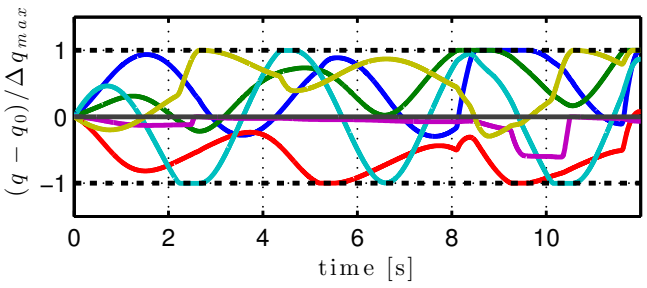

Fig. 9. Experiments on a KUKA LWR using the QP solver qpOASES [26]. Joint position variations with respect to the initial configuration, normalized with respect to the maximum allowed range.

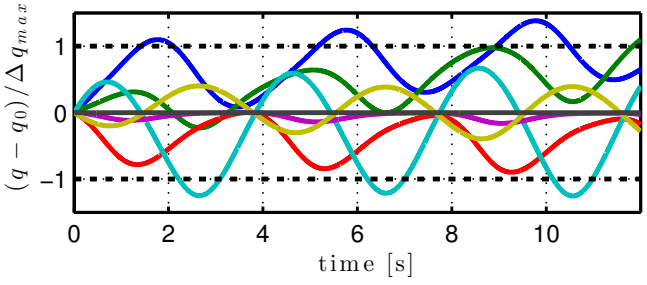

Fig. 10. Experiments on a KUKA LWR using a Projected Gradient approach with the aim of keeping the robot configuration close to the initial one. Joint position variations with respect to the initial configuration, normalized with respect to the maximum allowed range.

hard inequality constraints. The proposed RP method realizes instead the best compromise between an accurate solution and its fast computation.

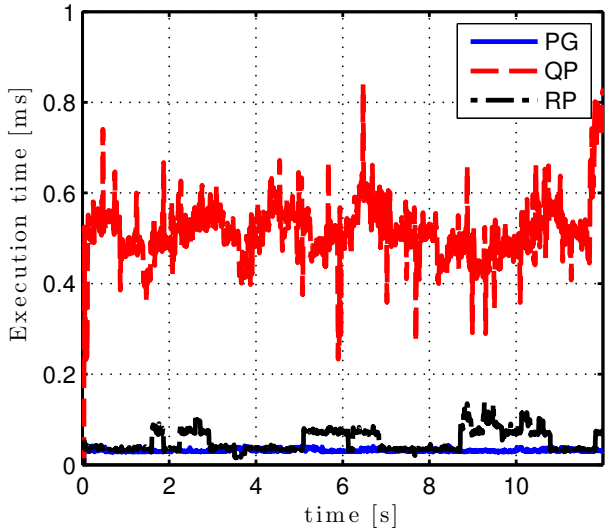

Fig. 11. Comparison of the execution times required for computing the solution at each sampling step using a Projected Gradient scheme (PG), an optimal QP solver (QP), and the proposed algorithm based on the Reverse Priority method (RP).

VII. Conclusions

We have presented an extension of our Reverse Priority redundancy resolution method that integrates also unilateral constraints at any level in the hierarchy of tasks. The main advantage is obtained thanks to the processing of tasks in the reverse order of priority, reducing the need of modifying a solution computed so far because of bad estimates of the active inequality constraints. Hard unilateral constraints are placed at the top of the priority stack and the check on their activation according to the current solution is very efficient. Moreover, suitable actions taken in the same framework guarantee the continuity of the joint velocity commands.

The effectiveness of the method was verified via multiple simulations and through actual experiments. Our method is a viable alternative to other existing approaches, with the main features of being simple to implement, having low computational cost, and guaranteeing always the fulfillment of hard inequality constraints.

In our future work, we plan the integration of the RP framework with the Saturation in the Null Space algorithm [4], which considers hard bounds on robot capabilities, exploiting task redundancy and scaling the task if needed.

References

[1] S. Chiaverini, G. Oriolo, and I. Walker, "Kinematically redundant manipulators," in *Springer Handbook of Robotics*, B. Siciliano and O. Khatib, Eds. Springer, 2008, pp. 245–268.

[2] L. Sciavicco and B. Siciliano, "A solution algorithm to the inverse kinematic problem for redundant manipulators," *IEEE J. on Robotics and Automation*, vol. 4, pp. 403–410, 1988.

[3] A. De Luca, G. Oriolo, and B. Siciliano, "Robot redundancy resolution at the acceleration level," *Laboratory Robotics and Automation*, vol. 4, no. 2, pp. 97–106, 1992.

[4] F. Flacco, A. De Luca, and O. Khatib, "Control of redundant robots under joint constraints: Saturation in the null space," *IEEE Trans. on Robotics*, vol. 31, no. 3, pp. 637–654, 2014.

[5] F. Arrichiello, S. Chiaverini, G. Indiveri, and P. Pedone, "The null-space-based behavioral control for mobile robots with velocity actuator saturations," *Int. J. of Robotics Research*, vol. 29, no. 10, pp. 1317–1337, 2010.

[6] Y. Zhang, J. Wang, and Y. Xia, "A dual neural network for redundancy resolution of kinematically redundant manipulators subject to joint limits and joint velocity limits," *IEEE Trans. on Neural Networks*, vol. 14, no. 3, pp. 658–667, 2003.

[7] F. Flacco, T. Kröger, A. De Luca, and O. Khatib, "A depth space approach to human-robot collision avoidance," in *Proc. IEEE Int. Conf. on Robotics and Automation*, 2012, pp. 338–345.

[8] N. Mansard and F. Chaumette, "Visual servoing sequencing able to avoid obstacles," in *Proc. IEEE Int. Conf. on Robotics and Automation*, 2005, pp. 3143–3148.

[9] L. Rosenberg, "Virtual fixtures: Perceptual tools for telerobotic manipulation," in *Proc. IEEE Virtual Reality Annual Int. Symp.*, 1993, pp. 76–82.

[10] G. Chesi, K. Hashimoto, D. Prattichizzo, and A. Vicino, "Keeping features in the field of view in eye-in-hand visual servoing: a switching approach," *IEEE Trans. on Robotics*, vol. 20, no. 5, pp. 908–914, 2004.

[11] E. Yoshida, O. Kanoun, C. Esteves, and J.-P. Laumond, "Task-driven support polygon reshaping for humanoids," in *Proc. 6th IEEE-RAS Int. Conf. on Humanoid Robots*, 2006, pp. 208–213.

[12] O. Kanoun, F. Lamiraux, and P.-B. Wieber, "Kinematic control of redundant manipulators: Generalizing the task-priority framework to inequality task," *IEEE Trans. on Robotics*, vol. 27, no. 4, pp. 785–792, 2011.

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

[14] F. Chaumette and E. Marchand, "A new redundancy-based iterative scheme for avoiding joint limits: Application to visual servoing," in *Proc. IEEE Int. Conf. on Robotics and Automation*, 2000, pp. 1720–1725.

[15] N. Mansard, O. Khatib, and A. Kheddar, "A unified approach to integrate unilateral constraints in the stack of tasks," *IEEE Trans. on Robotics*, vol. 25, no. 3, pp. 670–685, 2009.

[16] A. Liégeois, "Automatic supervisory control of the configuration and behavior of multibody mechanisms," *IEEE Trans. on Systems, Man and Cybernetics*, vol. 7, no. 12, pp. 868–871, 1977.

[17] B. Nelson and P. Khosla, "Strategies for increasing the tracking region of an eye-in-hand system by singularity and joint limit avoidance," *Int. J. of Robotics Research*, vol. 14, no. 3, pp. 418–423, 1995.

[18] M. Marey and F. Chaumette, "New strategies for avoiding robot joint limits: Application to visual servoing using a large projection operator," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2010, pp. 6222–6227.

[19] T. Chanand and R. Dubey, "A weighted least-norm solution based scheme for avoiding joint limits for redundant joint manipulators," *IEEE Trans. on Robotics*, vol. 11, no. 2, pp. 286–292, 1995.

[20] F. Flacco and A. De Luca, "A reverse priority approach to multi-task control of redundant robots," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2014, pp. 2421–2427.

[21] ——, "Discrete-time velocity control of redundant robots with acceleration/torque optimization properties," in *Proc. IEEE Int. Conf. on Robotics and Automation*, 2014, pp. 5139–5144.

[22] T. L. Boullion and P. L. Odell, *Generalized Inverse Matrices*. Wiley-Interscience, 1971.

[23] A. Deo and I. Walker, "Overview of damped least-squares methods for inverse kinematics of robot manipulators," *J. of Intelligent and Robotic Systems*, vol. 14, no. 1, pp. 43–68, 1995.

[24] G. Guennebaud, B. Jacob, *et al.*, "Eigen v3," http://eigen.tuxfamily.org, 2010.

[25] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: An open-source robot operating system," in *ICRA Work. on Open Source Robotics*, Kobe, JPN, May 2009.

[26] H. Ferreau, C. Kirches, A. Potschka, H. Bock, and M. Diehl, "qpOASES: A parametric active-set algorithm for quadratic programming," *Mathematical Programming Computation*, vol. 6, no. 4, pp. 327–363, 2014.