2013 IEEE International Conference on Robotics and Automation (ICRA)
Karlsruhe, Germany, May 6-10, 2013

Optimal Redundancy Resolution with Task Scaling under Hard Bounds in the Robot Joint Space

Fabrizio Flacco Alessandro De Luca

Abstract— **For robots that are redundant with respect to a given task, we present an optimal differential kinematic inversion method in the presence of hard bounds on joint range, joint velocity, and joint acceleration. These hard bounds specify the robot motion capabilities that cannot be exceeded at any time. On the other hand, scaling of the desired task trajectory is allowed whenever the robot capabilities are insufficient to execute the original task. For a problem formulated in this way, we have recently presented the Saturation in the Null Space (SNS) algorithm that produces an efficient solution, based on Jacobian pseudoinversion and recovery in the null space of the saturation effects of a reduced number of joint velocity commands. To investigate the optimality properties of the SNS algorithm, we recast the problem as a constrained quadratic programming (QP) problem, in which the joint velocity norm as well as the task scaling are to be minimized. Its solution leads to a variant of the original algorithm, the Optimal Saturation in the Null Space (Opt-SNS). The Opt-SNS guarantees an optimal solution also when the basic SNS fails to do so and improves the numerical performance over the state-of-the-art QP solver. The possible existence of discontinuous solutions for the formulated problem is avoided by the introduction of a task scaling margin. The extension to the multi-task case is also presented. Simulation results for the 7R lightweight KUKA LWR IV robot illustrate the properties and computational efficiency of the new algorithm.**

I. Introduction

The Saturation in the Null Space (SNS) algorithm introduced in [1] is a powerful method that solves the inverse differential kinematic problem of task-redundant robots, taking into account hard constraints on the joint velocity commands coming from bounds on the joint motion that can never be exceeded. The SNS works by projecting into the null space of the task Jacobian the joint velocities that saturate, and finds in this way a feasible solution even when classic Jacobian pseudoinversion fails. When the task is not feasible for the robot motion capabilities, the SNS integrates a task scaling procedure that allows the robot to execute at least part of the task without deforming it (e.g., if the task is a desired end-effector trajectory, the traced geometric path remains the same). The SNS method was formulated also at the acceleration level in [1], and then extended to the case of multiple tasks in [2], where we used the concept of preemptive priority: tasks with higher priority use in the best way all the robot capabilities they need, while lower priority tasks are accommodated with the residual capability without interfering with the execution of higher priority tasks.

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).

Uncompensated saturations of the velocity commands may perturb the geometry of the task under execution in a dangerous way, e.g., when the robot is working close to a human operator. This problem is crucial also for robots with many DOFs, such as humanoids. In fact, when trying to simultaneously execute a large multiplicity of tasks, each not highly demanding but jointly exceeding the overall robot capabilities, the unavoidable saturation of some joint commands may induce a relaxation of crucial high priority tasks, such as loss of balance (an inequality constraint on the Zero Moment Point location) or physical collisions with the environment. Since the SNS guarantees permanent fulfillment of hard joint constraints, the desired direction of each task and the preemptive priority of multiple tasks are always preserved even under saturated commands. As a matter of fact, even when very large or untuned task velocities are commanded, the joint velocity computed by the SNS method will safely preserve the task geometry, automatically scaling only the task timing.

Despite the above positive features of the SNS algorithm and the successful results reported in a large variety of actual robot experiments in [1], [2], the optimality of the solution found cannot be proved in general. In other words, there is no guarantee that the SNS minimizes the norm of the joint velocity command or the task scaling or both. In order to investigate the possible optimality of the SNS solution, we need to reformulate the inverse differential kinematics of a redundant robot as an optimization problem, typically a quadratic programming (QP) problem with linear equality and inequality constraints.

There is a consistent bulk of literature dealing with optimization-based redundancy resolution. If the desired task can be accomplished without saturation, i.e., the problem is actually unconstrained, the optimal solution in terms of minimum joint velocity norm is obtained using the pseudoinversion of the task Jacobian [3]. Constraints are usually taken into account by using task augmentation [4] while objective functions by the projected gradient method [5]. However, these methods do not guarantee in general both the constraint satisfaction and the optimality of the solution.

In general, inequality constrained optimization problems can be solved using an active-set method [6], where the active set is composed by those constraints that are satisfied as equalities (e.g., saturated joint variables in our robotic case). The associated optimization algorithms will find the optimal active set that leads to the constrained optimal solution. Unfortunately, these general algorithms cannot be used in real-time applications, due to their high computational cost.

In [7], the constrained kinematic redundancy problem was formulated as a QP and a Gram-Schmidt orthogonalization procedure applied for its solution. A compact QP method, using Gaussian elimination with partial pivoting, was developed in [8] to reduce the computational complexity, which however is still high. To solve the QP problem, a dual neural network was used in [9]. However, unfeasible tasks are not considered and the application to multiple tasks is not addressed. In [10], the constrained kinematic inversion was again modeled as a QP problem, but in a more elegant way that could cover also a hierarchy of prioritized tasks. However, the computational cost of the solution remained prohibitive for robots with a large number of DOFs, even when resorting to a state-of-the-art QP solver. Based on a similar stack of QP problems, the solver was improved in [11] by using a complete orthogonal decomposition to compute the null spaces of tasks with prioritized hierarchy. This improvement allowed a real-time implementation of the solver.

A common drawback of [10], [11] is that the case of task that are anyway *unfeasible* for the given set of robot capabilities is not addressed directly. In particular, the presence and use of slack variables in [11] allows in this case relaxatiuon of inequality constraints that should be treated instead as hard bounds. When the constraints have a physical meaning and needs to be treated as hard bounds, e.g., joint range limits, maximum joint velocity, or Cartesian obstacles, the robot will not be able to move as requested. By not allowing task scaling (as done instead in our formulation and solution), the task execution will be arbitrarily deformed, and the whole task priority architecture destroyed.

To address the above limitations, we present in this paper a deeper mathematical analysis of the SNS algorithm introduced in [1], [2], leading to a modified version that preserves its positive features and allows to generate an optimal solution in an efficient way. The resulting Optimal Saturation in the Null Space (Opt-SNS) algorithm can be seen as an active-set QP solver that takes advantage of the structure of the considered problem. After recalling the standard SNS algorithm in Sect. II, the redundancy resolution problem with hard joint constraints is formulated in Sect. III as a quadratic programming problem, and the necessary and sufficient conditions for optimality are checked against the algorithmic steps of the SNS. Section IV introduces the Optimal SNS algorithm, with some related implementation issues illustrated in Sect. V. Simulation results obtained with the 7R KUKA LWR IV robot are presented in Sect. VI, where the Opt-SNS method is also numerically compared with a state-of-the-art QP solver. The problem of discontinuous joint velocity commands encountered sometimes with the SNS method is addressed and solved in Sect. VII. Finally, Section VIII is devoted to the extension of the Opt-SNS algorithm to the case of multiple tasks with priority.

II. The SNS Algorithm

Let $\boldsymbol{q} \in \mathbb{R}^n$ be the vector of generalized (joint) coordinates of a robot, $\boldsymbol{x} \in \mathbb{R}^m$ the vector of variables describing a generic m-dimensional task, with $m < n$, and $\boldsymbol{J}(\boldsymbol{q})$ the associated $m \times n$ task Jacobian matrix. At a given robot configuration $\boldsymbol{q}$, the direct differential kinematics of the task is given by

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

Taking into account the presence following hard bounds (in vector form) on the robot joint motion

$$\begin{aligned} \boldsymbol{Q}_{min} &\leq \boldsymbol{q} \leq \boldsymbol{Q}_{max} && \text{(joint range)} \\ -\boldsymbol{V}_{max} &\leq \dot{\boldsymbol{q}} \leq \boldsymbol{V}_{max} && \text{(max joint velocity)} \\ -\boldsymbol{A}_{max} &\leq \ddot{\boldsymbol{q}} \leq \boldsymbol{A}_{max} && \text{(max joint acceleration)} \end{aligned} \tag{2}$$

we obtain the box constraint for the joint velocity command $\dot{\boldsymbol{q}}$ at time $t = t_k$

$$\dot{\boldsymbol{Q}}_{min}(t_k) \leq \dot{\boldsymbol{q}} \leq \dot{\boldsymbol{Q}}_{max}(t_k), \tag{3}$$

where, for each joint $i = 1, \dots, n$, the velocity $\dot{q}_i$ must guarantee: (i) that the joint range limits will not be exceeded in the next step; (ii) that it is not greater than the maximum velocity; (iii) that the joint will be able to stop before the joint limit, considering the maximum acceleration. For more details about the shaping of these constraints, see [1].

At a given robot configuration $\boldsymbol{q}$, the SNS algorithm for realizing the task $\dot{\boldsymbol{x}}$ at the velocity level under the box constraints (3) is given in pseudocode form by **Algorithm 1**.

Therein, the $n \times n$ selection matrix $\boldsymbol{W} = \mathrm{diag}\{W_{ii}\}$ with 0/1 elements is used to specify which joints are currently enabled or disabled: if $W_{ii} = 0$, then the velocity of joint i is set at its saturation level and the joint is disabled (for norm minimization purposes). The algorithm is initialized with $\boldsymbol{W} = \boldsymbol{I}$ (the identity matrix), a null-space vector $\dot{\boldsymbol{q}}_N = \boldsymbol{0}$, and two scaling factors $s = 1$ and $s^* = 0$. The core (and final) joint velocity command computed by this algorithm uses the SNS projection equation

$$\dot{\boldsymbol{q}} = \dot{\boldsymbol{q}}_N + (\boldsymbol{J}\boldsymbol{W})^{\#}\,(s\dot{\boldsymbol{x}} - \boldsymbol{J}\dot{\boldsymbol{q}}_N) \tag{4}$$

Another important aspect is the integrated use of the task scaling factor $s \in \mathbb{R}$, which is eventually chosen as the largest possible one (i.e., equal to 1 if the task is feasible) that is compatible with the box constraints (3). If some component of the joint velocity (4) exceeds its bounds, **Algorithm 2** is called to evaluate the best task scaling factor only among the enabled joints. If the jth joint is the most critical for task execution, i.e., its velocity needs the largest relative decrease to stay within the bounds, this velocity is saturated and we set $W_{jj} = 0$. Algorithm 1 stops when rank $(\boldsymbol{J}\boldsymbol{W}) < m$, providing as output the best feasible solution found so far. In case of task singularity (i.e., rank $(\boldsymbol{J}) < m$) the algorithm is not used, and it is replaced by a damped pseudoinversion of the Jacobian. In this case a simple task scaling is used to satisfy the joint constraints. Further analysis of the properties of this algorithm and of the obtained solution is provided in [1].

Algorithm 1 (The SNS algorithm)

```
$\boldsymbol{W} = \boldsymbol{I}$, $\dot{\boldsymbol{q}}_N = \boldsymbol{0}$, $s = 1$, $s^* = 0$
repeat
  limit_exceeded = FALSE
  $\left(\tilde{\boldsymbol{P}} = \boldsymbol{I} - (\boldsymbol{J}\boldsymbol{W})^{\#} \boldsymbol{J}\right)$
  $\dot{\boldsymbol{q}} = \dot{\boldsymbol{q}}_N + (\boldsymbol{J}\boldsymbol{W})^{\#} (\dot{\boldsymbol{x}} - \boldsymbol{J}\dot{\boldsymbol{q}}_N) \left(= (\boldsymbol{J}\boldsymbol{W})^{\#} \dot{\boldsymbol{x}} + \tilde{\boldsymbol{P}}\dot{\boldsymbol{q}}_N\right)$
  if $\left\{ \begin{array}{l} \exists\, i \in [1:n] : \\ \dot{q}_i < \dot{Q}_{min,i} \text{ .OR. } \dot{q}_i > \dot{Q}_{max,i} \end{array} \right\}$ then
    limit_exceeded = TRUE
    $\boldsymbol{a} = (\boldsymbol{J}\boldsymbol{W})^{\#} \dot{\boldsymbol{x}}$
    $\boldsymbol{b} = \dot{\boldsymbol{q}} - \boldsymbol{a}$
    getTaskScalingFactor($\boldsymbol{a}$, $\boldsymbol{b}$) (*call Algorithm 2*)
    if {task scaling factor} $> s^*$ then
      $s^*$ = {task scaling factor}
      $\boldsymbol{W}^* = \boldsymbol{W}$, $\dot{\boldsymbol{q}}_N^* = \dot{\boldsymbol{q}}_N$
    end if
    $j$ = {the most critical joint}
    $W_{jj} = 0$
    $\dot{q}_{N,j} = \left\{ \begin{array}{ll} \dot{Q}_{max} & \text{if } \dot{q}_j > \dot{Q}_{max} \\ \dot{Q}_{min} & \text{if } \dot{q}_j < \dot{Q}_{min} \end{array} \right.$
    if rank($\boldsymbol{J}\boldsymbol{W}$) $< m$ then
      $s = s^*$, $\boldsymbol{W} = \boldsymbol{W}^*$, $\dot{\boldsymbol{q}}_N = \dot{\boldsymbol{q}}_N^*$
      $\dot{\boldsymbol{q}} = \dot{\boldsymbol{q}}_N + (\boldsymbol{J}\boldsymbol{W})^{\#} (s\dot{\boldsymbol{x}} - \boldsymbol{J}\dot{\boldsymbol{q}}_N)$
      limit_exceeded = FALSE   (*outputs solution*)
    end if
  end if
until limit_exceeded = TRUE
```

Algorithm 2 (Task scaling factor)

```
function getTaskScalingFactor($\boldsymbol{a}$, $\boldsymbol{b}$)
for $i = 1 \rightarrow n$ 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 = 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
```

III. Optimality for the SNS solution

Despite the SNS finds a sub-optimal task scaling factor, only the norm of joint velocities associated to non-saturated joints is minimized. It may be possible that the same task scaling factor is obtained with a feasible joint velocity vector that is smaller in norm. To check if the current solution is optimal, we define an associated optimization problem as follows

$$\begin{array}{ll} \min\limits_{\dot{\boldsymbol{q}},s} & \frac{1}{2}\dot{\boldsymbol{q}}^T\dot{\boldsymbol{q}} + \frac{1}{2}M(1-s)^2 \\ \text{s.t.} & s\dot{\boldsymbol{x}} = \boldsymbol{J}\dot{\boldsymbol{q}}, \quad \dot{\boldsymbol{Q}}_{min} \leq \dot{\boldsymbol{q}} \leq \dot{\boldsymbol{Q}}_{max} \end{array} \tag{5}$$

To give to the task scaling factor maximization an higher priority than the minimization of the joint velocity norm we set $M \gg 1$. The parameter M works as a large penalty factor.

The optimization problem (5) can be rewritten as a standard, positive definite, quadratic problem with equality and inequality (box) constraints

$$\begin{array}{ll} \min\limits_{\boldsymbol{\xi}} & \frac{1}{2}\boldsymbol{\xi}^T\boldsymbol{H}\boldsymbol{\xi} \\ \text{s.t.} & \boldsymbol{A}\boldsymbol{\xi} = \boldsymbol{b}, \quad \boldsymbol{C}\boldsymbol{\xi} \leq \boldsymbol{d} \end{array} \tag{6}$$

by defining

$$\begin{array}{ll} \boldsymbol{\xi} = \begin{pmatrix} \dot{\boldsymbol{q}} \\ 1-s \end{pmatrix}, & \boldsymbol{H} = \begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0}^T & M \end{pmatrix}, \\ \boldsymbol{A} = \begin{pmatrix} \boldsymbol{J} & \dot{\boldsymbol{x}} \end{pmatrix}, & \boldsymbol{b} = \dot{\boldsymbol{x}}, \\ \boldsymbol{C} = \begin{pmatrix} -\boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{I} & \boldsymbol{0} \end{pmatrix}, & \boldsymbol{d} = \begin{pmatrix} -\dot{\boldsymbol{Q}}_{min} \\ \dot{\boldsymbol{Q}}_{max} \end{pmatrix}, \end{array} \tag{7}$$

where $\boldsymbol{0}$ represents the zero vector of dimension n, and $\boldsymbol{I}$ is the $n \times n$ identity matrix.

Being our QP problem (6) convex with linear constraints, necessary and sufficient optimality conditions are given by the Karush-Kuhn-Tucker (KKT) criteria [12]. A scale factor and a joint velocity are optimal for the problem (5) if and only if the following conditions are satisfied:

$$\boldsymbol{H}\boldsymbol{\xi} + \boldsymbol{A}^T\boldsymbol{\lambda} + \boldsymbol{C}^T\boldsymbol{\mu} = \boldsymbol{0} \tag{8a}$$

$$\boldsymbol{\mu}\,(\boldsymbol{C}\boldsymbol{\xi} - \boldsymbol{d}) = \boldsymbol{0} \tag{8b}$$

$$\boldsymbol{A}\boldsymbol{\xi} = \boldsymbol{b} \tag{8c}$$

$$\boldsymbol{C}\boldsymbol{\xi} \leq \boldsymbol{d} \tag{8d}$$

$$\boldsymbol{\mu} \geq \boldsymbol{0} \tag{8e}$$

where $\boldsymbol{\lambda} \in \mathbb{R}^m$ and $\boldsymbol{\mu} \in \mathbb{R}^{2n}$ are the Lagrange multipliers, respectively associated to the equality and to the inequality constraints. Condition (8c) requests the preservation of the (possibly scaled) task, i.e., $s\dot{\boldsymbol{x}} = \boldsymbol{J}\dot{\boldsymbol{q}}$. This condition, together with the constraint (8d) are automatically satisfied at every SNS step, thus we do not need to check them.

The so-called complementarity condition (8b) requests zero value for the components of the Lagrange multiplier $\boldsymbol{\mu}$ associated to non-saturated joints (i.e., $\boldsymbol{C}\boldsymbol{\xi} > \boldsymbol{d}$). This allows to extract from eq. (8a) a condition related only to non-saturated joints

$$\boldsymbol{H}\boldsymbol{\xi} + \begin{pmatrix} \boldsymbol{J}\boldsymbol{W} & \dot{\boldsymbol{x}} \end{pmatrix}^T \boldsymbol{\lambda} = \boldsymbol{0}. \tag{9}$$

Note that the SNS selection matrix $\boldsymbol{W}$ is useful here to extract from $\boldsymbol{A}$ only the non-saturated joints. The Lagrange multiplier $\boldsymbol{\lambda}$ can be obtained as

$$\boldsymbol{\lambda} = -\begin{pmatrix} (\boldsymbol{J}\boldsymbol{W})^T \\ \dot{\boldsymbol{x}}^T \end{pmatrix}^{\#} \boldsymbol{H}\boldsymbol{\xi}. \tag{10}$$

It should be noted that, thanks to the SNS algorithm, the pseudoinversion in (10) is always feasible, because the possibility that $\boldsymbol{J}\boldsymbol{W}$ is rank deficient is already checked inside the SNS algorithm. Therefore, since eq. (9) has a solution, there exist a Lagrange multiplier $\boldsymbol{\mu}$, with zeros for the elements associated to non active constraints, that satisfy the KKT condition (8b).

At this point $\boldsymbol{\mu}$ can by computed by imposing the satisfaction of the KKT condition (8a), using the $\boldsymbol{\lambda}$ obtained with eq. (10)

$$\bar{\boldsymbol{\mu}} = -\left(\boldsymbol{I} - \boldsymbol{A}^T \begin{pmatrix} (\boldsymbol{J}\boldsymbol{W})^T \\ \dot{\boldsymbol{x}}^T \end{pmatrix}^{\#}\right) \boldsymbol{H}\boldsymbol{\xi}, \tag{11}$$

and taking into account the fulfillment of condition (8b)

$$\begin{array}{ll} \mu_i = \mu_{i+n} = 0 & \text{if } W_{i,i} = 1 \\ \mu_i = -\bar{\mu}_i,\ \mu_{i+n} = 0 & \text{if } W_{i,i} = 0 \text{ AND } \dot{q}_i = \dot{Q}_{min,i} \\ \mu_i = 0,\ \mu_{i+n} = \bar{\mu}_i, & \text{if } W_{i,i} = 0 \text{ AND } \dot{q}_i = \dot{Q}_{max,i} \\ \text{for } i = 1, \ldots, n & \end{array} \tag{12}$$

Note that eq. (11) does not request expensive computation since the pseudoinversion can be obtained by appending a row (rank one update) to $(\boldsymbol{J}\boldsymbol{W})^{\#}$, which is already computed inside the SNS algorithm.

Summarizing, conditions (8b), (8c) and (8d) are satisfied directly by the SNS algorithm, condition (8a) is imposed using eq. (11), thus only condition (8e) has to be evaluated to check the optimality of the SNS solution.

A simplification of the optimality checking can be done by considering that the SNS algorithm is able to converge to a task scaling factor near to the maximum. Then we can remove the checking for the task scaling factor, by considering the simplified QP problem

$$\begin{array}{ll} \min\limits_{\dot{\boldsymbol{q}}} & \frac{1}{2}\dot{\boldsymbol{q}}^T\dot{\boldsymbol{q}} \\ \text{s.t.} & s\dot{\boldsymbol{x}} = \boldsymbol{J}\dot{\boldsymbol{q}}, \quad \boldsymbol{C}\dot{\boldsymbol{q}} \le \boldsymbol{d} \end{array} \tag{13}$$

where s is the scale factor obtained inside the SNS. The considerations conducted in Sect. VII will show that this simplification does not worsen the algorithm's performance.

Setting $\boldsymbol{\xi} = \dot{\boldsymbol{q}}$, $\boldsymbol{H} = \boldsymbol{I}$, $\boldsymbol{A} = \boldsymbol{J}$, $\boldsymbol{b} = s\dot{\boldsymbol{x}}$ and $\boldsymbol{C} = \begin{pmatrix} -\boldsymbol{I} & \boldsymbol{I} \end{pmatrix}^T$, the KKT criteria (8) applied to the QP problem (13) brings to the same consideration of the previous case. Namely, conditions (8b), (8c) and (8d) are satisfied directly by the SNS algorithm, and condition (8a) is imposed to compute $\boldsymbol{\lambda}$. Thus the only needed check is the positiveness of the components of the Lagrange multiplier $\boldsymbol{\mu}$, which can be computed using relation (12), with

$$\bar{\boldsymbol{\mu}} = -\tilde{\boldsymbol{P}}^T\dot{\boldsymbol{q}}, \tag{14}$$

where

$$\tilde{\boldsymbol{P}} = \boldsymbol{I} - (\boldsymbol{J}\boldsymbol{W})^{\#}\boldsymbol{J}, \tag{15}$$

is the projector to the task null space, using only non-saturated joints. The computation of eq. (14) is very fast, since $\tilde{\boldsymbol{P}}$ is already computed inside the SNS step,

$$\dot{\boldsymbol{q}} = (\boldsymbol{J}\boldsymbol{W})^{\#}\dot{\boldsymbol{x}} + \tilde{\boldsymbol{P}}\dot{\boldsymbol{q}}_N, \tag{16}$$

which is obtained by suitably rearranging the SNS joint velocity computation (see Algorithm 1).

IV. The Optimal SNS

In the previous section we proposed a method to check whether a solution of the SNS algorithm is optimal or not. The information given by the KKT conditions permits also to detect if a saturated joint should not be in the optimal active set. Namely if $\mu_i < 0$ the i-th joint is not in the optimal active set, i.e., at the optimal solution it is not saturated. Therefore the i-th joint can be removed from the saturation, $W_{ii} = 1\,, \dot{q}_{N,i} = 0$.

The Optimal SNS (Opt-SNS) is derived from the standard SNS, as proposed in **Algorithm 3**.

Algorithm 3 (The Opt-SNS algorithm)

```
$\boldsymbol{W} = \boldsymbol{W}^-$, $\dot{\boldsymbol{q}}_N = \boldsymbol{0}$, $s = 1$, $s^* = 0$
repeat
  ... (same code of the SNS Algorithm 1)
  $\tilde{\boldsymbol{P}} = \boldsymbol{I} - (\boldsymbol{J}\boldsymbol{W})^{\#}\boldsymbol{J}$
  $\bar{\boldsymbol{\mu}} = -\tilde{\boldsymbol{P}}^T\dot{\boldsymbol{q}}$
  for $i = 1 \to n$ do
    if $W_{ii} = 0$ then
      if $\left\{ \begin{array}{c} \dot{q}_i = \dot{Q}_{min,i} \text{ .AND. } \bar{\mu}_i > 0 \\ \text{.OR.} \\ \dot{q}_i = \dot{Q}_{max,i} \text{ .AND. } \bar{\mu}_i < 0 \end{array} \right\}$ then
        $W_{ii} = 1$, $\dot{q}_{N,i} = 0$
        limit_exceeded = TRUE
      end if
    end if
  end for
until limit_exceeded = TRUE
$\boldsymbol{W}^- = \boldsymbol{W}$, $\dot{\boldsymbol{q}}_{SNS} = \dot{\bar{\boldsymbol{q}}}$
```

A joint can enter in the active set (saturated) if it is the most critical one for the current solution (i.e., the joint which request the higher task scaling), and it is removed from the active set if the associated Lagrange multiplier μ_i is negative. Note that it is needed to compute only $\bar{\boldsymbol{\mu}}$, because the conversion in $\boldsymbol{\mu}$ Eq. (12) is considered inside the *if* statement.

The algorithm stops when the optimal solution is reached. For this reason a more consistent name for the variable *limit_exceeded*, in Algorithm 3, would be *non_optimal_solution*.

Note that the active set represented by the selection matrix $\boldsymbol{W}$ is initialized with the optimal active set obtained at the previous execution of the algorithm, while in the standard

SNS it is initialized with an empty active set. This is feasible thanks to the Opt-SNS capability of removing a joint from the active set, while the standard SNS is able only to insert a saturated joint in the active set. This initialization reduces the number of internal steps of the algorithm, because, if the desired task is smooth, the optimal solutions for two consecutive points of the trajectory will be close in the joint velocity space. Therefore, they will have a similar active set.

V. Pseudoinverse Computation

The most time-consuming step in SNS and in Opt-SNS is computing both the pseudoinverse and the rank of $\boldsymbol{JW}$, an operation that needs to be repeated each time the active set changes. Therefore, for the computation of the pseudoinverse it is convenient to adopt a method that it also rank revealing. Moreover, since pseudoinversion will be applied repeatedly on matrices that differ just by one or few columns, fast rank-one updates of the pseudoinverse are an appealing choice. In fact, for humanoids robots with a large number of DOFs (say, 20–40), rank-one updates allow real-time computations and should be used together with a RRQR (rank-revealing QR) decomposition [13]. On the other hand, we experienced no benefit for robots with few DOFs (3–8), such as manipulators: the computational time is roughly the same as that for recomputing the pseudoinverse from scratch. In this second case, we suggest the use of *thin* SVD (Singular Value Decomposition) of matrix $\boldsymbol{WJ}^T = \boldsymbol{U\Sigma V}^T$, which is faster than a complete SVD. Pseudoinversion is then performed as $(\boldsymbol{JW})^\# = \boldsymbol{U\Sigma}^{-1}\boldsymbol{V}^T$, and the condition rank $(\boldsymbol{JW}) < m$ occurs when the mth singular value $\sigma_m < \epsilon$, with $\epsilon > 0$ depending on the numerical approximation used.

VI. Simulation Results

The Opt-SNS has been tested using a kinematic model of the KUKA LWR IV robot ($n = 7$). All joint range limits are symmetric $\boldsymbol{Q}_{max} = (170, 120, 170, 120, 170, 120, 170)$ [deg] $= -\boldsymbol{Q}_{min}$ and the maximum joint velocities are $\boldsymbol{V}_{max} = (100, 110, 100, 130, 130, 180, 180)$ [deg/s]. Further, a maximum acceleration $A_{max} = 300$ [deg/s^2] has been chosen, equal for all joints. The sampling time is $T = 1$ [ms].

The capabilities of the SNS have been shown in the simulations and experiments presented in [1], [2]. To illustrate Opt-SNS improvements, we present the case of a task in which the robot end-effector moves through three Cartesian points along linear paths and with constant speed $V = 2$ [m/s]. For the next generic position $\boldsymbol{x}_r$, the task velocity is specified by

$$\dot{\boldsymbol{x}} = V\frac{\boldsymbol{x}_r - \boldsymbol{x}}{\|\boldsymbol{x}_r - \boldsymbol{x}\|}.$$

The robot starts at $\boldsymbol{q}(0) = (0, 45, 45, 45, 0, 0, 0)$ [deg]. This particular task has been chosen because it clearly shows the non-optimality of the standard SNS (Fig. 1).

The simulations are performed in Matlab® on a Intel i7-2600 3.4 GHz with 4 Gb of RAM. We compared the Opt-SNS with the standard SNS and with a state-of-the-art QP solver. For a meaningful comparison, the standard SNS computes its solution in parallel with the Opt-SNS but is not used to control motion. In the QP formulation (5) associated to the Opt-SNS, we set $M = 10^{10}$. The QP solver is the active set method implemented in the Optimization Toolbox™ of Matlab®.

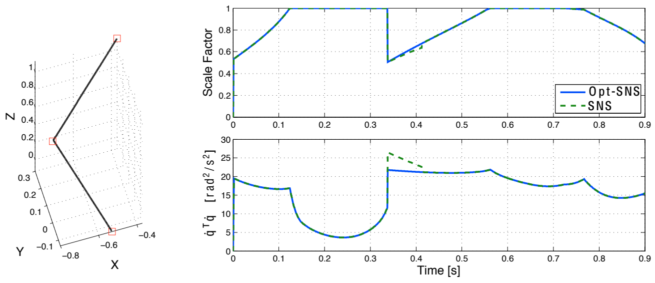

Fig. 1. Simulation 1: Task scaling factor (top right) and squared norm of the joint velocity (bottom right) obtained using the Opt-SNS (solid, blue) or the standard SNS (dashed, green), and the resulting end-effector trajectory (left). The QP solver produces the same results of the Opt-SNS

Figure 1 shows the results of the first simulation. The end-effector path is composed by perfect straight lines through the desired points. From the task scaling factor and the quadratic norm of the joint velocity, we see that the Opt-SNS provides the same results of the QP solver (i.e., the optimal solution) even when using eq. (14) instead of (11) to compute the Lagrange multiplier $\boldsymbol{\mu}$. As expected, the distance from the optimum with the standard SNS is quite small for the task scale factor, but may become non-negligible for the norm of the joint velocity (around $t = 0.34$ s).

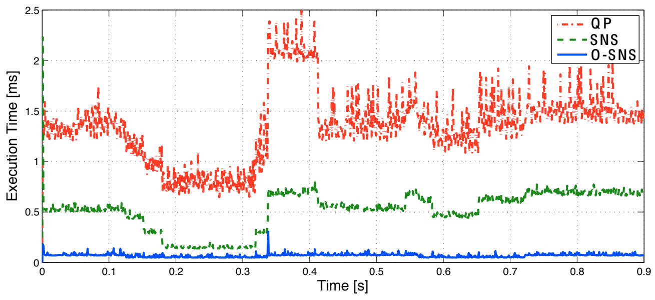

Fig. 2. Simulation 1: Execution times with the QP solver (dot-dashed, red), the standard SNS (dashed, green), and the Opt-SNS (solid, blue)

The execution times of the three algorithms during robot motion is shown in Fig. 2. The Opt-SNS is about 10 times faster than the standard SNS and 20 times faster than the QP solver. The average values of the task scaling s, of the squared norm of the joint velocity $\|\dot{\boldsymbol{q}}\|^2$, and of the execution times are reported in Tab. I. Note that both the Opt-SNS and the QP solver are initialized at each call with the active set obtained at the end of the previous one. While Matlab® was used in simulations to compare our method with an established version of the state-of-the-art QP solver, we expect indeed better performance with C++ implementations.

In the second simulation (see Fig. 3), the task is a point-to-point motion for the robot end-effector from the start configuration to the Cartesian point $\boldsymbol{x}_r = \begin{pmatrix} 0.7 & 0.15 & 0.2 \end{pmatrix}^T$

TABLE I
AVERAGE VALUES IN SIMULATION 1

Solver	s	$\dot{\boldsymbol{q}}^T\dot{\boldsymbol{q}}$ [rad^2/s^2]	exec. time [ms]
Opt-SNS	0.87817	15.608	0.0698
SNS	0.87672	15.862	0.514
QP	0.87817	15.608	1.4

[m]. This simple task puts in evidence a drawback of the considered formulation of the optimization problem, which affects equally all solution methods. As evident from the joint velocity behaviors in Fig. 4, even if the desired task is smooth the optimal joint velocities are not. The joint velocities shown in Fig. 4 are those obtained both with the Opt-SNS and with the QP solver. Therefore, this discontinuous behavior is not a result of the chosen Opt-SNS algorithm, but it is intrinsic to the optimal solution of the formulated problem. This phenomenon was noticed also in [1], [2], where we proposed the transposition of the algorithm to the acceleration level in order to recover continuous velocities.

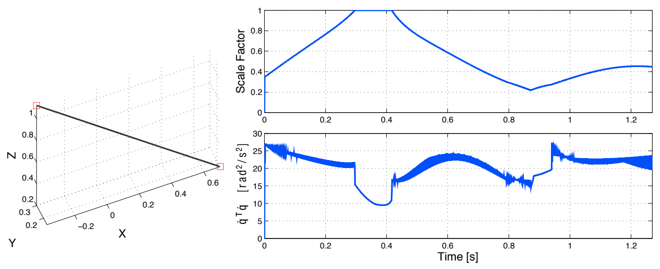

Fig. 3. Simulation 2: Task scaling factor (top right) and squared norm of the joint velocity (bottom right) obtained using the Opt-SNS, and the resulting end-effector trajectory (left). The same result is obtained with the QP solver

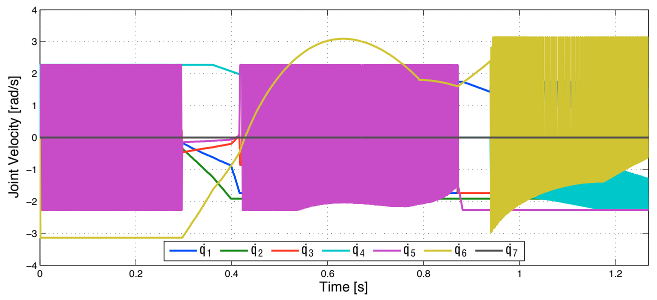

Fig. 4. Simulation 2: A drawback of the optimization problem is the possibility of discontinuous joint velocity for a continuous desired task

VII. RECOVERING CONTINUITY

Two main sources of potential discontinuities of the joint velocity commands have been identified. The first is due to the separate imposition of velocity bounds and joint range limits. When a robot joint reaches its limit with a velocity which is still below its maximum bound, a sudden velocity jump occurs due to the desire of preserving a

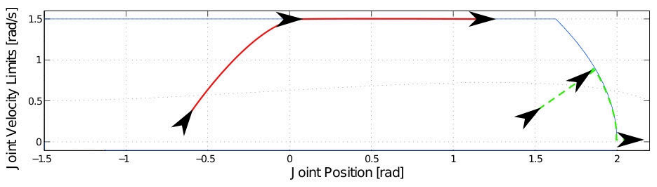

Fig. 5. Example of joint saturation: Joint velocity bound (blue), reaching the maximum joint velocity (solid, red), and reaching the joint limit (dashed, green). In both cases there are no jumps on the realizable joint velocity

feasible motion. This effect is eliminated in our method through the shaping of the velocity constraint in eq. (3), which dynamically combines all hard bounds (2) on position, velocity and acceleration (see [1] for details). The effect of this constraint shaping is sketched in Fig. 5, with the joint velocity constraint being smoothly activated, both when $|\dot{q}_i| \to V_i$ and when $q_i \to Q_{min/max}$.

The second source of discontinuity is due to the need of task scaling. In fact, we observed in all our case studies that the joint velocity solution is never discontinuous when the task is feasible, even when multiple saturations occur. A possible explanation of this fact follows from the structure of the objective function in the QP problem (5). In fact, the QP solver guarantees that the optimal value of the objective is continuous for continuous tasks. However, since it is required that $M \gg 1$, this continuity is reflected almost completely on the task scaling factor alone, while discontinuities cannot be excluded for $\dot{\boldsymbol{q}}$ (as can be recognized in the plots of Fig. 3).

To recover continuity, we introduce a slightly conservative margin $s_m > 0$ with respect to the optimal task scaling factor. The following relaxation procedure is used:

1) execute the Opt-SNS with maximum admissible task scaling equal to $1+s_m$ (rather than 1) to find a modified optimal task scaling factor s^*;
2) relax the desired task as $\dot{\boldsymbol{x}}_m = (s^* - s_m)\dot{\boldsymbol{x}}$;
3) apply the Opt-SNS for the relaxed task $\dot{\boldsymbol{x}}_m$.

Note that the relaxed task velocity $\dot{\boldsymbol{x}}_m$ is certainly feasible. Figure 6 shows the joint velocities obtained with this relaxation procedure for the task considered in Fig. 3 and using $s_m = 0.1$. The robot performs the task, a linear path of length 1.4133 [m], slightly slower (1.278 rather than 1.269 s), but velocity discontinuities are completely removed.

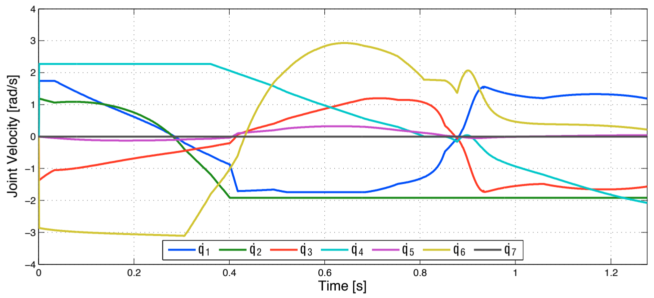

Fig. 6. Simulation 2: Joint velocities obtained with task scaling relaxation

The best value of the scaling margin s_m depends on the kinematics of the manipulator and on the desired task. Here, we have just set it to a value that was sufficient to avoid any discontinuity in all performed simulations, but indeed further investigation is needed. Anyway, by construction, the larger is the margin s_m, the smaller is the probability of running into discontinuities. Therefore, it is always possible to find a sufficiently large value that is suitable for a specific application.

VIII. Extension to the Multi-task Case

In [2] we presented the extension of the SNS algorithm to the prioritized multi-task case. To extend similarly the Opt-SNS, we need to evaluate the KKT conditions (8) for a QP problem that takes into account task priorities. Consider l tasks $\dot{\boldsymbol{x}}_k$ with Jacobians $\boldsymbol{J}_k$, for $k = 1, \dots, l$. The optimization problem for the kth task is

$$\begin{aligned} \min_{\dot{\boldsymbol{q}}_k} \quad & \tfrac{1}{2}\, \dot{\boldsymbol{q}}_k^T \dot{\boldsymbol{q}}_k \\ \text{s.t.} \quad & \dot{\boldsymbol{X}}_{A,k} = \boldsymbol{J}_{A,k}\dot{\boldsymbol{q}}_k, \quad \boldsymbol{C}\dot{\boldsymbol{q}}_k \leq \boldsymbol{d} \end{aligned} \tag{17}$$

where

$$\dot{\boldsymbol{X}}_{A,k} = \begin{pmatrix} s_1\dot{\boldsymbol{x}}_1 \\ s_2\dot{\boldsymbol{x}}_2 \\ \vdots \\ s_k\dot{\boldsymbol{x}}_k \end{pmatrix}, \quad \boldsymbol{J}_{A,k} = \begin{pmatrix} \boldsymbol{J}_1 \\ \boldsymbol{J}_2 \\ \vdots \\ \boldsymbol{J}_k \end{pmatrix} \tag{18}$$

are the augmented scaled task and the augmented Jacobian, and with $\dot{\boldsymbol{q}}_k = \dot{\bar{\boldsymbol{q}}}_k$ denoting the solution up to the kth task. The joint constraints are the same of the single task case. As in Sect. III, the evaluation of the KKT conditions is reduced to the evaluation of the Lagrangian multiplier $\boldsymbol{\mu}_k$, since all other conditions are guaranteed by the SNS algorithm.

Considering the same approach used for the single task case is not surprising that we obtain a similar results, indeed

$$\bar{\boldsymbol{\mu}}_k = -\tilde{\boldsymbol{P}}_k^T \dot{\bar{\boldsymbol{q}}}_k\,, \tag{19}$$

where $\tilde{\boldsymbol{P}}_k$ is the projector in the null space of the augmented Jacobian (i.e., for all k tasks) that uses only non-saturated joints

$$\tilde{\boldsymbol{P}}_k = \left(\boldsymbol{I} - \left(\boldsymbol{J}_k\bar{\boldsymbol{P}}_k\right)^{\#}\boldsymbol{J}_k\right)\left(\left(\boldsymbol{I} - \boldsymbol{W}_k\right)\boldsymbol{P}_{k-1}\right)^{\#} \tag{20}$$

where

$$\bar{\boldsymbol{P}}_k = \left(\boldsymbol{I} - \left(\left(\boldsymbol{I} - \boldsymbol{W}_k\right)\boldsymbol{P}_{k-1}\right)^{\#}\right)\boldsymbol{P}_{k-1}, \tag{21}$$

and $\boldsymbol{P}_k$ is the null space of the augmented Jacobian obtained recursively as

$$\begin{aligned} \boldsymbol{P}_0 &= \boldsymbol{I} \\ \boldsymbol{P}_k &= \boldsymbol{P}_{k-1} - \left(\boldsymbol{J}_k\boldsymbol{P}_{k-1}\right)^{\#}\left(\boldsymbol{J}_k\boldsymbol{P}_{k-1}\right). \end{aligned} \tag{22}$$

The computational cost of evaluating (19) for the optimality check is negligible, since $\tilde{\boldsymbol{P}}_k$ is already computed inside the multi-task SNS algorithm. This can be seen by considering that the method proposed in [2] can be written to compute the joint velocity as

$$\dot{\bar{\boldsymbol{q}}}_k = \left(\boldsymbol{J}_k\bar{\boldsymbol{P}}_k\right)^{\#}\left(s_k\dot{\boldsymbol{x}}_k - \boldsymbol{J}_k\dot{\bar{\boldsymbol{q}}}_{k-1}\right) + \tilde{\boldsymbol{P}}_k\dot{\boldsymbol{q}}_{N,k}. \tag{23}$$

We note that $\tilde{\boldsymbol{P}}_k$ in (23) plays the same role as $\tilde{\boldsymbol{P}}$ in (16).

IX. Conclusions

We have introduced Opt-SNS, a novel and efficient algorithm for computing an optimal solution to the inverse differential kinematic problem for redundant robots, in the presence of hard joint constraints that cannot be relaxed. Opt-SNS improves the former SNS algorithm and guarantees the generation of an optimal solution that has minimum norm for the joint velocity and that allows the minimum task scaling.

The proposed method has been compared with a state-of-the-art QP solver, showing that the Opt-SNS provides the same results (i.e., the optimum) but with a computational cost that is about 20 times smaller. This promotes the Opt-SNS algorithm to become an appropriate method for real-time control applications.

The problem of joint velocity discontinuities encountered with the SNS has been solved by introducing a conservative margin in the task scaling procedure. Finally, the extension to the multi-task case was presented. Future work will address further improvements in computational times, working out on some peculiar aspects of the Opt-SNS, and consider different types of constraints (e.g., inequalities in the Cartesian space).

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

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

[6] J. Nocedal and S. Wright, *Numerical Optimization*, 2nd ed. Springer-Verlag, 2006.

[7] F. Cheng, T. Chen, and Y. Sun, "Resolving manipulator redundancy under inequality constraints," *IEEE Trans. on Robotics and Automation*, vol. 10, pp. 65–71, 1994.

[8] F. Cheng, R. Sheu, and T. Chen, "The improved compact QP method for resolving manipulator redundancy," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 25, pp. 1521–1530, 1995.

[9] 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.

[10] O. Khanoun, 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] H. W. Kuhn and A. Tucker, "Nonlinear programming," in *Proc. of 2nd Berkeley Symposium*, 1951.

[13] P. Businger and G. Golub, "Linear least squares solution by Householder transformations," *Numerische Mathematik*, vol. 7, no. 3, pp. 269–276, 1965.