2013 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS)
November 3-7, 2013. Tokyo, Japan

Fast Redundancy Resolution for High-Dimensional Robots Executing Prioritized Tasks under Hard Bounds in the Joint Space

Fabrizio Flacco Alessandro De Luca

***Abstract*— A kinematically redundant robot with limited motion capabilities, expressed by inequality constraints of the box type on joint variables and commands, needs to perform a set of tasks, expressed by linear equality constraints on robot commands, possibly organized with priorities. Robot motion capabilities cannot be exceeded at any time, and the resulting constraints are to be considered as hard bounds. Instead, robot tasks can be relaxed by velocity scaling if no feasible solution exists. To address this redundancy resolution problem, we developed a method in which joint space commands are successively saturated and their effect compensated in the null space of a suitable task Jacobian (SNS, Saturation in the Null Space). Computationally efficient versions of the basic and optimal SNS algorithms are proposed here, based on a task augmentation reformulation, a QR factorization of the main matrices involved, and a so-called warm start procedure. The obtained performance allows to control in real time robots with high-dimensional configuration spaces executing a large number of prioritized tasks, and with an associated high number of hard bounds that saturate during motion.**

I. Introduction

Consider a robot with a large number of degrees of freedom (DOF) that is kinematically redundant with respect to a task to be executed. Typically, the desired evolution of task variables (e.g., the pose of the robot end-effector or the position of the elbow joint in a manipulator arm) is encoded at the differential level in a task velocity, which may be specified also in real time from sensory feedback. In the current robot state, each task is then locally defined by a set of *linear equality constraints* on the joint velocity commands. Because of robot redundancy, there will be an infinite number of joint velocity commands realizing the task (in an exact or least squares error sense), and we typically look for commands having a minimum norm property [1]. Multiple tasks may be assigned to the robot, and a priority order can be enforced in case the whole stack of tasks cannot be executed simultaneously [2]. A recursive solution for multiple tasks has been proposed in [3], and refined in [4] with an efficient recursive formula for the null-space projection matrix.

In this framework, a given solution may become unfeasible when considering also the actual *robot motion capabilities*. These are usually expressed by the presence of upper and lower limits on the configuration variables (joint ranges), as well as on differential quantities in the joint space (velocity, acceleration, torque limits). None of such constraints can be violated during motion: these additional linear (box) inequalities should be treated as *hard bounds* on the solution. Whenever the robot capabilities are insufficient to execute the original set of tasks, it is reasonable to allow for a *task scaling* (on the equality constraints). In this way, the obtained solution will be feasible for a relaxed problem, where the original direction of the desired task velocity is preserved, even if its execution speed is not.

In [5], we have introduced a novel algorithm, called Saturation in the Null Space (SNS), for solving on line the above problem in the case of a single task. All joint-space limits are first combined into hard bounds for the joint velocity commands at the current robot configuration. In its basic form, the SNS algorithm successively modifies a pseudoinverse solution by bringing back to its saturated value one overdriven joint velocity command at a time (actually, the most violating one), and projecting this into the null space of the task Jacobian of the enabled (non-saturated) joints. A task scaling procedure is embedded in the algorithm, so that the process can be repeated until a final feasible solution is always found. The hard bounds in the joint space and the SNS algorithm can also be redefined at the level of joint acceleration commands [5]. The method has been extended in [6] to the case of a stack of prioritized tasks, each expressed as linear equality constraints. In this situation, the SNS algorithm enforces a *preemptive strategy*: higher priority tasks are preserved (up to scaling, when strictly needed) by using all the available robot capabilities they need, while lower priority tasks are accommodated with the residual robot capabilities (and scaling). In [7], only the presence of joint range limits has been considered next to multiple equality tasks with priority, and the simultaneous clamping of joints was not handled in a way as efficient as by the SNS.

The problem considered so far fits indeed into the general

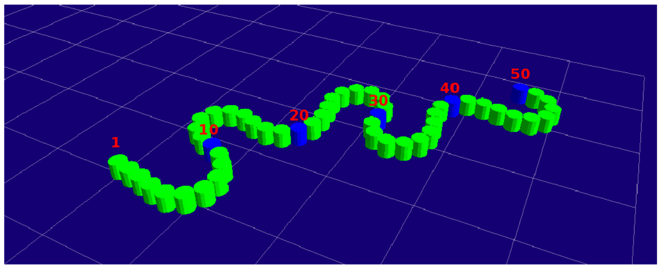

Fig. 1. A 50-DOF robot accomplishing 5 two-dimensional tasks in real time, taking into account the equivalent of 200 hard joint limits

The authors are with the Dipartimento di Ingegneria informatica, automatica e gestionale Antonio Ruberti, Università di Roma “La Sapienza”, Via Ariosto 25, 00185 Rome, Italy ({fflacco,deluca}@dis.uniroma1.it). This work is supported by the European Commission, within the FP7 ICT-287513 SAPHARI project (www.saphari.eu).

framework of Quadratic Programming (QP), with a quadratic objective function to be minimized (typically, a norm) and linear equality and inequality constraints [8]. The robotic application of this optimization method puts the emphasis on the proper formulation of tasks, on the handling of their priorities, and, foremost, on the computational efficiency for real time execution. The inclusion of linear inequality constraints in a task priority approach was considered first in [9] and [10]. Several numerical issues were then addressed for the faster resolution of a hierarchy of inverse kinematics problems with inequality constraints [11], [12], including the concepts of active constraints (and related Karush-Kuhn-Tucker multipliers), QR factorizations (of task Jacobians and null-space projection matrices), 'warm start' (reusing part of the previous solution in a time sequence of closely related QP problems), and so on.

In [13], we reformulated our SNS approach in the context of quadratic programming with linear task equalities and box (inequality) constraints. In doing so, we modified the basic algorithm (essentially, tuning the order in which the single overdriven commands are being saturated, based on the analysis of the multipliers) so as to guarantee optimality of the solution (*Opt-SNS* algorithm). The main outcome is that the SNS (or the Opt-SNS) algorithm is faster than a general QP solver (e.g., the one used in [10]), because it exploits the structure of the given problem.

Nevertheless, apart from the efficiency in finding a solution, there are some fundamental differences between our SNS approach and the general formulation of [10], [11]. First, the box inequality constraints that characterize robot motion capabilities are taken out of the stack of tasks and *enforced separately*. Second, the hard bounds on the commands include also a prediction of violation of joint ranges, resulting in an overall *smoother behavior* (if not continuity). Third, when the set of equality tasks cannot be realized by the robot capabilities, *task scaling* is automatically considered in the problem formulation. This prevents the output of an approximate minimum error solution that would however violate the hard bounds[1].

As a matter of fact, possible differences between the provided solutions with the approach of [10], [11] and with our SNS method arise only when the problem admits no feasible region (i.e., no command satisfy simultaneously all equality task constraints and all box inequality constraints). Hard inequality constraints should always be set at the top of the priority stack in [10], [11]. But the handling of such hard inequalities is not fully clear (e.g., whether these constraints need to be enforced at any priority level, or if a backtrack to previous tasks occurs in case the k-th one needs more of the robot capabilities), and unacceptable violations may still occur if they correspond to an approximate solution with the least possible error in norm —a situation ruled out within our SNS approach.

[1]An underlying hypothesis in SNS operation is that the origin of the command space (e.g., the velocity $\dot{\boldsymbol{q}} = \boldsymbol{0}$) is always a feasible choice for a relaxed problem with equality constraints being sufficiently scaled. When the generic k-th equality task is not feasible (i.e., the task velocity $\dot{\boldsymbol{x}}_k$ does not belong to the range space of the Jacobian $\mathcal{R}\{\boldsymbol{J}_k\}$, then the problem is addressed using a damped least squares approach and the SNS algorithm is applied using the damped Jacobian pseudoinverse as a basis.

The main goal of this paper is to improve the numerical performance of the SNS [5] and Opt-SNS [13] algorithms, by using similar machinery as in [11], [12]. After recalling some basic results in Sects. II and III, we introduce in Sec. IV another formulation of the problem based on task augmentation [14], where the information on saturated commands is inserted as additional equality constraints (this is quite similar to the active set idea). The QR factorization of the main involved matrices is used to speed up computations, leading to the *Fast-SNS* algorithm of Sect. V and to the *FastOpt-SNS* algorithm of Sect. VI. Section VII gives a few details on the warm start of the method, while numerical results are presented in Sect. VIII for a hyper-redundant planar robot (see Fig. 1), with up to $n = 200$ degrees of freedom (DOF) and under very severe saturation conditions.

II. Redundancy Resolution

Let $\boldsymbol{q} \in \mathbb{R}^n$ be the generalized (joint) coordinates of a robot, $\boldsymbol{x}_k \in \mathbb{R}^{m_k}$ the variables describing a generic task, with $m_k \leq n$, and $\boldsymbol{J}_k = \boldsymbol{J}_k(\boldsymbol{q})$ the associated $m_k \times n$ task Jacobian matrix. At a given robot configuration $\boldsymbol{q}$, the task differential kinematics is

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

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

$$\dot{\boldsymbol{q}}_k = \boldsymbol{J}_k^{\#} \dot{\boldsymbol{x}}_k + \boldsymbol{P}_k \dot{\boldsymbol{q}}_N, \tag{2}$$

where $\boldsymbol{J}_k^{\#}$ is the (unique) pseudoinverse of the task Jacobian [15], $\boldsymbol{P}_k = \boldsymbol{I} - \boldsymbol{J}_k^{\#} \boldsymbol{J}_k$ is the $n \times n$ orthogonal projector in the Jacobian null space, and $\dot{\boldsymbol{q}}_N \in \mathbb{R}^n$ is a generic joint velocity.

The problem can be extended to l tasks in the form (1), each of dimension m_k, $k = 1, \ldots, l$ (typically, but not necessarily, with $\sum_{k=1}^{l} m_k \leq n$), that are ordered by their priority, i.e., task i has higher priority than task j when $i < j$. The execution of a task of lower priority should not interfere with the execution of tasks having higher priority, and this hierarchy is guaranteed by projecting the execution of the k-th task of the stack in the null space of all higher priority tasks. This is obtained by using the recursive formula [3]

$$\dot{\boldsymbol{q}}_k = \dot{\boldsymbol{q}}_{k-1} + (\boldsymbol{J}_k \boldsymbol{P}_{A,k-1})^{\#} \left(\dot{\boldsymbol{x}}_k - \boldsymbol{J}_k \dot{\boldsymbol{q}}_{k-1}\right), \tag{3}$$

initialized with $\dot{\boldsymbol{q}}_0 = \boldsymbol{0}$ and $\boldsymbol{P}_{A,0} = \boldsymbol{I}$, and where $\boldsymbol{P}_{A,k}$ is the projector in the null space of the augmented Jacobian of the first k tasks

$$\boldsymbol{J}_{A,k} = \begin{pmatrix} \boldsymbol{J}_1^T & \boldsymbol{J}_2^T & \ldots & \boldsymbol{J}_k^T \end{pmatrix}^T. \tag{4}$$

Matrix $\boldsymbol{P}_{A,k}$ can also be expressed recursively as [4]

$$\boldsymbol{P}_{A,k} = \boldsymbol{P}_{A,k-1} - (\boldsymbol{J}_k \boldsymbol{P}_{A,k-1})^{\#} \boldsymbol{J}_k \boldsymbol{P}_{A,k-1}. \tag{5}$$

Different numerical methods can be used to compute the solutions (2) or (3). The most common is to resort to a Singular Value Decomposition (SVD) of matrices. Using

SVD provides the singular values of a matrix, and thus its rank and condition number. It is then possible to check if the task is singular or close to a singularity, and in that case obtain a damped pseudoinverse as an approximate solution (selectively damping the singular values [16]). However, the SVD is computationally quite expensive. On the other hand, a QR decomposition is certainly faster [17]. In addition, for the case of multiple tasks with priorities, using the approach presented in [18], it is possible to progressively reduce the dimension of the matrices to be (pseudo)inverted.

Consider the QR decomposition of (the transpose of) the generic task Jacobian

$$\boldsymbol{J}_k^T = \boldsymbol{Q}_k \begin{pmatrix} \boldsymbol{R}_k \\ \boldsymbol{0} \end{pmatrix}, \tag{6}$$

where $\boldsymbol{Q}_k = (\boldsymbol{Y}_k \quad \boldsymbol{Z}_k)$ is a $n \times n$ orthogonal matrix, decomposed in a $n \times m_k$ matrix $\boldsymbol{Y}_k$ and a $n \times (n - m_k)$ matrix $\boldsymbol{Z}_k$, and the upper triangular matrix $\boldsymbol{R}_k$ has dimension $m_k \times m_k$. The solution (2) for a single task is obtained as

$$\dot{\boldsymbol{q}}_k = \boldsymbol{Y}_k \boldsymbol{R}_k^{-T} \dot{\boldsymbol{x}}_k + \boldsymbol{Z}_k \boldsymbol{z}_N. \tag{7}$$

The columns of $\boldsymbol{Z}_k$ provide a (minimal) basis for the Jacobian null space, and $\boldsymbol{z}_N \in \mathbb{R}^{n-m_k}$ is a generic vector in the *reduced* space of redundant DOFs. The standard null space projector can be obtained as $\boldsymbol{P}_k = \boldsymbol{Z}_k \boldsymbol{Z}_k^T$.

For the multi-task case, it is possible to derive a recursive formula based on eq. (3). It is

$$\dot{\boldsymbol{q}}_k = \dot{\boldsymbol{q}}_{k-1} + \boldsymbol{Z}_{A,k-1} \boldsymbol{Y}_{A,k} \boldsymbol{R}_{A,k}^{-T} \left(\dot{\boldsymbol{x}}_k - \boldsymbol{J}_k \dot{\boldsymbol{q}}_{k-1} \right), \tag{8}$$

initialized with $\dot{\boldsymbol{q}}_0 = \boldsymbol{0}$ and $\boldsymbol{Z}_{A,0} = \boldsymbol{I}$, and using the QR decomposition

$$\left(\boldsymbol{J}_k \boldsymbol{Z}_{A,k-1} \right)^T = (\boldsymbol{Y}_{A,k} \quad \boldsymbol{Z}_k) \begin{pmatrix} \boldsymbol{R}_{A,k} \\ \boldsymbol{0} \end{pmatrix}. \tag{9}$$

Moreover, we compute recursively

$$\boldsymbol{Z}_{A,k} = \boldsymbol{Z}_{A,k-1} \boldsymbol{Z}_k. \tag{10}$$

The dimension of $\boldsymbol{J}_k \boldsymbol{Z}_{A,k-1}$ is $m_k \times (n - \sum_{i=1}^{k-1} m_i)$. Therefore, adding the contribution of a low priority task will become faster as we go down the stack of tasks.

For simplicity, equations (7) and (8) have been written assuming that the generic task k is non-singular (and no *algorithmic singularities* [2] occur in the stack up to task k), so that matrices $\boldsymbol{R}_k$ or $\boldsymbol{R}_{A,k}$ are invertible. When the determinant, e.g., of matrix $\boldsymbol{R}_k$ (the product of its diagonal elements) is smaller than a given threshold, the task will be considered singular, and a damped solution is obtained through the damped pseudoinversion of $\boldsymbol{R}_k$, achieved using the SVD of $\boldsymbol{R}_k$. This operation is computationally fast, being $\boldsymbol{R}_k$ a square upper triangular matrix (of dimension m_k).

III. SNS ALGORITHMS

The robot motion capabilities are limited by the following box constraints set on joint-space quantities:

$$\begin{aligned} \boldsymbol{Q}_{min} &\le \boldsymbol{q} \le \boldsymbol{Q}_{max} && \text{(joint range)} \\ -\boldsymbol{V}_{max} &\le \dot{\boldsymbol{q}} \le \boldsymbol{V}_{max} && \text{(max joint velocity)} \\ -\boldsymbol{A}_{max} &\le \ddot{\boldsymbol{q}} \le \boldsymbol{A}_{max} && \text{(max joint acceleration)} \end{aligned} \tag{11}$$

Indeed, these limits can never be violated. In this paper, we assume that the robot is commanded with joint velocities $\dot{\boldsymbol{q}}$, so that only the first two sets of $4n$ inequality constraints in (11) can be enforced. To handle directly also acceleration limits, the command should be defined at the same differential level—see [5] for more details. At the current configuration $\boldsymbol{q}$, we can derive from (11) the following equivalent hard bounds on $\dot{\boldsymbol{q}}$

$$\dot{\boldsymbol{Q}}_{min}(\boldsymbol{q}) \le \dot{\boldsymbol{q}} \le \dot{\boldsymbol{Q}}_{max}(\boldsymbol{q}), \tag{12}$$

which specify that, for each joint $i = 1, \dots, n$, the velocity command $\dot{q}_i$ must guarantee *i)* that the joint range limits will not be exceeded in the next step, *ii)* that $\dot{q}_i$ is within its limits, and *iii)* that the joint will be able to stop its motion before reaching its closest joint limit, taking into account the maximum acceleration limits.

The basic Saturation in the Null Space (SNS) algorithm has been developed to resolve redundancy for a single task [5], or for multiple tasks with priorities [6], under the hard bounds (12). The core idea of the algorithm is to modify a pseudoinverse solution by removing at each step the most critical command that exceeded its bounds, and reintroducing its contribution at the saturation level through the null space of the task. When a task velocity $\dot{\boldsymbol{x}}_k$ is not feasible for the robot capabilities, the SNS scales its magnitude preserving the desired task direction.

The SNS redundancy resolution formula for multiple tasks with priority is

$$\dot{\boldsymbol{q}}_k = \dot{\boldsymbol{q}}_{k-1} + \left(\boldsymbol{J}_k \bar{\boldsymbol{P}}_k \right)^{\#} \left(s_k \dot{\boldsymbol{x}}_k - \boldsymbol{J}_k \dot{\boldsymbol{q}}_{k-1} \right) + \tilde{\boldsymbol{P}}_k \dot{\boldsymbol{q}}_{N,k}, \tag{13}$$

where $\dot{\boldsymbol{q}}_0 = \boldsymbol{0}$ and

$$\bar{\boldsymbol{P}}_k = \left(\boldsymbol{I} - \left(\left(\boldsymbol{I} - \boldsymbol{W}_k \right) \boldsymbol{P}_{A,k-1} \right)^{\#} \right) \boldsymbol{P}_{A,k-1} \tag{14}$$

is the projector in the null space of the augmented task (4), considering only non-saturated joints (i.e., the contribution of $\left(\boldsymbol{J}_k \bar{\boldsymbol{P}}_k \right)^{\#} \left(s_k \dot{\boldsymbol{x}}_k - \boldsymbol{J}_k \dot{\boldsymbol{q}}_{k-1} \right)$ is zero for saturated joints). The $n \times n$ selection matrix $\boldsymbol{W}_k = \operatorname{diag}\{W_{k,ii}\}$ with 0/1 elements specifies which joints are currently enabled or disabled: if $W_{k,ii} = 0$, the velocity of joint i is at its saturation level and the joint is disabled (for norm minimization purposes). The matrix

$$\tilde{\boldsymbol{P}}_k = \left(\boldsymbol{I} - \left(\boldsymbol{J}_k \bar{\boldsymbol{P}}_k \right)^{\#} \boldsymbol{J}_k \right) \left(\left(\boldsymbol{I} - \boldsymbol{W}_k \right) \boldsymbol{P}_{A,k-1} \right)^{\#} \tag{15}$$

is a special projector that allows to accommodate a joint velocity saturation, without deforming the task nor the hierarchy of priorities. The i-th element of $\dot{\boldsymbol{q}}_{N,k}$ contains the desired value for the i-th joint. If the i-th joint is saturated and $\dot{Q}_{sat,i}$ ($sat = \{min, max\}$) is its saturation value, then $\dot{q}_{N,k,i} = \dot{Q}_{sat,i} - \dot{q}_{k-1,i}$; otherwise, it is equal to zero. The scalar s_k represents the task scaling factor and is computed with Algorithm 1, where $\boldsymbol{a} = \left(\boldsymbol{J}_k \bar{\boldsymbol{P}}_k \right)^{\#} \dot{\boldsymbol{x}}_k$ and $\boldsymbol{b} = \dot{\boldsymbol{q}}_k - \boldsymbol{a}$.

If $s_k \ge 1$, the task is feasible with the current solution. Else, Algorithm 1 identifies the most critical joint commands that will be saturated. When $\operatorname{rank}\left(\boldsymbol{J}_k \bar{\boldsymbol{P}}_k \right) < m_k$, no more saturations are possible: the task is unfeasible and the solution with the highest scale factor computed so far is used.

Algorithm 1 (Task scaling factor)

```
function getTaskScalingFactor($\boldsymbol{a}$, $\boldsymbol{b}$)
for $i = 1 \to n$ do
    $S_{min,i} = \left(\dot{Q}_{min,i} - b_i\right)/a_i$
    $S_{max,i} = \left(\dot{Q}_{max,i} - b_i\right)/a_i$
    if $S_{min,i} > S_{max,i}$ then
        {switch $S_{min,i}$ and $S_{max,i}$}
    end if
end for
$s_{max} = \min_i \{S_{max,i}\}$
$s_{min} = \max_i \{S_{min,i}\}$
the most critical joint = $\mathrm{argmin}_i \{S_{max,i}\}$
if $s_{min} > s_{max}$ .OR. $s_{max} < 0$ .OR. $s_{min} > 1$ then
    task scaling factor = 0
else
    task scaling factor = $s_{max}$
end if
```

In [13], the SNS method for the k-th priority task has been reformulated as the solution of the QP problem:

$$\begin{aligned}\dot{\boldsymbol{q}}_k = \arg \min_{\dot{\boldsymbol{q}} \in \mathbb{R}^n} & \ \frac{1}{2}\|\dot{\boldsymbol{q}}\|^2 \\ \text{s.t. } & \boldsymbol{J}_k \dot{\boldsymbol{q}} = s_k \dot{\boldsymbol{x}}_k, \\ & \boldsymbol{J}_{A,k-1}\dot{\boldsymbol{q}} = \boldsymbol{J}_{A,k-1}\dot{\boldsymbol{q}}_{k-1}, \\ & \dot{\boldsymbol{Q}}_{min} \leq \dot{\boldsymbol{q}} \leq \dot{\boldsymbol{Q}}_{max},\end{aligned} \tag{16}$$

where the task scaling factor s_k is the highest found with the SNS algorithm. Based on the Karush-Kuhn-Tucker (KKT) conditions [19] for this QP problem, the Opt-SNS algorithm has been derived. It allows to verify if a saturated joint velocity has to be removed from saturation, by checking the sign of the multipliers

$$\boldsymbol{\mu}_k = \tilde{\boldsymbol{P}}_k^T \dot{\boldsymbol{q}}_k, \tag{17}$$

leading to the optimal solution of the QP problem (16).

IV. A NEW APPROACH TO SNS

It is possible to show that the same SNS redundancy resolution formula (13) can be obtained using a suitable task augmentation [14], which will then allow the use of a faster QR decomposition. Let[2]

$$\begin{aligned}\dot{\boldsymbol{q}}_k &= \dot{\boldsymbol{q}}_{k-1} + \boldsymbol{J}_T^{\#}\dot{\boldsymbol{q}}_T \\ &= \dot{\boldsymbol{q}}_{k-1} + \begin{pmatrix} \boldsymbol{J}_k \boldsymbol{P}_{A,k} \\ \boldsymbol{J}_W \end{pmatrix}^{\#} \begin{pmatrix} s_k \dot{\boldsymbol{x}}_k - \boldsymbol{J}_k \dot{\boldsymbol{q}}_{k-1} \\ \dot{\boldsymbol{q}}_W \end{pmatrix},\end{aligned} \tag{18}$$

where $\boldsymbol{J}_T$ is the augmentation of $\boldsymbol{J}_k \boldsymbol{P}_{A,k}$ with a $p \times n$ matrix $\boldsymbol{J}_W$, and being p the number of saturated commands. If joint i is saturated, the associated row $\boldsymbol{J}_{W,i}$ of the auxiliary Jacobian $\boldsymbol{J}_W$ contains all zeros, except a 1 as i-th element. The vector $\dot{\boldsymbol{q}}_W = \boldsymbol{J}_W \dot{\boldsymbol{q}}_{N,k}$, related to the SNS vector $\dot{\boldsymbol{q}}_{N,k}$, is composed by the saturation values associated to $\boldsymbol{J}_W$. The result of eq. (18) coincides with that of eq. (13) if there exists at least a feasible solution. Otherwise, equation (18) provides the smallest violation (in a least squares sense) of the constraints, producing however a relaxation of the box constraints that is forbidden in the SNS framework. This situation is in fact related to the SNS exit condition, namely $\mathrm{rank}\left(\boldsymbol{J}_k \bar{\boldsymbol{P}}_k\right) < m_k$.

[2]From now on, to reduce notational burden, some intermediate quantities that are used only within the solution of the k-th task will bear no k index.

By simple inspection, the pseudoinversion of $\boldsymbol{J}_T$ can be partitioned as

$$\boldsymbol{J}_T^{\#} = \left(\left(\boldsymbol{J}_k \bar{\boldsymbol{P}}_k\right)^{\#} \quad \boldsymbol{B}\right), \tag{19}$$

with $\boldsymbol{B}$ a $n \times p$ matrix containing the columns of $\tilde{\boldsymbol{P}}_k$ associated to the saturated joints, i.e., $\boldsymbol{B} = \tilde{\boldsymbol{P}}_k \boldsymbol{J}_W^T$. Taking into account that at each step of the SNS algorithm only one joint is inserted in (or removed from) the saturation list, equation (18) can be obtained from the previous step as a *rank one* update [20] of the pseudoinverse of the augmented Jacobian. Moreover, due to the simple structure of the row that has to be appended to $\boldsymbol{J}_T$, namely $\boldsymbol{J}_{W,i}$ if the i-th joint saturated, the update reduces to very simple formulas.

We detail next the operations associated with a new command (the $(p+1)$-th) reaching saturation. Removal from saturation is done following similar steps. Introducing the $n \times n$ orthogonal projector $\hat{\boldsymbol{P}}_{A,k}$, initialized to $\boldsymbol{P}_k$ as in (5) and composed by the columns $\hat{\boldsymbol{p}}_{k,i}$ $(i = 1, \ldots, n)$, the *update vector* $\boldsymbol{b}_i$ is obtained as

$$\boldsymbol{b}_i = \frac{\hat{\boldsymbol{p}}_{k,i}}{\hat{p}_{k,ii}}, \tag{20}$$

where $\hat{p}_{k,ii}$ denotes the i-th element of vector $\hat{\boldsymbol{p}}_{k,i}$. Then, matrix $\boldsymbol{B}$ is updated and augmented by one column as

$$\boldsymbol{B} = \left(\left(\boldsymbol{I} - \boldsymbol{b}_i \boldsymbol{J}_{W,i}\right)\boldsymbol{B} \quad \boldsymbol{b}_i\right), \tag{21}$$

and the SNS solution becomes

$$\begin{aligned}\dot{\boldsymbol{q}}_k &= \dot{\boldsymbol{q}}_{k-1} + \boldsymbol{B}\boldsymbol{J}_W \dot{\boldsymbol{q}}_{N,k} \\ &+ \left(\boldsymbol{I} - \boldsymbol{B}\boldsymbol{J}_W\right)\left(\boldsymbol{J}_k \boldsymbol{P}_{A,k-1}\right)^{\#}\left(s_k \dot{\boldsymbol{x}}_k - \boldsymbol{J}_k \dot{\boldsymbol{q}}_{k-1}\right).\end{aligned} \tag{22}$$

Finally, the introduced projection matrix $\hat{\boldsymbol{P}}_k$ is updated as

$$\hat{\boldsymbol{P}}_k = \hat{\boldsymbol{P}}_k + \boldsymbol{B}\boldsymbol{J}_W\left(\boldsymbol{I} - \hat{\boldsymbol{P}}_k\right). \tag{23}$$

Note that the columns of $\hat{\boldsymbol{P}}_k$ associated to saturated joints are equal to the columns of $\tilde{\boldsymbol{P}}_k$ in (15), and to the corresponding columns of $\boldsymbol{B}$.

V. THE FAST-SNS ALGORITHM

Based on the results of the previous section, the update of the SNS solution can be achieved with a minimum amount of operations. If the joint i saturates, the update vector $\boldsymbol{b}_i$ is obtained directly from the QR decomposition of (9) and (10) as

$$\boldsymbol{b}_i = \boldsymbol{Z}_{A,k}\, \boldsymbol{z}_i^{\#} = \boldsymbol{Z}_{A,k} \frac{\boldsymbol{z}_i^T}{\boldsymbol{z}_i \boldsymbol{z}_i^T}, \tag{24}$$

where $\boldsymbol{z}_i$ represents the i-th row of $\boldsymbol{Z}_{A,k}$.

For the *Fast-SNS* algorithm update, it is useful to split the SNS solution in four terms

$$\dot{\boldsymbol{q}}_k = \dot{\boldsymbol{q}}_{k-1} + s_k \dot{\bar{\boldsymbol{q}}}' + \dot{\bar{\boldsymbol{q}}}'' + \dot{\bar{\boldsymbol{q}}}_W, \tag{25}$$

with initializations $\dot{\bar{\boldsymbol{q}}}' = \dot{\boldsymbol{q}}'$, $\dot{\bar{\boldsymbol{q}}}'' = \dot{\boldsymbol{q}}''$, $\dot{\bar{\boldsymbol{q}}}_W = \dot{\boldsymbol{q}}_W$, where

$$\begin{aligned} \dot{\boldsymbol{q}}' &= \boldsymbol{Z}_{A,k-1}\boldsymbol{Y}_k\boldsymbol{R}_k^{-T}\dot{\boldsymbol{x}}_k \\ \dot{\boldsymbol{q}}'' &= -\boldsymbol{Z}_{A,k-1}\boldsymbol{Y}_k\boldsymbol{R}_k^{-T}\boldsymbol{J}_k\dot{\boldsymbol{q}}_{k-1} \\ \dot{\boldsymbol{q}}_W &= \boldsymbol{0} \end{aligned} \tag{26}$$

come from the solution (8), which is obtained without considering the inequalities (12). In the following, we will call (26) the *unconstrained* solution. At each new saturation, these terms are updated as

$$\begin{aligned} \dot{\bar{\boldsymbol{q}}}' &= \dot{\bar{\boldsymbol{q}}}' - \boldsymbol{b}_i\dot{\bar{\boldsymbol{q}}}'_i \\ \dot{\bar{\boldsymbol{q}}}'' &= \dot{\bar{\boldsymbol{q}}}'' - \boldsymbol{b}_i\dot{\bar{\boldsymbol{q}}}''_i \\ \dot{\bar{\boldsymbol{q}}}_W &= \dot{\bar{\boldsymbol{q}}}_W + \boldsymbol{b}_i\left(\dot{Q}_{sat,i} - \dot{\boldsymbol{q}}_{k-1,i} - \dot{\bar{\boldsymbol{q}}}_{W,i}\right), \end{aligned} \tag{27}$$

where $\dot{Q}_{sat,i}$ is the saturation value. The scaling factor s_k is given by Algorithm 1, called with $\boldsymbol{a} = \dot{\bar{\boldsymbol{q}}}'$ and $\boldsymbol{b} = \dot{\boldsymbol{q}}_k - \boldsymbol{a}$ (this is the reason for splitting the second and third terms in (25)). The null space projector is then updated as

$$\boldsymbol{Z}_{A,k} = \boldsymbol{Z}_{A,k} - \boldsymbol{b}_i\boldsymbol{z}_i. \tag{28}$$

VI. The FastOpt-SNS Algorithm

The algorithm of Sect. V yields in general a suboptimal solution in terms of the QP problem (16). Namely, only the norm of the velocity vector of non-saturated joints is minimized. Following the results of [13], as recalled in Sect. III, we need then *i)* to compute and update the multipliers (17), and *ii)* to update the solution when a joint command is removed from its saturation state. This is the core of the *FastOpt-SNS* algorithm presented next.

The multiplier associated to a new saturated joint command of index i (the $(p+1)$-th saturated one) is given by

$$\mu_{k,i} = \boldsymbol{b}_i^T\dot{\boldsymbol{q}}_k. \tag{29}$$

All other p multipliers associated to previously saturated joints have to be updated as

$$\mu_{k,j} = \mu_{k,j} - b_{j,i}\,\mu_{k,i}, \qquad j = 1,\dots,p, \tag{30}$$

where $b_{j,i}$ is the i-th element of the update vector $\boldsymbol{b}_j$ for the j-th saturated joint. Equation (30) shows that these vectors need to be stored and updated at each new saturation. This can be done with

$$\boldsymbol{b}_j = \boldsymbol{b}_j - \boldsymbol{b}_i b_{j,i}, \qquad j = 1,\dots,p. \tag{31}$$

Since the p vectors $\boldsymbol{b}_j$ compose the matrix $\boldsymbol{B}$, eq. (31) executes the same update as eq. (21).

Using the multipliers, it is possible to identify a saturated joint command, say with index o, that should not saturate in the optimal solution. In this case, joint o has to be removed from the list of p saturated commands and the solution downgraded. The first step is to downgrade the update vectors associated to the $p-1$ joints that will remain in saturation:

$$\boldsymbol{b}_j = \boldsymbol{b}_j - \boldsymbol{b}_o\left(\frac{\boldsymbol{b}_o^T\boldsymbol{b}_j}{\boldsymbol{b}_o^T\boldsymbol{b}_o}\right), \qquad j = 1,\dots,p. \tag{32}$$

Indeed, $\boldsymbol{b}_o$ becomes zero and will be discarded. Next, the solution is downgraded as

$$\begin{aligned} \dot{\bar{\boldsymbol{q}}}' &= \dot{\bar{\boldsymbol{q}}}' + \boldsymbol{b}_o\left(\dot{\bar{q}}'_o - \sum_{j=1}^{p} b_{j,o}\dot{\bar{q}}'_j\right) \\ \dot{\bar{\boldsymbol{q}}}'' &= \dot{\bar{\boldsymbol{q}}}'' + \boldsymbol{b}_o\left(\dot{\bar{q}}''_o - \sum_{j=1}^{p} b_{j,o}\dot{\bar{q}}''_j\right) \\ \dot{\bar{\boldsymbol{q}}}_W &= \dot{\bar{\boldsymbol{q}}}_W - \boldsymbol{b}_o\left(\dot{Q}_o - \sum_{j=1}^{p} b_{j,o}\left(\dot{Q}_j - \dot{q}_{k-1,j}\right)\right), \end{aligned} \tag{33}$$

where $\dot{\boldsymbol{q}}'$ and $\dot{\boldsymbol{q}}''$ compose the unconstrained solution obtained with eqs. (26). The null space projector has to restore the additional direction given by the o-th joint. Thus, $\boldsymbol{Z}_{A,k}$ is downgraded as

$$\boldsymbol{Z}_{A,k} = \boldsymbol{Z}_{A,k} - \boldsymbol{b}_0\left(\tilde{\boldsymbol{z}}_o - \sum_{j=1}^{p} b_{j,o}\tilde{\boldsymbol{z}}_j\right), \tag{34}$$

where $\tilde{\boldsymbol{z}}_i$ is the i-th row of $\boldsymbol{Z}_{A,k-1}\boldsymbol{Z}_k$. Finally, the $p-1$ multipliers associated to joints that will remain in saturation are downgraded by

$$\mu_{k,j} = \mu_{k,j} + b_{j,o}\,\mu_{k,o} \qquad j = 1,\dots,p, \quad j \neq o. \tag{35}$$

After the downgrade, $\mu_{k,o}$ will be set to zero.

VII. Warm Start

With the robot in motion, the stack of tasks needs to be executed at every control sampling time. If the variation of a desired task is small between one sample and the next one, the new redundancy resolution problem will be close to the previously solved one, and so its solution. Therefore, almost the same set of saturated commands can be expected. In a warm start, we assume to know which joints were in saturation for the k-th task at the previous time sample, e.g., the matrix $\boldsymbol{J}_W$ is known. The new solution can be efficiently computed starting from the same previous set of saturated joints and proceeding as follows.

We first obtain the update vectors associated to all saturated joints as

$$\boldsymbol{B} = \boldsymbol{Z}_{A,k}\left(\boldsymbol{J}_W\boldsymbol{Z}_{A,k}\right)^{\#}. \tag{36}$$

The update vectors allow to obtain the SNS solution from the unconstrained solution in (26)

$$\begin{aligned} \dot{\bar{\boldsymbol{q}}}' &= \dot{\boldsymbol{q}}' - \boldsymbol{B}\boldsymbol{J}_W\dot{\boldsymbol{q}}' \\ \dot{\bar{\boldsymbol{q}}}'' &= \dot{\boldsymbol{q}}'' - \boldsymbol{B}\boldsymbol{J}_W\dot{\boldsymbol{q}}'' \\ \dot{\bar{\boldsymbol{q}}}_W &= \dot{\boldsymbol{q}}_W + \boldsymbol{B}\boldsymbol{J}_W\dot{\boldsymbol{q}}_{N,k}, \end{aligned} \tag{37}$$

where $\dot{\boldsymbol{q}}_{N,k}$ was introduced in Sect. III. The directions associated to the saturated joints are removed from the null space projector by

$$\boldsymbol{Z}_{A,k} = \boldsymbol{Z}_{A,k} - \boldsymbol{B}\boldsymbol{J}_W\boldsymbol{Z}_{A,k}, \tag{38}$$

and the multipliers associated to the saturated joints are obtained by

$$\boldsymbol{\mu}_k = \boldsymbol{B}^T\dot{\boldsymbol{q}}_k. \tag{39}$$

Note finally that, in the actual implementation, multiplication by $\boldsymbol{J}_W$ represents only an extraction/reordering of a submatrix or subvector from the multiplied quantity.

VIII. Numerical Results

To evaluate the performance of the new redundancy resolution algorithms *Fast-SNS* and *FastOpt-SNS*, a high-dimensional planar robot with a (varying) number n of revolute joints and links of unitary lengths has been considered (see Fig. 1). The planar case makes a parametric analysis easier. However, the outcome would be very similar also in case of spatial redundant robots moving in 3D. All SNS versions have been developed in C++, using the Eigen library [21] for algebraic computations. Simulations were performed using the ROS environmenton a Intel Core i7-2600 CPU 3.4GHz, with 8Gb of RAM.

First test. The planar positioning of the end-effector of the n-DOF robot is the single task considered ($m_1 = 2$). The joint limits (11) are, for $i = 1, \dots, n$,

$$\begin{aligned} Q_{max,i} &= -Q_{min,i} = 90 \quad \text{[deg]} \\ V_{max,i} &= 1 \quad \text{[deg/s]} \\ A_{max,i} &= 3 \quad \text{[deg/s}^2\text{]}, \end{aligned} \tag{40}$$

with a (non-time based) task velocity

$$\dot{\boldsymbol{x}} = V_C \sin\left(\left(1 - \frac{\|\boldsymbol{x}_d - \boldsymbol{x}\|}{\|\boldsymbol{x}_d - \boldsymbol{x}_0\|}\right)\pi + \varepsilon\right)\frac{\boldsymbol{x}_d - \boldsymbol{x}}{\|\boldsymbol{x}_d - \boldsymbol{x}_0\|}, \tag{41}$$

where $\boldsymbol{x}$, $\boldsymbol{x}_d$, and $\boldsymbol{x}_0$ are the current, desired, and initial Cartesian positions of the end effector, and $\varepsilon = 10^{-4}$ is a small parameter that allows motion ignition. The robot starts from the stretched configuration $\boldsymbol{q}_0 = \boldsymbol{0}$, corresponding to $\boldsymbol{x}_0 = (n\ 0)^T$ (along the $y = 0$ axis). The desired Cartesian position of the tip of the generic link $r \in \{1, \dots, n\}$ is

$$\boldsymbol{x}_d(r) = \left(\frac{\sqrt{2}}{2}r \quad \frac{\sqrt{2}}{2}r\right)^T, \tag{42}$$

with $r = n$ being the case of the robot end-effector considered first. The task velocity contains a scalar that was set to $V_C = 2n$ [m/s], a relatively large value. Note that the joint limits and desired task velocity have been designed so as to induce a maximal number (i.e., up to $n - m_1$) of joint velocity saturations during the execution of the task.

Figure 2 shows the worst-case execution times obtained when increasing the number n of DOF of the robot. The Opt-SNS, which was already found to be much faster than a standard QP solver [13], becomes too slow when n increases. Assuming 10 [ms] as the maximum limit of execution time

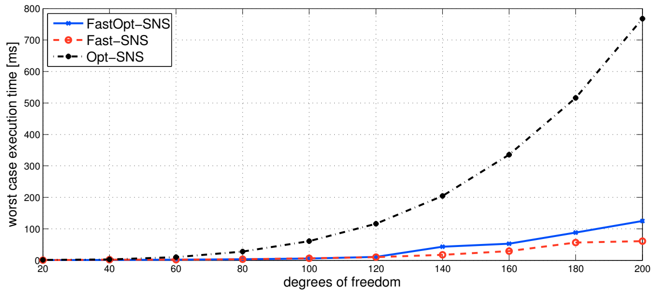

Fig. 2. Worst-case execution times to accomplish a single two-dimensional task with a n-DOF robot, when varying n from 20 to 200. The redundancy resolution algorithm considers $m_1 = 2$ task equalities and the equivalent of $4n$ box inequalities

for on-line use, the Opt-SNS cannot be used for a hyper-redundant robot with more than 60 DOF, while the Fast-SNS and the FastOpt-SNS can handle up to 120 DOF. Note that the Fast-SNS does not guarantee the optimality of the solution (in terms of the QP problem (16)), and does not compute multipliers nor update the associated vectors. Therefore, its execution time is smaller than the one of FastOpt-SNS, especially when the n grows larger than 120.

Second test. In the second test, the number of DOF of the planar robot is fixed to $n = 50$ and the number l of prioritized two-dimensional position tasks is varied from 2 to 10. Each task is characterized by a final desired position for the tip of link r in the kinematic chain, as given by eq. (42). The considered links were extracted from the following list, ordered according to the priorities of the tasks: $\{50, 30, 40, 10, 20, 45, 5, 35, 15, 25\}$. The same joint limits (40) of the first test have been used, with the same type of desired velocity law (41) for each task.

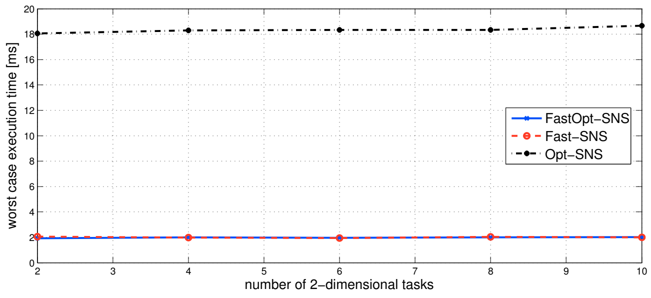

Fig. 3. Worst-case execution times to accomplish l prioritized two-dimensional tasks with a robot having $n = 50$ DOF, when varying l from 2 to 10. The redundancy resolution algorithm considers a total of $2l$ task equalities and the equivalent of 200 box inequalities

The worst-case execution times of the redundancy resolution algorithms are plotted in Fig. 3. It is clear that this execution time increases only slightly with the number of tasks for all version of SNS. The two *Fast* versions run at worst in 2 [ms] even when 10 tasks are considered, thus being eligible for applications in real time. The video clip accompanying the paper shows the animation of the 50-DOF robot for $l = 1$, $l = 3$, and $l = 5$ tasks with priority when using FastOpt-SNS. Figure 4 shows the execution times obtained while the 50-DOF robot is running 5 position tasks in the form (41–42), taking into account the joint limits (40). The final robot configuration obtained once all tasks are completed is the one shown in Fig. 1. The number of joint command saturations occurring during robot motion are plotted in Fig. 5. The practical independence of execution times from the number of saturated commands is quite evident with the *Fast* versions of the SNS algorithm.

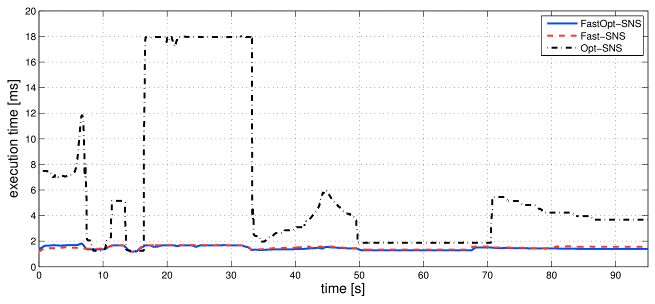

Fig. 4. Evolution of the execution time while running $l = 5$ prioritized two-dimensional tasks with a robot having $n = 50$ DOF

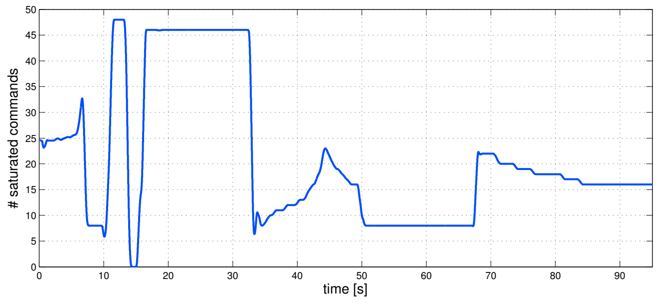

Fig. 5. Number of saturated joint commands in the simulation of Fig. 4

IX. Conclusions

We have presented computationally efficient versions of our basic and optimal SNS algorithms for resolving redundancy in high-dimensional robots that execute prioritized sets of tasks, under hard bounds on joint variables and commands. The achieved performance is compatible with real-time control applications. This has been obtained by exploiting the problem structure and thanks to the use of QR factorizations of the augmented/saturated task Jacobians and associated null-space projection matrices. In view of the need to solve the problem recurrently during robot operation, a warm start procedure was adopted. The algorithms are competitive or superior to the state-of-the-art in hierarchical quadratic programming, but still with unique features: *i)* hard bounds are never violated under any circumstance, *ii)* correct priority of tasks is preserved even in unfeasible cases, thanks to the pre-emptive strategy, and *iii)* task scaling (preserving the desired direction and relaxing just intensity) is automatically performed on the equality constraints, only when strictly needed to guarantee command feasibility.

This last feature is useful when the robot shares the workspace with a human, and their safe coexistence requires the robot to be driven away from potential collisions detected on the fly. The tuning of control parameters, e.g., of the repulsive potential field used in [22], is no longer a critical issue, and even very large gains could be used. In fact, the SNS algorithm will automatically scale the commanded motion according to the robot capabilities, while keeping the (best) escape direction.

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] Y. Nakamura, *Advanced Robotics: Redundancy and Optimization*. Addison-Wesley, 1991.

[3] B. Siciliano and J. J. Slotine, "A general framework for managing multiple tasks in highly redundant robotic systems," in *Proc. 5th Int. Conf. on Advanced Robotics*, 1991, pp. 1211–1216.

[4] P. Baerlocher and R. Boulic, "Task-priority formulations for the kinematic control of highly redundant articulated structures," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 1998, pp. 323–329.

[5] F. Flacco, A. De Luca, and O. Khatib, "Motion control of redundant robots under joint constraints: Saturation in the null space," in *Proc. IEEE Int. Conf. on Robotics and Automation*, 2012, pp. 285–292.

[6] F. Flacco, A. De Luca, and O. Khatib, "Prioritized multi-task motion control of redundant robots under hard joint constraints," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2012, pp. 3970–3977.

[7] P. Baerlocher and R. Boulic, "An inverse kinematic architecture enforcing an arbitrary number of strict priority levels," *The Visual Computer*, vol. 6, no. 20, pp. 402–417, 2004.

[8] D. Luenberger, *Linear and Nonlinear Programming*. Addison-Wesley, 1984.

[9] 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.

[10] 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.

[11] A. Escande, N. Mansard, and P.-B. Wieber, "Fast resolution of hierarchized inverse kinematics with inequality constraints," in *Proc. IEEE Int. Conf. on Robotics and Automation*, 2010, pp. 3733–3738.

[12] ——, "Hierarchical Quadratic Programming, Tech. Rep. LAAS no. 12794, Oct. 2012. [Online]. Available: http://hal.archives-ouvertes.fr/hal-00751924.

[13] F. Flacco and A. De Luca, "Optimal redundancy resolution with task scaling under hard bounds in the robot joint space," in *Proc. IEEE Int. Conf. on Robotics and Automation*, 2013, pp. 3954–3960.

[14] P. Chiacchio, S. Chiaverini, L. Sciavicco, and B. Siciliano, "Closed-loop inverse kinematics schemes for constrained redundant manipulators with task space augmentation and task priority strategy," *Int. J. of Robotics Research*, vol. 10, no. 4, pp. 410–425, 1991.

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

[16] A. Maciejewki and C. Klein, "Numerical filtering for the operation of robotic manipulators through kinematically singular configurations," *J. of Robotic Systems*, vol. 5, no. 6, pp. 527–552, 1988.

[17] G. Golub and C. Van Loan, *Matrix Computations*. Johns Hopkins University Press, 1996.

[18] O. Kanoun, "Real-time prioritized kinematic control under inequality constraints for redundant manipulators," in *Proc. of Robotics: Science and Systems VII*, 2011.

[19] H. W. Kuhn and A. Tucker, "Nonlinear programming," in *Proc. of 2nd Berkeley Symp.*, 1951.

[20] T. Greville, "Some applications of pseudoinverse of a matrix," *SIAM Review*, vol. 2, pp. 15–22, 1960.

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

[22] 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.