Control of Redundant Robots under Hard Joint Constraints: Saturation in the Null Space

Fabrizio Flacco, *Member, IEEE,* Alessandro De Luca, *Fellow, IEEE,* and Oussama Khatib, *Fellow, IEEE*

***Abstract*—We present an efficient method for addressing online the inversion of differential task kinematics for redundant manipulators, in the presence of hard limits on joint space motion that can never be violated. The proposed SNS (Saturation in the Null Space) algorithm proceeds by successively discarding the use of joints that would exceed their motion bounds when using the minimum norm solution. When processing multiple tasks with priority, the SNS method realizes a preemptive strategy by preserving the correct order of priority in spite of the presence of saturations. In the single- and multi-task case, the algorithm automatically integrates a least possible task scaling procedure, when an original task is found to be unfeasible. The optimality properties of the SNS algorithm are analyzed by considering an associated Quadratic Programming problem. Its solution leads to a variant of the algorithm, which guarantees optimality also when the basic SNS algorithm does not. Numerically efficient versions of these algorithms are proposed. Their performance allows real-time control of robots executing many prioritized tasks with a large number of hard bounds. Experimental results are reported.**

***Index Terms*—Redundant robots, motion control, inverse differential kinematics, hard joint constraints, saturation of commands, optimal joint velocity.**

I. Introduction

INVERSION of the differential task kinematics is the standard way to command the joint motion of redundant robots [1]. Kinematic control methods use typically a generalized inverse (most often, the pseudoinverse) of the task Jacobian in order to convert velocity or acceleration commands from the task (e.g., Cartesian) space to the joint space, where actuation takes place. Redundancy is exploited locally for collision avoidance, for joint motion optimization, or for augmenting the primary task with multiple additional ones, possibly with priority [2]. All these approaches can be casted as the selection of suitable joint commands in the null space of the task Jacobian.

In this framework, *hard limits* imposed on joint space motion, such as limited joint range and bounds on velocity, acceleration, or even torque, are barely taken into account, at least in an explicit way. The underlying assumption is that either the joint motion and/or the actuator capabilities can be considered unlimited in practice, or that the robot task has been smoothly tailored in space and scaled in time so as to fit to the robot limitations in a conservative fashion. However, for sensor-driven robotic tasks in dynamic environments, in particular during physical human-robot interaction (pHRI), it is not unlikely that large instantaneous task velocities or accelerations are suddenly requested in response to an unexpected situation. These may lead to nominal commands in the joint space exceeding some bounds, with an associated saturation that makes the resulting robot motion unpredictable.

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 critical also for robots with many degrees of freedom (DOFs). In fact, when trying to exploit DOFs for the simultaneous execution of a large multiplicity of tasks, each of which not highly demanding but jointly exceeding the robot capabilities, saturation of some joint commands may lead to an uncontrolled relaxation of crucial high priority tasks, and thus, e.g., to physical collisions with the environment or loss of balance for a humanoid robot.

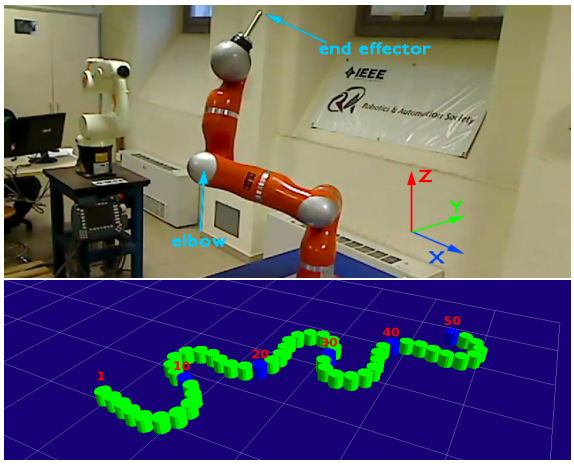

Fig. 1. Two sample robots that have been controlled in real time with the SNS algorithm for executing multiple prioritized tasks under hard joint limits. [Top] The 7-DOF KUKA LWR 4 in the DIAG Robotics Lab, used for the numerical results of Sec. VIII and the experiments of Sec. IX. [Bottom] A 50-DOF planar snake robot, used for the performance evaluation of Sec. X.

A. Previous works

Hard limits in the joint space have often been converted into soft ones, resolving redundancy via task constrained optimization of suitable objective functions, e.g., keeping the joints closer to their range center [3], [4], using the joint ranges in a weighted pseudoinversion [5], or defining an infinity norm to be minimized at the velocity level [6]. However, the commanded joint motion may still violate some of the bounds, and its saturation produces then a wrong instantaneous behavior in the task space.

F. Flacco and A. De Luca are with Dipartimento di Ingegneria Informatica, Automatica e Gestionale, Sapienza Università di Roma, Via Ariosto 25, 00185 Rome, Italy ({fflacco,deluca}@diag.uniroma1.it).

O. Khatib is with the Artificial Intelligence Laboratory, Stanford University, Stanford, CA 94305, USA (khatib@cs.stanford.edu).

This work is supported by the European Commission, within the FP7 ICT-287513 SAPHARI project (www.saphari.eu).

A simple way to recover feasibility with respect to the given joint bounds is by scaling the task in time[1], i.e., reducing the speed/acceleration of the (single or multiple) task commands. Relaxation by task scaling has been used for satisfying joint velocity [7] and/or acceleration [8] bounds. In [9], the velocity term in a one-dimensional null space of a 7-DOF robot is scaled so as to satisfy joint velocity bounds, if at all possible. For multiple tasks, prioritization may still be preserved (see, e.g., [10]), using again the mechanism of projection in the null space of the task Jacobians. Before resorting to task scaling, one should verify whether alternative joint motions can be generated that still execute the original task while satisfying the hard joint limits (and preserving also prioritization, in case of multiple tasks). This obviously requires a smart exploitation of the null space of the task Jacobian(s).

The idea of partitioning saturated and non-saturated commands was used in [11] for a single task under velocity and acceleration constraints, and in [12], within a visual servoing problem with joint range limits. An implicit task scaling was used in [12], modifying when needed the control gain on the position task error. Instead, a joint velocity scaling procedure followed by a time warping was proposed in [11], leading to an approximate solution as soon as two or more accelerations are saturated. Another method that explicitly handles joint velocity or acceleration bounds in a redundant robot performing a single task has been introduced in [13]. All joint commands exceeding their bounds are pushed back at their saturation levels. This effect is then compensated by the selection of a null space contribution for satisfying the task. Beside their restricted scope (single task, no optimality), these techniques share a common drawback, namely the *simultaneous* treatment of all overdriven joints at the same time. This may lead to a premature ending of the algorithm (whereas a feasible solution exists) or to the introduction of unnecessary task scaling.

A similar approach has been proposed for the case of multiple tasks in the animation of avatars [14], assuming the presence of joint range limits alone. Feasibility with respect to these limits is verified only after adding the contributions of all tasks, which is not computationally efficient. Moreover, when the whole set of tasks cannot be realized within the given limits, the deformed execution is spanned over all tasks and does not remain confined just to the low priority ones. .

The considered problem can also be tackled as the constrained minimization of a quadratic objective function under linear equality/inequality constraints. If the desired task can be accomplished without saturation of the inequality constraints, the optimal solution in terms of minimum joint velocity norm is obtained by pseudoinversion of the task Jacobian [1]. Equality constraints are usually taken into account by task augmentation [15], while inequality constraints are embedded in an objective function that is optimized by the projected gradient method [16]. However, this combination does not always guarantee satisfaction of the inequality constraints, nor the optimality of joint velocity solution.

Constrained optimization problems with inequalities can be solved using an active-set method [17], where the active set is composed by those constraints that are satisfied as equalities (e.g., saturated joint position variables in our robotic application). The optimization algorithm will find the optimal active set that leads to the constrained optimal solution. Unfortunately, due to their high computational costs, such general-purpose algorithms cannot be used as such for real-time applications and extra strategies should be implemented for the problem at hand. For instance, the constrained redundancy resolution was formulated as a Quadratic Programming (QP) problem in [18], and a Gram-Schmidt orthogonalization procedure was applied for its solution. A compact QP method using Gaussian elimination with partial pivoting was developed in [19]. The computational complexity of these methods is reduced, but it is still too high when the number of DOFs increases. Moreover, the cases of unfeasible tasks or multiple prioritized tasks are not addressed.

In [20], constrained kinematic inversion of tasks was modeled and addressed in a general way as an extended QP problem, covering linear equalities and inequalities and also a possible hierarchy of prioritized tasks. When resorting to state-of-the-art QP solvers, the computational cost of the solution remained prohibitive for robots with a large number of DOFs. Based on a similar stack of QP problems, the solver was improved in [21] by using a complete orthogonal decomposition to obtain the null spaces of tasks with a prioritized hierarchy. This improvement allowed a satisfactory real-time implementation of the method [22]. A common benefit of [20]–[22], as well as of other generic QP solvers, is the possibility of considering all constraints in a unified framework. Nonetheless, in the case of hard joint inequality constraints, no advantage is taken from the particular structure of the problem, resulting in computationally less efficient algorithms than those presented here.

B. Paper contribution and organization

In this paper, we present the Saturation in the Null Space (SNS) method, consolidating and expanding the results of our more recent works [23]–[26].

In our method, all existing limits in the joint space are first combined into *hard constraints* on the joint velocity commands at the current robot configuration. The initial pseudoinverse solution is iteratively modified 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.

The *basic SNS* algorithm automatically takes care of tasks that are unfeasible for the given motion capabilities of the robot. In fact, a *task scaling* procedure is integrated which allows the robot to execute at least the directional part of the task in the desired way (e.g., if the task is an end-effector trajectory, the traced geometric path will remain the same). Moreover, in the case of a stack of prioritized tasks, each expressed locally by linear equality constraints, the SNS

[1]In view of the linearity of the forward differential map at the velocity level, scaling motion time by c (i.e., $t \rightarrow ct$) and task or joint scaling by a factor $s = 1/c$ are equivalent when only one single task is of concern. However, when multiple tasks are considered, as in this paper, the term task scaling is preferred because it implies the possibility of scaling each task differently.

algorithm enforces a *preemptive strategy*: higher priority tasks are preserved (up to a scaling, if needed) by using all the available robot capabilities they need, while lower priority tasks are accommodated (and scaled, if needed) with the residual robot capabilities, and without ever interfering with the execution of higher priority tasks[2].

By reformulating the SNS method in the context of quadratic programming with linear task equalities and linear inequality constraints (of the box type), we can modify the basic method so as to guarantee optimality of the solution found (the *Opt-SNS* algorithm). Essentially, this involves tuning the order in which the single overdriven commands are being saturated, based on the analysis of the multipliers. Finally, the numerical performance of SNS algorithms can be improved by using similar machinery as in [21], [22]. For this, the problem is reformulated based on task augmentation [27], and information on saturated commands is inserted in the form of additional equality constraints (this is quite similar to the active set idea). A QR factorization of the relevant matrices involved is used to speed up computations, leading respectively to the *Fast-SNS* and *FastOpt-SNS* algorithms.

The paper is organized as follows. The redundancy formalisms used for single and multiple tasks throughout the paper are summarized in Sec. II. In Sec. III, the basic SNS algorithm is introduced for a single task at the velocity control level. Section IV deals with the optimal version of the SNS algorithm, while a numerical problem of discontinuity in the optimal joint velocity solution is addressed and solved in Sec. V. The extension to the multiple task case is presented in Sec. **??**. Section VII considers the reformulation of the SNS method so as to obtain a faster implementation, exploiting the special structure of the problem. Simulation and experimental results for a KUKA LWR 4 robot (see Fig. 1) are reported in Sec. VIII and IX, respectively. Section X is devoted to a numerical comparison conducted on the high-DOF snake robot shown in Fig. 1, in order to evaluate the real-time performance of the proposed algorithms.

With respect to our conference papers [23]–[26], many new contributions have been included here. A common formalism has been used for all SNS versions, with a more intuitive presentation of the basic algorithm. Further properties of the method and computational issues are described in Sec. III-C. Section VI-B shows how to modify the SNS algorithm for dealing with singularities. With respect to [23], the simulation results in Sec. VIII refer now to smoother trajectories. Last but not least, the experimental results of Sec. IX are new.

II. Redundancy Resolution

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

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

[2]This is the usual principle of the Task Priority (TP) method. We assume that the reader is familiar with the fact that the correct order of task priorities can be destroyed by command saturations when using a standard TP algorithm. See the illustrative example in [24].

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

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

where $\boldsymbol{J}^{\#}$ is the (unique) Moore-Penrose pseudoinverse [28] of the task Jacobian, $\boldsymbol{P} = \boldsymbol{I} - \boldsymbol{J}^{\#}\boldsymbol{J}$ is the $n \times n$ orthogonal projector in the Jacobian null space, and $\dot{\boldsymbol{q}}_{\mathcal{N}} \in \mathbb{R}^n$ is a generic joint velocity. Equation (2) gives the joint velocity $\dot{\boldsymbol{q}}$ that satisfies (1) (or minimizes the norm of the error $\dot{\boldsymbol{x}} - \boldsymbol{J}\dot{\boldsymbol{q}}$, if $\dot{\boldsymbol{x}}$ is not in the range of $\boldsymbol{J}$), while minimizing in norm the distance to $\dot{\boldsymbol{q}}_{\mathcal{N}}$. The solution with minimum norm is obtained for $\dot{\boldsymbol{q}}_{\mathcal{N}} = \boldsymbol{0}$.

Use of redundancy can be extended to the execution of l tasks in the form (1), $\dot{\boldsymbol{x}}_k = \boldsymbol{J}_k\dot{\boldsymbol{q}}$, each of dimension m_k, $k = 1, \ldots, l$ (usually, 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 if $i < j$. 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 solution to the k-th task of the stack in the null space of all higher priority tasks. This is obtained by using the recursive formula [29]

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

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 [30]

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

Different numerical methods can be used to compute the solution (2) or (3). The most common is to resort to a Singular Value Decomposition (SVD) of the matrix to be pseudoinverted. The SVD provides the singular values of the matrix, and thus its rank and condition number. It is then possible to check if a task is (kinematically or algorithmically) singular or close to a singularity, and in that case to switch to a damped pseudoinverse as approximate solution (also, selectively damping along directions based on the singular values [31]).

However, performing SVD is a computationally expensive step and the use of a QR decomposition is certainly faster [32]. Consider the QR decomposition of the transpose of a generic task Jacobian

$$\boldsymbol{J}^T = \boldsymbol{Q}\begin{pmatrix} \boldsymbol{R} \\ \boldsymbol{0} \end{pmatrix}, \qquad \boldsymbol{Q} = \begin{pmatrix} \boldsymbol{Y} & \boldsymbol{Z} \end{pmatrix}, \tag{6}$$

being $\boldsymbol{Q}$ a $n \times n$ orthogonal matrix, decomposed in a $n \times m$ matrix $\boldsymbol{Y}$ and a $n \times (n-m)$ matrix $\boldsymbol{Z}$, and $\boldsymbol{R}$ a $m \times m$ upper triangular matrix. Assuming for the moment a full rank Jacobian, the solution (2) for a single task is obtained as

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

The columns of $\boldsymbol{Z}$ provide thus a (minimal) basis for the Jacobian null space, and the usual null space projector can be obtained as $\boldsymbol{P} = \boldsymbol{Z}\boldsymbol{Z}^T$. Finally, $\boldsymbol{z}_{\mathcal{N}} \in \mathbb{R}^{n-m}$ is a generic

vector in the *reduced* space of redundant DOFs. The relation $z_{\mathcal{N}} = Z^T \dot{q}_{\mathcal{N}}$ holds.

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

$$(\boldsymbol{J}_k \boldsymbol{Z}_{A,k-1})^T = (\boldsymbol{Y}_{A,k} \quad \boldsymbol{Z}_k) \begin{pmatrix} \boldsymbol{R}_{A,k} \\ \boldsymbol{0} \end{pmatrix}, \tag{8}$$

one obtains

$$\dot{\boldsymbol{q}}_k = \dot{\boldsymbol{q}}_{k-1} + \boldsymbol{Z}_{A,k-1} \boldsymbol{Y}_{A,k} \boldsymbol{R}_{A,k}^{-T} \left(\dot{\boldsymbol{x}}_k - \boldsymbol{J}_k \dot{\boldsymbol{q}}_{k-1} \right), \tag{9}$$

initialized with $\dot{\boldsymbol{q}}_0 = \boldsymbol{0}$ and $\boldsymbol{Z}_{A,0} = \boldsymbol{I}$. Moreover, a basis for the null space of the augmented Jacobian $\boldsymbol{J}_{A,k}$ is computed recursively as

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

Note that the dimension of $\boldsymbol{J}_k \boldsymbol{Z}_{A,k-1}$ is $m_k \times (n - \sum_{i=1}^{k-1} m_i)$ —the number of columns shrinks with k. Therefore, adding the contribution of lower priority tasks will become faster as we go down the stack [33].

Equations (7) and (9) have been written assuming that a generic task is nonsingular (and that no *algorithmic singularities* [2] occur in the stack up to task k), so that matrices $\boldsymbol{R}$ or $\boldsymbol{R}_{A,k}$ are invertible. When the determinant, e.g., of matrix $\boldsymbol{R}$ (the product of its diagonal elements) is smaller than a given threshold, the task is considered singular, and a solution is obtained through damped pseudoinversion of $\boldsymbol{R}$, achieved using its SVD. This operation is computationally fast, being $\boldsymbol{R}$ a square matrix of dimension m.

The above redundancy resolution schemes implicitly assume that the tasks can always be executed with the given robot capabilities. In other words, joint limits or maximum velocity bounds are not taken into account. When a joint velocity obtained in this way is used to control a robot, command saturation will actually deform the execution of tasks and their priority order may also be compromised. Some illustrative examples of these behaviors have been presented in [23] for the single task case, and in [24] for the multiple task case.

III. The SNS Method

We present here the SNS method in its basic form, assuming that the robot is controlled at the velocity level. In Sec. III-A, constraints on joint velocities are locally shaped taking into account the joint range limits, and the velocity and acceleration bounds. The SNS algorithm, including task scaling, is presented then in Sec. III-B.

A. Shaping the joint velocity constraints

The robot motion capabilities are defined by the following limits imposed on joint space quantities:

$$\boldsymbol{Q}_{min} \leq \boldsymbol{q} \leq \boldsymbol{Q}_{max} \quad \text{(joint range)} \tag{11a}$$

$$\boldsymbol{V}_{min} \leq \dot{\boldsymbol{q}} \leq \boldsymbol{V}_{max} \quad \text{(velocity range)} \tag{11b}$$

$$\boldsymbol{A}_{min} \leq \ddot{\boldsymbol{q}} \leq \boldsymbol{A}_{max} \quad \text{(acceleration range).} \tag{11c}$$

In the following, velocity and acceleration bounds will be taken symmetric[3], $\boldsymbol{V}_{min} = -\boldsymbol{V}_{max}$ and $\boldsymbol{A}_{min} = -\boldsymbol{A}_{max}$. Indeed, these limits should not be violated. However, since the robot is commanded at the joint velocity level, only the first two sets of $4n$ inequalities in (11) can be strictly enforced. In order to handle directly also acceleration (as well as torque) bounds, robot commands should be defined at the second-order differential level —see [23], [24].

At the current configuration $\boldsymbol{q}$, we can derive from (11) the hard (box) constraints on $\dot{\boldsymbol{q}}$,

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

which specify that, for each joint $i = 1, \ldots, n$, the velocity command $\dot{q}_i$ must guarantee that *i)* the joint range limits will not be exceeded in the next step, *ii)* $|\dot{q}_i|$ is smaller than the maximum velocity, and *iii)* the joint will be able to stop its motion before reaching its closest joint limit, taking into account the bounds on maximum acceleration. These three requirements shape the constraints in (12) as follows.

In the control implementation, the joint velocity command $\dot{\boldsymbol{q}}$ is kept constant at the computed value $\dot{\boldsymbol{q}}_h = \dot{\boldsymbol{q}}(t_h)$ for a sampling time of duration T, where $t_h = hT$. Suppose that at $t = t_h$ the current joint position $\boldsymbol{q} = \boldsymbol{q}_h$ is feasible. The next position $\boldsymbol{q}_{h+1} \simeq \boldsymbol{q}_h + \dot{\boldsymbol{q}}_h T$ needs still to be within the joint range limits, and thus

$$\frac{\boldsymbol{Q}_{min} - \boldsymbol{q}_h}{T} \leq \dot{\boldsymbol{q}}_h \leq \frac{\boldsymbol{Q}_{max} - \boldsymbol{q}_h}{T}. \tag{13}$$

Furthermore, suppose that we need to stop robot motion in the fastest possible way, namely by maximally decelerating a joint i which is moving at $\dot{q}_{h,i} > 0$ so as to remain within the available joint range (the following reasoning is specular for the case of maximum acceleration from a negative velocity). For the ith joint subject to $-A_{max,i}$, the position and velocity at a generic $t \geq t_k$ are

$$q_i(t) = q_{h,i} + \dot{q}_{h,i}(t - t_h) - \frac{A_{max,i}}{2} (t - t_h)^2,$$
$$\dot{q}_i(t) = \dot{q}_{h,i} - A_{max,i}(t - t_h).$$

The extreme situation to consider is when the joint reaches its upper limit $Q_{max,i}$ at some $t = t^* > t_h$ with a joint velocity $\dot{q}_i(t^*) = 0$ —the joint stops right at the boundary of its range. It is easy to check that the largest positive velocity of joint i that can be accepted at $t = t_h$ is then upper bounded by

$$\dot{q}_{h,i} \leq \sqrt{2A_{max,i} \left(Q_{max,i} - q_{h,i} \right)}. \tag{14}$$

Similarly, the largest negative velocity is lower bounded by

$$-\sqrt{2A_{max,i} \left(q_{h,i} - Q_{min,i} \right)} \leq \dot{q}_{h,i}. \tag{15}$$

Considering together inequalities (11b) and (13), as well as those given by (14–15) for $i = 1, \ldots, n$, we finally obtain the box constraints (12) for the command $\dot{\boldsymbol{q}}$ at time $t = t_h$, where

$$\dot{Q}_{min,i} = \max\left\{ \tfrac{Q_{min,i} - q_{h,i}}{T}, -V_{max,i}, -\sqrt{2A_{max,i}(q_{h,i} - Q_{min,i})} \right\}$$

is negative and

$$\dot{Q}_{max,i} = \min\left\{ \tfrac{Q_{max,i} - q_{h,i}}{T}, V_{max,i}, \sqrt{2A_{max,i}(Q_{max,i} - q_{h,i})} \right\}$$

is positive. This implies that $\dot{\boldsymbol{q}} = \boldsymbol{0}$ is an admissible command (which is a useful property, e.g., when downscaling the task

[3]This hypothesis is very mild, and in fact not strictly necessary. However, we assume that admissible intervals on differential quantities contain the zero values.

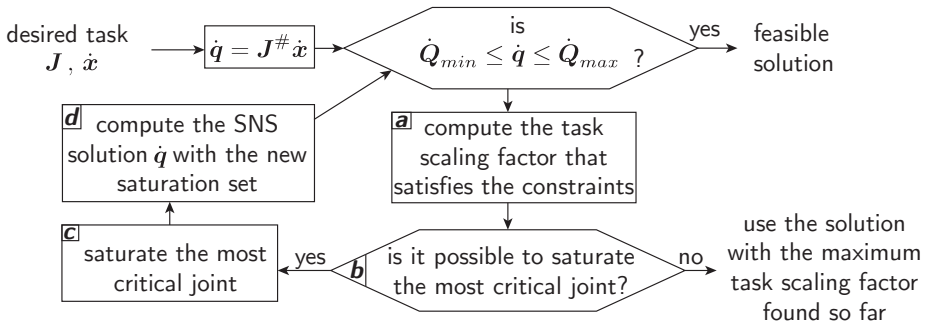

Fig. 3. Schematic representation of the *SNS algorithm*.

velocity). Figure 2 shows an example of the admissible area obtained for a joint velocity command, for some velocity and acceleration bounds, as a function of the joint position within the given range limits.

B. The basic SNS algorithm

With reference to the scheme in Fig. 3, given a desired task specified by the pair $(\boldsymbol{J},\ \dot{\boldsymbol{x}})$ in (1) and the constraints (12), the basic *SNS algorithm* starts by using the minimum velocity norm solution (2). If this solution is admissible for the constraints (12), it is used for controlling the robot. If it exceeds instead some of the hard constraints, then the SNS algorithm is used to find a solution, eventually scaling the task if this is necessary.

Within the algorithm cycle, step **a** (see Fig. 3) considers a new scaled task $s\dot{\boldsymbol{x}}$, for a positive scalar s. Note that, due to the shape of the hard constraints (12), there exists always a scaling factor $s \in [0, 1]$ such that the joint velocity solution is admissible. The task scaling factor is computed for two reasons: *i)* to check if the current solution allows a larger scaling factor than previous solutions, and in such case the current solution is saved as the best so far; *ii)* to evaluate the most critical joint, i.e., the joint whose velocity needs the largest scaling (i.e., the smallest scaling factor s) to stay within the bounds.

The most critical joint is eligible for entering in the *saturation set*, which specifies the joint commands that are forced to their saturation values by the algorithm. This set is coded by the $n \times n$ matrix $\boldsymbol{W} = \mathrm{diag}\{W_{ii}\}$ with $0/1$ elements: if

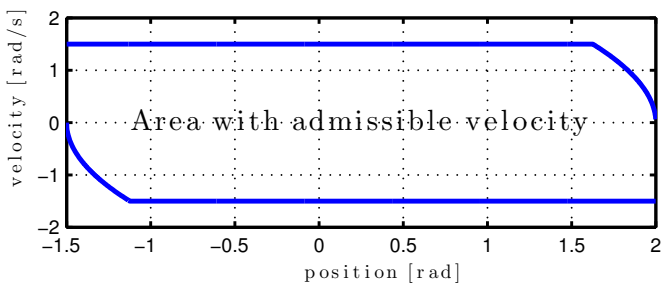

Fig. 2. Shaping a joint velocity constraint: $Q_{min} = -1.5$, $Q_{max} = 2$ [rad], $V_{max} = 1.5$ [rad/s], and $A_{max} = 3$ [rad/s^2].

$W_{ii} = 0$, the velocity of joint i is set at its saturation level and the joint is disabled; otherwise, this joint is still enabled (for norm minimization purposes). If joint j is the most critical joint, it is inserted in the saturation set with $W_{jj} = 0$.

Step **b** in the cycle checks whether it is still possible to execute the task using only the enabled joints, being the velocity of the disabled joints in saturation. From an algebraic point of view, one needs to check if $\mathrm{rank}(\boldsymbol{JW}) \geq m$, being m the dimension of the task. If this check fails, the original task is unfeasible for the given robot capabilities, and the solution with largest task scaling factor (i.e., the s value closest to 1) found so far will be the joint velocity command provided as output. Otherwise, step **c** saturates the most critical joint velocity by setting $W_{jj} = 0$, and the associated saturation value is inserted in the jth component of the null space vector $\dot{\boldsymbol{q}}_N$ (this step gives the name to the algorithm). We note that $\boldsymbol{W}\dot{\boldsymbol{q}}_N = \boldsymbol{0}$.

Step **d** is the core of the algorithm, and is given by the following *SNS projection* equation

$$\dot{\boldsymbol{q}} = \dot{\boldsymbol{q}}_N + (\boldsymbol{JW})^{\#}\left(\dot{\boldsymbol{x}} - \boldsymbol{J}\dot{\boldsymbol{q}}_N\right) = (\boldsymbol{JW})^{\#}\dot{\boldsymbol{x}} + \tilde{\boldsymbol{P}}\dot{\boldsymbol{q}}_N, \quad (16)$$

where

$$\tilde{\boldsymbol{P}} = \boldsymbol{I} - (\boldsymbol{JW})^{\#}\boldsymbol{J} \quad (17)$$

is a (non-orthogonal) projection matrix (i.e., $\tilde{\boldsymbol{P}}\tilde{\boldsymbol{P}} = \tilde{\boldsymbol{P}}$) in the null space of the modified task Jacobian, as obtained when using only the enabled joints. The velocity command (16) is the current solution that tries to execute (at best) the desired task, by forcing the velocity of some set of overdriven joints to saturation. The algorithm checks the velocity constraints for the new solution, and the loop is repeated.

The complete procedure is presented in pseudocode form as **Algorithm 1**. The algorithm is initialized with $\boldsymbol{W} = \boldsymbol{I}$ (all joints enabled), a null space vector $\dot{\boldsymbol{q}}_N = \boldsymbol{0}$, the nominal scaling factor $s = 1$, and the current best scaling factor $s^* = 0$. With this initialization, eq. (16) collapses to the usual pseudoinverse (minimum norm) solution $\boldsymbol{J}^{\#}\dot{\boldsymbol{x}}$.

An important aspect in the proposed method is the integrated use of a task scaling procedure, presented in pseudocode form as **Algorithm 2**. As a matter of fact, the projection of $\dot{\boldsymbol{q}}_N$ in a suitable null space allows the use of a larger task scaling factor (possibly equal to 1) with respect to classical task scaling —see examples in [23, Sec. II]. Morover,

the robot capabilities will be used at best since saturated joint velocities are not affected by the task scaling, as opposed to the joint velocity scaling proposed in [11].

Algorithm 1 SNS algorithm

$\boldsymbol{W} = \boldsymbol{I}$, $\dot{\boldsymbol{q}}_N = \boldsymbol{0}$, $s = 1$, $s^* = 0$
repeat
  limit_exceeded = FALSE
  $\dot{\boldsymbol{q}} = \dot{\boldsymbol{q}}_N + (\boldsymbol{JW})^{\#} (\dot{\boldsymbol{x}} - \boldsymbol{J}\dot{\boldsymbol{q}}_N)$ } d
  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
  $\boldsymbol{a} = (\boldsymbol{JW})^{\#} \dot{\boldsymbol{x}}$
  $\boldsymbol{b} = \dot{\boldsymbol{q}} - \boldsymbol{a}$
  getTaskScalingFactor($\boldsymbol{a}$, $\boldsymbol{b}$) (∗**call Algorithm 2**∗) } a
  if {task scaling factor} $> s^*$ **then**
    $s^* = \{\text{task scaling factor}\}$
    $\boldsymbol{W}^* = \boldsymbol{W}$, $\dot{\boldsymbol{q}}^*_N = \dot{\boldsymbol{q}}_N$
  end if
  j = {the most critical joint}
  $W_{jj} = 0$
  $\dot{q}_{N,j} = \left\{ \begin{array}{ll} \dot{Q}_{max} & \text{if } \dot{q}_j > \dot{Q}_{max} \\ \dot{Q}_{min} & \text{if } \dot{q}_j < \dot{Q}_{min} \end{array} \right.$ } c
  if rank$(\boldsymbol{JW}) < m$ **then**
    $s = s^*$, $\boldsymbol{W} = \boldsymbol{W}^*$, $\dot{\boldsymbol{q}}_N = \dot{\boldsymbol{q}}^*_N$
    $\dot{\boldsymbol{q}} = \dot{\boldsymbol{q}}_N + (\boldsymbol{JW})^{\#} (s\dot{\boldsymbol{x}} - \boldsymbol{J}\dot{\boldsymbol{q}}_N)$
    limit_exceeded = FALSE *(∗outputs solution∗)* } b
  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

C. Properties and computational issues

A first property of the basic SNS algorithm is that the provided solution minimizes the norm of the velocity of the enabled joints (those not in saturation). From the definition of $\boldsymbol{W}$, it follows that the ith column of matrix $\boldsymbol{JW}$ is zero, so that the ith row of $(\boldsymbol{JW})^{\#}$ will be zero as well. Thus, (16) will not update the joint velocity $\dot{q}_i$ once the Algorithm 1 has saturated its value. At a generic iteration, let n_s be the number of velocity commands that are in saturation (disabled), while the remaining $n_e = n - n_s \geq m$ are enabled (and yet to be defined). After relabeling/reordering the joints, we can partition $\dot{\boldsymbol{q}}$, and accordingly the task Jacobian $\boldsymbol{J}$ and the matrix $\boldsymbol{W}$, as

$$\dot{\boldsymbol{q}} = \begin{pmatrix} \dot{\boldsymbol{q}}_e \\ \dot{\boldsymbol{q}}_s \end{pmatrix}, \quad \boldsymbol{J} = \begin{pmatrix} \boldsymbol{J}_e & \boldsymbol{J}_s \end{pmatrix}, \quad \boldsymbol{W} = \begin{pmatrix} \boldsymbol{I}_{n_e} & \\ & \boldsymbol{O}_{n_s} \end{pmatrix}$$

with blocks of suitable dimensions. Vector $\dot{\boldsymbol{q}}_s \in \mathbb{R}^{n_s}$ contains the saturated joint velocities. Thus, we have

$$\boldsymbol{JW} = \begin{pmatrix} \boldsymbol{J}_e & \boldsymbol{O} \end{pmatrix}, \qquad \dot{\boldsymbol{q}}_N = \begin{pmatrix} \boldsymbol{0} \\ \dot{\boldsymbol{q}}_s \end{pmatrix}$$

and equation (16) (with a generic value of the scaling factor s) can be rewritten as

$$\dot{\boldsymbol{q}}_{SNS} = \begin{pmatrix} \boldsymbol{0} \\ \dot{\boldsymbol{q}}_s \end{pmatrix} + \begin{pmatrix} \boldsymbol{J}_e^{\#} \\ \boldsymbol{O} \end{pmatrix} (s\dot{\boldsymbol{x}} - \boldsymbol{J}_s\dot{\boldsymbol{q}}_s). \qquad (18)$$

It is easy to see that $\dot{\boldsymbol{q}}_e$ obtained from (18) is the minimum norm solution of the following optimization problem

$$\min \frac{1}{2}\dot{\boldsymbol{q}}^T\boldsymbol{W}\dot{\boldsymbol{q}} \quad \text{s.t.} \quad \boldsymbol{J}\dot{\boldsymbol{q}} = s\dot{\boldsymbol{x}},$$

which is in fact equivalent to

$$\min \frac{1}{2}\dot{\boldsymbol{q}}_e^T\dot{\boldsymbol{q}}_e \quad \text{s.t.} \quad \boldsymbol{J}_e\dot{\boldsymbol{q}}_e = s\dot{\boldsymbol{x}} - \boldsymbol{J}_s\dot{\boldsymbol{q}}_s, \qquad (19)$$

for a given set of saturated joints $\dot{\boldsymbol{q}}_s$ and associated values. Indeed, we can also express (18) in a form containing explicitly the projection matrix (17) that results from the partition into enabled/disabled joints:

$$\dot{\boldsymbol{q}}_{SNS} = (\boldsymbol{JW})^{\#} s\dot{\boldsymbol{x}} + \tilde{\boldsymbol{P}}\dot{\boldsymbol{q}}_N = \begin{pmatrix} \boldsymbol{J}_e^{\#} \\ \boldsymbol{O} \end{pmatrix} s\dot{\boldsymbol{x}} + \begin{pmatrix} -\boldsymbol{J}_e^{\#}\boldsymbol{J}_s \\ \boldsymbol{I}_{n_s} \end{pmatrix} \dot{\boldsymbol{q}}_s. \qquad (20)$$

Another unique feature of the SNS algorithm is the inclusion of a task scaling procedure, which is automatically activated as soon as robot motion capabilities are found insufficient to achieve the original task. When a task is unfeasible, state-of-the art methods like those in [20], [21] look for relaxed solutions that minimize the norm of the task error vector. However, the resulting task will be uniformly deformed[4]. The outcome is that the instantaneous robot motion will be unpredictable and may lead to unfeasible or dangerous motions. An example of this behavior can be seen in the numerical results of Sec. VIII (see Fig. 7). On the other hand, task scaling guarantees at least the geometric directionality of the executed task: the desired task trajectory will be slowed down, but the path will be preserved. This characteristic is essential in many applications, e.g., welding or cutting tasks,

[4]A similar comment applies to the case of multiple tasks that is considered in Sec. **??**.

and even more when the robot has to coexist with human operators or move in the vicinity of obstacles.

One possible computational drawback of the presented method is the need to recompute $(\boldsymbol{JW})^{\#}$ (actually $\boldsymbol{J}_e^{\#}$, as just shown) each time a new joint is saturated and $\boldsymbol{W}$ is modified during the iterations. However, since only one element at a time is changed in the diagonal of $\boldsymbol{W}$, and thus only one column is modified in the next $\boldsymbol{JW}$, its pseudoinverse can be obtained as a rank-one update of the previous solution. Simple formulas can be found in [34] to compute such update, without the need of any SVD or QR operation. This characteristic will be one of the key ingredients used in Sec. VII for obtaining a faster version of the algorithm.

IV. Optimal SNS Method

We present a variant of the basic SNS algorithm, able to guarantee optimality in terms of velocity norm minimization over all joints.

A. Optimality of the SNS solution

Algorithm 1 finds a solution that minimizes the norm of the velocity sub-vector associated to the current set of enabled joints, see (19). In addition, **Algorithm 2** finds by construction an associated sub-optimal task scaling factor. However, nothing is said about the optimality of the current set of saturated joint commands. Thus, it may be possible that the same task scaling factor can be obtained with a different joint velocity vector having a smaller norm. To check if the solution provided by the basic SNS method is optimal in some useful sense, we introduce the following optimization problem:

$$\begin{aligned} \min_{\dot{\boldsymbol{q}}\in\mathbb{R}^n,\, s\in[0,1]} \quad & \frac{1}{2}\,\|\dot{\boldsymbol{q}}\|^2 + \frac{1}{2}\,M(1-s)^2 \\ \text{s.t.} \quad & \boldsymbol{J}\dot{\boldsymbol{q}} = s\dot{\boldsymbol{x}}, \quad \dot{\boldsymbol{Q}}_{min} \le \dot{\boldsymbol{q}} \le \dot{\boldsymbol{Q}}_{max}, \end{aligned} \tag{21}$$

where the parameter $M \gg 1$ works as a penalty factor and is used to weight more the maximization of the task scaling factor than the minimization of the joint velocity norm.

Problem (21) can be rewritten as a positive definite QP problem with linear equality and inequality constraints,

$$\begin{aligned} \min_{\boldsymbol{\xi}} \quad & \frac{1}{2}\,\boldsymbol{\xi}^T\boldsymbol{H}\boldsymbol{\xi} \\ \text{s.t.} \quad & \boldsymbol{A}\boldsymbol{\xi} = \boldsymbol{b}, \quad \boldsymbol{C}\boldsymbol{\xi} \le \boldsymbol{d}, \end{aligned} \tag{22}$$

by defining

$$\begin{aligned} \boldsymbol{\xi} &= \begin{pmatrix} \dot{\boldsymbol{q}} \\ 1-s \end{pmatrix}, \qquad \boldsymbol{H} = \begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0}^T & M \end{pmatrix} \\ &\qquad\qquad \triangleq \text{ blockdiag}\{\boldsymbol{I}, M\}, \\ \boldsymbol{A} &= \begin{pmatrix} \boldsymbol{J} & \dot{\boldsymbol{x}} \end{pmatrix}, \qquad \boldsymbol{b} = \dot{\boldsymbol{x}}, \\ \boldsymbol{C} &= \begin{pmatrix} -\boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{I} & \boldsymbol{0} \end{pmatrix}, \quad \boldsymbol{d} = \begin{pmatrix} -\dot{\boldsymbol{Q}}_{min} \\ \dot{\boldsymbol{Q}}_{max} \end{pmatrix}, \end{aligned} \tag{23}$$

where $\boldsymbol{0}$ represents the zero vector of dimension n.

Being the QP problem (22) convex, necessary and sufficient optimality conditions are given by the Karush-Kuhn-Tucker (KKT) criteria [35]. A task scale factor s and a joint velocity $\dot{\boldsymbol{q}}$ are optimal for problem (21) if and only if the following conditions are satisfied:

$$\boldsymbol{H}\boldsymbol{\xi} + \boldsymbol{A}^T\boldsymbol{\lambda} + \boldsymbol{C}^T\boldsymbol{\mu} = \boldsymbol{0} \tag{24a}$$

$$\boldsymbol{\mu}^T(\boldsymbol{C}\boldsymbol{\xi} - \boldsymbol{d}) = 0 \tag{24b}$$

$$\boldsymbol{A}\boldsymbol{\xi} = \boldsymbol{b} \tag{24c}$$

$$\boldsymbol{C}\boldsymbol{\xi} \le \boldsymbol{d} \tag{24d}$$

$$\boldsymbol{\mu} \ge \boldsymbol{0} \tag{24e}$$

where $\boldsymbol{\lambda} \in \mathbb{R}^m$ and $\boldsymbol{\mu} \in \mathbb{R}^{2n}$ are the Lagrange multipliers associated to the equality and, respectively, inequality constraints.

Condition (24c) requires preserving the (possibly scaled) task, i.e., $\boldsymbol{J}\dot{\boldsymbol{q}} = s\dot{\boldsymbol{x}}$. This condition, together with the satisfaction of the hard constraints (24d), is automatically imposed at every SNS step. Equation (24b) is the so-called complementarity condition. When the generic constraint j is saturated (active), it is $\boldsymbol{c}_j\boldsymbol{\xi} - d_j = 0$, where $\boldsymbol{c}_j$ is the jth row of $\boldsymbol{C}$ and d_j the jth element of $\boldsymbol{d}$, and this condition is satisfied irrespective of the value of the component μ_j of the multiplier $\boldsymbol{\mu} \ge \boldsymbol{0}$. In order to satisfy (24b) also for a non-saturated constraint (i.e., when $\boldsymbol{c}_j\boldsymbol{\xi} < d_j$), the associated component μ_j needs to be equal to zero. This allows extracting from (24a) an expression related only to the non-saturated joint velocities, where no component of $\boldsymbol{\mu}$ will appear. Multiplying on the left the first n equations in (24a) by the $n \times n$ diagonal selection matrix $\boldsymbol{W}^T = \boldsymbol{W}$ of the SNS method yields

$$\bar{\boldsymbol{H}}\boldsymbol{\xi} + \begin{pmatrix} \boldsymbol{JW} & \dot{\boldsymbol{x}} \end{pmatrix}^T\boldsymbol{\lambda} = \boldsymbol{0}, \tag{25}$$

where $\bar{\boldsymbol{H}} = \text{blockdiag}\{\boldsymbol{W}, M\}$. The vanishing of the term $(\boldsymbol{C} \cdot \text{blockdiag}\{\boldsymbol{W}, 1\})^T\boldsymbol{\mu} = \boldsymbol{0}$ is due to the fact that the components of $\boldsymbol{\mu}$ associated to non-saturated constraints should be zero at the optimum.

Since the possibility that $\boldsymbol{JW}$ is rank deficient is already checked inside the SNS algorithm, equation (25) has always a solution $\boldsymbol{\lambda}$ that can be obtained by pseudoinversion as

$$\boldsymbol{\lambda} = -\begin{pmatrix} (\boldsymbol{JW})^T \\ \dot{\boldsymbol{x}}^T \end{pmatrix}^{\#} \bar{\boldsymbol{H}}\boldsymbol{\xi}. \tag{26}$$

Accordingly, there exists a Lagrange multiplier $\boldsymbol{\mu}$, with zeros elements associated to non-active constraints, that satisfies the KKT condition (24b).

At this stage, $\boldsymbol{\mu}$ can by computed by imposing the satisfaction of the KKT condition (24a). Using $\boldsymbol{\lambda}$ from eq. (26) yields

$$\bar{\boldsymbol{\mu}} = -\left(\boldsymbol{H} - \boldsymbol{A}^T\begin{pmatrix} (\boldsymbol{JW})^T \\ \dot{\boldsymbol{x}}^T \end{pmatrix}^{\#} \bar{\boldsymbol{H}}\right)\boldsymbol{\xi}, \tag{27}$$

and taking into account the fulfillment of condition (24b), it follows that:

$$\begin{aligned} &\mu_i = \mu_{i+n} = 0, && \text{if } W_{i,i} = 1 \\ &\mu_i = -\bar{\mu}_i,\ \mu_{i+n} = 0, && \text{if } W_{i,i} = 0 \text{ AND } \dot{q}_i = \dot{Q}_{min,i} \\ &\mu_i = 0,\ \mu_{i+n} = \bar{\mu}_i, && \text{if } W_{i,i} = 0 \text{ AND } \dot{q}_i = \dot{Q}_{max,i} \\ &\text{for } i = 1, \ldots, n. \end{aligned} \tag{28}$$

Summarizing, conditions (24b), (24c), and (24d) are satisfied directly within the SNS algorithm, while condition (24a)

is imposed using eq. (27). Thus, only condition (24e) remains to be evaluated in order to check the optimality of the SNS solution. Note that eq. (27) does not require expensive computation since the pseudoinversion therein can be obtained by appending a row (rank-one update) to $(\boldsymbol{JW})^{\#}$, which is already computed inside the SNS algorithm.

The optimality check can be further simplified by observing that the SNS algorithm is already able to output a task scaling factor close to 1 (its maximum). Therefore, we can remove the optimality request (and check) for the task scaling factor and consider the simplified QP problem

$$\begin{aligned} \min_{\dot{\boldsymbol{q}}\in\mathbb{R}^n} \ & \frac{1}{2}\|\dot{\boldsymbol{q}}\|^2 \\ \text{s.t.} \quad & \boldsymbol{J}\dot{\boldsymbol{q}} = s\dot{\boldsymbol{x}}, \quad \dot{\boldsymbol{Q}}_{min} \le \dot{\boldsymbol{q}} \le \dot{\boldsymbol{Q}}_{max}, \end{aligned} \tag{29}$$

where s is the scale factor obtained inside the SNS algorithm.

Problem (29) is written in the form (22) by 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 (24) applied to the QP problem (29) lead to the same analysis of the previous case. Namely, conditions (24b), (24c), and (24d) are satisfied directly within the SNS algorithm, while condition (24a) can be imposed by choosing $\boldsymbol{\lambda}$ as in (26). As a result, one needs only to check the non-negativity of the components of the multiplier $\boldsymbol{\mu}$, which can be computed using again relation (28), now with

$$\bar{\boldsymbol{\mu}} = -\tilde{\boldsymbol{P}}^T\dot{\boldsymbol{q}} \tag{30}$$

in place of (27). The evaluation of eq. (30) is very fast, being $\tilde{\boldsymbol{P}}$ already computed inside each SNS step —see eq. (17).

We will see in Sec. V that the simplified optimization problem does not worsen the algorithm performance in a noticeable way.

B. The Opt-SNS algorithm

The information given by the KKT conditions allows detecting whether a joint command in saturation should or should not belong to the optimal saturation set. Namely, if $\mu_i < 0$ the ith joint will not be in the optimal saturation set. Therefore, if the ith joint velocity is currently in saturation, it will be removed from the saturation set by choosing $W_{ii} = 1$, $\dot{q}_{N,i} = 0$.

The Optimal SNS (Opt-SNS) algorithm is derived from the basic one, as presented in pseudocode form in **Algorithm 3**. A joint enters the active set (its velocity will be saturated) if it is the most critical one at the current step, i.e., it requires the largest task scaling. A joint can now also be removed from the active set if the associated multiplier μ_i is negative. Note that only $\bar{\boldsymbol{\mu}}$ needs to be computed for this test, because the conversion in $\boldsymbol{\mu}$ (see (28)) is already embedded in the innermost *if* statement. Algorithm 3 stops when the optimal solution has been reached[5].

Another difference between the basic SNS and the Opt-SNS algorithm is in the initialization of the selection matrix $\boldsymbol{W}$ that represents the current saturation set. While this is initialized with an empty set in Algorithm 1, the initialization in Opt-SNS uses $\boldsymbol{W}^-$, the optimal saturation set obtained at the previous execution of the algorithm. Indeed, Opt-SNS can both add and remove a joint from the saturation set, while the basic SNS was only able to add a saturated joint. Such feasible initialization typically reduces the number of internal steps of the algorithm: if the desired task is smooth, the optimal solutions associated to two consecutive samples of the task trajectory will be close in the joint velocity space. Therefore, they will have a similar saturation set. These considerations will be used again and expanded in Sec. VII-D to obtain a so-called warm start.

[5]For this reason, a more appropriate name for the variable *limit_exceeded* inherited from the basic SNS algorithm would be *non_optimal_solution*.

Algorithm 3 Opt-SNS algorithm

$\boldsymbol{W} = \boldsymbol{W}^-$, $\dot{\boldsymbol{q}}_N = \boldsymbol{0}$, $s = 1$, $s^* = 0$
repeat
  ... (same code of **Algorithm 1**)
  $\bar{\boldsymbol{\mu}} = -\tilde{\boldsymbol{P}}^T\dot{\boldsymbol{q}}$
  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
$\boldsymbol{W}^- = \boldsymbol{W}$

V. Joint Velocity Discontinuities

An intrinsic drawback of the considered optimization problem (22) is that it may lead to optimal joint velocities that are discontinuous over time, even if the desired task trajectory is smooth. Such possible discontinuities are due to the enable/disable switching mechanism of joints when considering the presence of task scaling. Thus, they are not a result of the chosen Opt-SNS algorithm, but affect equally all solution methods. The phenomenon was first noticed in [23], where we proposed the transposition of the algorithm to the acceleration level in order to recover continuous velocities, and then found also in [25], where state-of-the-art QP solvers were used for comparison.

Two main sources of potential discontinuities of joint velocity commands have been identified. The first is observed when a robot joint reaches its limit with a non-zero velocity, thus a sudden velocity jump occurs due to the desire of preserving a feasible motion. This effect is eliminated in our method through the shaping of the velocity constraints in eq. (12), which dynamically combines all hard limits (11) on position, velocity, and acceleration. The effect of this constraint shaping is sketched in Fig. 4, with the joint velocity constraint being smoothly activated both when $|\dot{q}_i| \to V_{max,i}$ or when $q_i \to Q_{min/max,i}$.

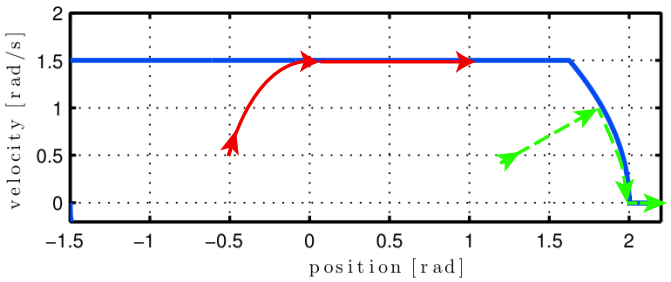

Fig. 4. Examples of joint saturation based on the hard constraints (12) for the joint velocity (in blue). There are no jumps on the realizable joint velocity, both when reaching the maximum joint velocity (solid, red) or when reaching the joint limit (dashed, green).

The second source of discontinuity is due to the need of task scaling. In fact, in all our case studies we observed 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 formulated QP problem (21). In fact, any QP solver would guarantee that the optimal value of the objective is continuous for continuous tasks. However, since a large $M \gg 1$ is required, this continuity is reflected almost completely on the task scaling factor alone, while discontinuities of numerical nature cannot be excluded for $\dot{\boldsymbol{q}}$.

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. This minimal increase in motion time pays off, since joint velocity discontinuities are completely removed —see Fig. 6 in Sec. VIII.

The actual chosen value of the scaling margin s_m is related to the kinematics of the manipulator and to the desired task. For its properties, the larger the scale margin is, the smaller is the probability to encounter discontinuities; thus, it is always possible to find a suitable (small) value for a specific application. In our experience, values of the scale margin between 0.05 and 0.15 are sufficient to guarantee absence of discontinuities. In any event, as anticipated in Sec. IV-A, such a modification in the task scaling ensuring continuity of velocity commands supports also the use of the simplified optimal problem (29) in place of the original (21).

VI. Extension to Multiple Tasks

labelsec:multiple

Consider now l tasks $\dot{\boldsymbol{x}}_k = \boldsymbol{J}_k\dot{\boldsymbol{q}}$, each of dimension m_k ($k = 1,\dots,l$) and organized with priority, as introduced in Sec. II. The SNS method can be extended to this case, by imposing a correct *preemptive* prioritization strategy in spite of the presence of hard constraints. As a matter of fact, higher priority tasks will use in the best way all robot capabilities they need, while lower priority tasks are accommodated with the residual capability so as not to interfere with the execution of tasks of higher priority. **Algorithm 4** presents in pseudocode form the solution method for the multi-task case.

Algorithm 4 SNS algorithm for multiple tasks

$\boldsymbol{P}_{A,0} = \boldsymbol{I}$, $\dot{\boldsymbol{q}}_0 = \boldsymbol{0}$
for $k = 1 \to l$ **do**
  $\boldsymbol{W}_k = \boldsymbol{I}$, $\dot{\boldsymbol{q}}_{N,k} = \boldsymbol{0}$, $s_k = 1$, $s_k^* = 0$, $\bar{\boldsymbol{P}}_k = \boldsymbol{P}_{A,k-1}$
  repeat
    limit_exceeded = FALSE
    $\tilde{\boldsymbol{P}}_k = \left(\boldsymbol{I} - \left(\boldsymbol{J}_k\bar{\boldsymbol{P}}_k\right)^{\#}\boldsymbol{J}_k\right)\left(\left(\boldsymbol{I}-\boldsymbol{W}_k\right)\boldsymbol{P}_{A,k-1}\right)^{\#}$
    $\dot{\boldsymbol{q}}_k = \dot{\boldsymbol{q}}_{k-1} + \left(\boldsymbol{J}_k\bar{\boldsymbol{P}}_k\right)^{\#}\left(s_k\dot{\boldsymbol{x}}_k - \boldsymbol{J}_k\dot{\boldsymbol{q}}_{k-1}\right) + \tilde{\boldsymbol{P}}_k\dot{\boldsymbol{q}}_{N,k}$
    if $\left\{\begin{array}{l}\exists\ i \in [1\!:\!n]\ : \\ \dot{q}_{k,i} < \dot{Q}_{min,i}\ \text{.OR. }\dot{q}_{k,i} > \dot{Q}_{max,i}\end{array}\right\}$ **then**
      limit_exceeded = TRUE
      $\boldsymbol{a} = \left(\boldsymbol{J}_k\bar{\boldsymbol{P}}_k\right)^{\#}\ddot{\boldsymbol{x}}_k$
      $\boldsymbol{b} = \dot{\boldsymbol{q}}_k - \boldsymbol{a}$
      getTaskScalingFactor($\boldsymbol{a}$, $\boldsymbol{b}$) (∗**call Algorithm 2**∗)
      if {task scaling factor} $> s_k^*$ **then**
        $s_k^* = \{\text{task scaling factor}\}$
        $\boldsymbol{W}_k^* = \boldsymbol{W}_k$, $\dot{\boldsymbol{q}}_{N,k}^* = \dot{\boldsymbol{q}}_{N,k}$, $\bar{\boldsymbol{P}}_k^* = \bar{\boldsymbol{P}}_k$
      end if
      j = {the most critical joint}
      $W_{k,jj} = 0$
      $\dot{q}_{N,k,j} = \left\{\begin{array}{ll}\dot{Q}_{max,j} - \dot{q}_{k-1,j} & \text{if } \dot{q}_{k,j} > \dot{Q}_{max,j} \\ \dot{Q}_{min,j} - \dot{q}_{k-1,j} & \text{if } \dot{q}_{k,j} < \dot{Q}_{min,j}\end{array}\right.$
      $\bar{\boldsymbol{P}}_k = \left(\boldsymbol{I} - \left(\left(\boldsymbol{I}-\boldsymbol{W}_k\right)\boldsymbol{P}_{A,k-1}\right)^{\#}\right)\boldsymbol{P}_{A,k-1}$
      if rank$\left(\boldsymbol{J}_k\bar{\boldsymbol{P}}_k\right) < m_k$ **then**
        $s_k = s_k^*$, $\boldsymbol{W}_k = \boldsymbol{W}_k^*$, $\dot{\boldsymbol{q}}_{N,k} = \dot{\boldsymbol{q}}_{N,k}^*$, $\bar{\boldsymbol{P}}_k = \bar{\boldsymbol{P}}_k^*$
        $\dot{\bar{\boldsymbol{q}}}_{N,k} = \left(\left(\boldsymbol{I}-\boldsymbol{W}_k\right)\boldsymbol{P}_{A,k-1}\right)^{\#}\dot{\boldsymbol{q}}_{N,k}$
        $\bar{\boldsymbol{P}}_k = \left(\boldsymbol{I} - \left(\left(\boldsymbol{I}-\boldsymbol{W}_k\right)\boldsymbol{P}_{A,k-1}\right)^{\#}\right)\boldsymbol{P}_{A,k-1}$
        $\tilde{\dot{\boldsymbol{q}}}_k = \dot{\boldsymbol{q}}_{k-1} + \dot{\bar{\boldsymbol{q}}}_{N,k}$
        $\dot{\boldsymbol{q}}_k = \tilde{\dot{\boldsymbol{q}}}_k + \left(\boldsymbol{J}_k\bar{\boldsymbol{P}}_k\right)^{\#}\left(\dot{\boldsymbol{x}}_k - \boldsymbol{J}_k\tilde{\dot{\boldsymbol{q}}}_k\right)$
        limit_exceeded = FALSE (∗*outputs solution*∗)
      end if
    end if
  until limit_exceeded = TRUE
  $\boldsymbol{P}_{A,k} = \boldsymbol{P}_{A,k-1} - \left(\boldsymbol{J}_k\boldsymbol{P}_{A,k-1}\right)^{\#}\left(\boldsymbol{J}_k\boldsymbol{P}_{A,k-1}\right)$
end for
$\dot{\boldsymbol{q}}_{SNS} = \dot{\boldsymbol{q}}_l$

In the algorithm, we denote by $\dot{\boldsymbol{q}}_k$ the joint velocity that satisfies at best the first $k-1$ tasks (which is thus *feasible* w.r.t. the hard constraints) and includes the current guess of the joint velocity that addresses task k. For each task (looping over all tasks, $k = 1,\dots,l$), the initializations of matrix $\boldsymbol{W}_k$, of the null-space vector $\dot{\boldsymbol{q}}_{N,k}$, and of the two scaling factors s_k and s_k^* are the same as in Algorithm 1. In addition, we define the auxiliary projection matrix $\bar{\boldsymbol{P}}_k$, which has the same

functionality of $\boldsymbol{W}$ in the single task case, but takes into account also the null space of previous tasks.

The joint velocity $\dot{\boldsymbol{q}}_1$ that satisfies the first (highest priority) task is the same obtained with Algorithms 1–2, being $\bar{\boldsymbol{P}}_1 = \boldsymbol{P}_{A,0} = \boldsymbol{I}$. When attacking the generic task k, the joint velocity is computed with the SNS projection equation

$$\dot{\boldsymbol{q}}_k = \dot{\boldsymbol{q}}_{k-1} + \left(\boldsymbol{J}_k \bar{\boldsymbol{P}}_k\right)^{\#} \left(s_k \dot{\boldsymbol{x}}_k - \boldsymbol{J}_k \dot{\boldsymbol{q}}_{k-1}\right) + \tilde{\boldsymbol{P}}_k \dot{\boldsymbol{q}}_{N,k}, \quad (31)$$

where

$$\tilde{\boldsymbol{P}}_k = \left(\boldsymbol{I} - \left(\boldsymbol{J}_k \bar{\boldsymbol{P}}_k\right)^{\#} \boldsymbol{J}_k\right) \left(\left(\boldsymbol{I} - \boldsymbol{W}_k\right) \boldsymbol{P}_{A,k-1}\right)^{\#} \quad (32)$$

is a special projector that accommodates a joint velocity saturation, without deforming the task nor the hierarchy of priorities.

Similarly to the single task case, we check first if the task can be executed within the joint velocity constraints (12). If not, the task scaling factor and the most critical joint are computed using again Algorithm 2. With the scaling factor being the largest computed so far, the current solution dataset (s_k, $\boldsymbol{W}_k$, $\dot{\boldsymbol{q}}_{N,k}$, $\bar{\boldsymbol{P}}_k$) is saved. Next, the most critical joint j is disabled in the execution of the current task ($W_{k,jj} = 0$), and the velocity contribution of this saturated joint is assigned as null-space vector component $\dot{q}_{N,k,j}$. The auxiliary null-space projector is

$$\bar{\boldsymbol{P}}_k = \left(\boldsymbol{I} - \left(\left(\boldsymbol{I} - \boldsymbol{W}_k\right) \boldsymbol{P}_{A,k-1}\right)^{\#}\right) \boldsymbol{P}_{A,k-1}, \quad (33)$$

which is obtained by considering the addition of $\dot{\boldsymbol{q}}_{N,k}$ as an auxiliary task (at the configuration space level), thus with an associated Jacobian $(\boldsymbol{I} - \boldsymbol{W}_k)$.

If the rank of $\boldsymbol{J}_k \bar{\boldsymbol{P}}_k$ is strictly less than m_k, the kth loop of the algorithm terminates with the best parameters saved so far, and the velocity command is provided as output by (31). Otherwise, the joint velocity is recomputed with the current parameters and the process is repeated. Note that when (31) is used with saturated commands, the auxiliary null-space vector $\dot{\bar{\boldsymbol{q}}}_{N,k}$ forces the disabled joints to their saturated values without modifying the previous $k-1$ tasks. Once task k is satisfied (with scaling, if needed), the algorithm moves to the next task $k+1$. The final output $\dot{\boldsymbol{q}}_{SNS} = \dot{\boldsymbol{q}}_l$ of the algorithm is obtained after processing the (last) task l.

Remark 1: Equation (31) collapses into the kth step of the prioritized multi-task motion control scheme (3), as long as there are no saturations ($\boldsymbol{W}_k = \boldsymbol{I}$) and no scaling ($s_k = 1$) involved in the execution of the additional task k. In particular, if all tasks can be realized without command saturation, then Algorithm 4 is equivalent to eqs. (3–5).

Remark 2: Matrix $\boldsymbol{W}_k$ is rebuilt independently of the obtained matrices $\boldsymbol{W}_i$ at steps $i < k$. As a result, even if the velocity of a joint has been saturated for the execution of a higher priority task, the joint is still enabled in principle and could be reused by a lower priority task, which might then push this joint velocity away from its saturation level.

Remark 3: Some tasks may request a desired robot behavior directly in the configuration space (CS). This is of interest when addressing as a task the optimization of auxiliary criteria (e.g., robot manipulability by the Projected Gradient method), or when the redundant robot is commanded at the acceleration or torque level, in order to damp otherwise uncontrolled self-motion velocities. Within the framework of multiple tasks with priority, the SNS method provides a computationally simple and effective solution executing only that part of a CS task which preserves higher priority tasks, without violating the constraints on the joint velocity commands. When a generic task k in the hierarchy is a CS task, a simplified scheme can replace the kth loop in Algorithm 4 [24]. In particular, Algorithm 2 needs to be called just once, scaling the whole CS task by a common factor in order to fulfill the constraints (12).

A. Optimal version

To extend similarly the Opt-SNS algorithm to the case of multiple tasks with priority, the k-th priority task is reformulated as the following QP problem:

$$\begin{aligned} \min_{\dot{\boldsymbol{q}} \in \mathbb{R}^n} \; & \frac{1}{2} \left\| \dot{\boldsymbol{q}} \right\|^2 \\ \text{s.t.} \quad & \boldsymbol{J}_k \dot{\boldsymbol{q}} = s_k \dot{\boldsymbol{x}}_k, \quad \boldsymbol{J}_{A,k-1} \dot{\boldsymbol{q}} = \boldsymbol{J}_{A,k-1} \dot{\boldsymbol{q}}_{k-1}, \\ & \dot{\boldsymbol{Q}}_{min} \leq \dot{\boldsymbol{q}} \leq \dot{\boldsymbol{Q}}_{max}, \end{aligned} \quad (34)$$

where the task scaling factor s_k is the largest found with the SNS algorithm. As in Sec. IV-A, the evaluation of the KKT conditions is reduced to the evaluation of the Lagrangian multiplier $\boldsymbol{\mu}_k$, being all other conditions are guaranteed by the SNS algorithm. Following the same approach as in the single task case, it is not surprising that we obtain a similar result with

$$\bar{\boldsymbol{\mu}}_k = -\tilde{\boldsymbol{P}}_k^T \dot{\boldsymbol{q}}_k, \qquad \text{for any } k \in \{1, \ldots, l\}. \quad (35)$$

The cost of evaluating (35) for checking optimality and for updating the saturation set is negligible, since $\tilde{\boldsymbol{P}}_k$ is already computed within the SNS algorithm for multiple tasks.

B. Dealing with singularities

In the multi-task case, the SNS method has to deal with three different types of singularities, namely: *i)* when a task Jacobian looses rank by itself (*kinematic singularity*); *ii)* when a lower priority task is linearly dependent from a higher priority task, resulting in a rank deficient $\boldsymbol{J}_k \boldsymbol{P}_{A,k-1}$ matrix (*algorithmic singularity*); *iii)* when too many joints are saturated and the task is unfeasible, i.e., rank$(\boldsymbol{J}_k \bar{\boldsymbol{P}}_k) < m_k$.

The Opt-SNS algorithm discards any solution that is associated with the third kind of singularity, since such case will anyway not be optimal. When the kth task Jacobian $\boldsymbol{J}_k$ or the projected task Jacobian $\boldsymbol{J}_k \boldsymbol{P}_{A,k-1}$ is rank deficient, no joint can be saturated further to find a different solution. In these cases, damped pseudoinversion is used within the algorithm to find the next candidate solution and the task is then scaled to bring the joint velocity within the bounds.

When used with Opt-SNS, this strategy provides a better execution of the task and reduces the problems related to algorithmic singularities. In fact, the usual way to deal with singularities is to use relatively large damping to prevent high joint velocities, resulting in a task deformation for type *i)* singularities, and on a deformation of the whole chain of tasks up to the current one for type *ii)* singularities. Instead, the

solution with task scaling used in the SNS method leads in general to the need of a smaller damping. Therefore, tasks are deformed less and the influence of linearly dependent lower priority tasks on the execution of higher priority tasks is negligible.

VII. Fast SNS Methods

In this section, we address the issue of real-time performance improvements of the SNS algorithms introduced so far, in their basic or optimal version. For generality, we consider directly the case of multiple tasks with priority.

A. Reformulation of the SNS method

It can be shown that the same SNS projection equation (31) can be obtained using a suitable formulation based on task augmentation [27], which will allow in turn a faster QR decomposition. To this end, let[6]

$$\begin{aligned}\dot{\boldsymbol{q}}_k &= \dot{\boldsymbol{q}}_{k-1} + \boldsymbol{J}_T^{\#}\dot{\boldsymbol{q}}_T \\ &= \dot{\boldsymbol{q}}_{k-1} + \begin{pmatrix} \boldsymbol{J}_k\boldsymbol{P}_{A,k} \\ \boldsymbol{J}_W \end{pmatrix}^{\#} \begin{pmatrix} s_k\dot{\boldsymbol{x}}_k - \boldsymbol{J}_k\dot{\boldsymbol{q}}_{k-1} \\ \dot{\boldsymbol{q}}_W \end{pmatrix},\end{aligned} \tag{36}$$

where $\boldsymbol{J}_T$ is the augmentation of $\boldsymbol{J}_k\boldsymbol{P}_{A,k}$ with a $n_s \times n$ matrix $\boldsymbol{J}_W$, being n_s the number of saturated commands. If the velocity of 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 ith element. The velocity vector $\dot{\boldsymbol{q}}_W = \boldsymbol{J}_W\dot{\boldsymbol{q}}_{N,k}$, related to the SNS vector $\dot{\boldsymbol{q}}_{N,k}$, is then composed by the saturation values associated to $\boldsymbol{J}_W$. The joint velocity obtained with eq. (36) coincides with that of eq. (31) if there exists at least one feasible solution. Otherwise, equation (36) provides the smallest violation (in a least squares sense) of the constraints. However, this will produce also a relaxation of the hard inequality constraints, which is strictly forbidden in our framework. In fact, this situation is related to the exit condition of the SNS and Opt-SNS algorithms, namely rank $\left(\boldsymbol{J}_k\bar{\boldsymbol{P}}_k\right) < m_k$. Some further manipulation is thus needed.

By simple inspection, the pseudoinverse of $\boldsymbol{J}_T$ can be written in partitioned form as

$$\boldsymbol{J}_T^{\#} = \begin{pmatrix} \left(\boldsymbol{J}_k\bar{\boldsymbol{P}}_k\right)^{\#} & \boldsymbol{B} \end{pmatrix}, \tag{37}$$

where $\boldsymbol{B}$ is a $n \times n_s$ 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 added to (or removed from, in the Opt-SNS version) the saturation set, equation (36) can be obtained from the previous step as a rank one update [34] of the pseudoinverse of the augmented Jacobian. Moreover, due to the simple structure of the row that has been appended to $\boldsymbol{J}_T$, namely $\boldsymbol{J}_{W,i}$ if the ith joint is saturated, the update reduces to very simple formulas.

The operations associated with a new joint velocity component (the (n_s+1)-th) reaching saturation are detailed next. Removal from saturation is done following similar steps. Introduce the $n\times n$ orthogonal projector $\hat{\boldsymbol{P}}_{A,k}$, whose columns are denoted as $\hat{\boldsymbol{p}}_{k,i}$ $(i = 1,\ldots,n)$, and which is initialized at $\boldsymbol{P}_{A,k}$. The *update vector* $\boldsymbol{b}_i$ is obtained as

$$\boldsymbol{b}_i = \frac{\hat{\boldsymbol{p}}_{k,i}}{\hat{p}_{k,ii}}, \tag{38}$$

where $\hat{p}_{k,ii}$ is the ith element of vector $\hat{\boldsymbol{p}}_{k,i}$. Matrix $\boldsymbol{B}$ is updated and augmented by one column as

$$\boldsymbol{B} = \begin{pmatrix} (\boldsymbol{I} - \boldsymbol{b}_i\boldsymbol{J}_{W,i})\,\boldsymbol{B} & \boldsymbol{b}_i \end{pmatrix}, \tag{39}$$

and the SNS solution becomes

$$\begin{aligned}\dot{\boldsymbol{q}}_k = \dot{\boldsymbol{q}}_{k-1} &+ \boldsymbol{B}\boldsymbol{J}_W\dot{\boldsymbol{q}}_{N,k} \\ &+ (\boldsymbol{I} - \boldsymbol{B}\boldsymbol{J}_W)\,(\boldsymbol{J}_k\boldsymbol{P}_{A,k-1})^{\#}\left(s_k\dot{\boldsymbol{x}}_k - \boldsymbol{J}_k\dot{\boldsymbol{q}}_{k-1}\right).\end{aligned} \tag{40}$$

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

$$\hat{\boldsymbol{P}}_k = \hat{\boldsymbol{P}}_k + \boldsymbol{B}\boldsymbol{J}_W\left(\boldsymbol{I} - \hat{\boldsymbol{P}}_k\right). \tag{41}$$

Note that the columns of $\hat{\boldsymbol{P}}_k$ associated to saturated joints coincide with the columns of $\tilde{\boldsymbol{P}}_k$ in (31) and with the corresponding columns of $\boldsymbol{B}$ in (37) (or, equivalently, in (39)).

B. The Fast-SNS algorithm

Based on the formulas of Sec. VII-A, the solution update in the SNS algorithm can be achieved with a minimum amount of operations. If joint i saturates, the update vector $\boldsymbol{b}_i$ is obtained directly from the QR decomposition in (8) and (10). As recalled in Sec. II, it is $\hat{\boldsymbol{P}}_{A,k} = \boldsymbol{Z}_{A,k}\boldsymbol{Z}_{A,k}^T$. Thus, in eq. (38) we have $\hat{\boldsymbol{p}}_{k,i} = \boldsymbol{Z}_{A,k}\boldsymbol{z}_i^T$ and $\hat{p}_{k,ii} = \boldsymbol{z}_i\boldsymbol{z}_i^T$, where $\boldsymbol{z}_i$ represents the ith row of $\boldsymbol{Z}_{A,k}$. Then, it easily follows that

$$\boldsymbol{b}_i = \boldsymbol{Z}_{A,k}\,\frac{\boldsymbol{z}_i^T}{\boldsymbol{z}_i\boldsymbol{z}_i^T} = \boldsymbol{Z}_{A,k}\,\boldsymbol{z}_i^{\#}. \tag{42}$$

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

$$\dot{\boldsymbol{q}}_k = \dot{\boldsymbol{q}}_{k-1} + s_k\dot{\bar{\boldsymbol{q}}}' + \dot{\bar{\boldsymbol{q}}}'' + \dot{\bar{\boldsymbol{q}}}_W \tag{43}$$

initialized as $\dot{\bar{\boldsymbol{q}}}' = \dot{\boldsymbol{q}}'$, $\dot{\bar{\boldsymbol{q}}}'' = \dot{\boldsymbol{q}}''$, and $\dot{\bar{\boldsymbol{q}}}_W = \dot{\boldsymbol{q}}_W$, with

$$\begin{aligned}\dot{\boldsymbol{q}}' &= \boldsymbol{Z}_{A,k-1}\boldsymbol{Y}_k\boldsymbol{R}_k^{-T}\dot{\boldsymbol{x}}_k \\ \dot{\boldsymbol{q}}'' &= -\boldsymbol{Z}_{A,k-1}\boldsymbol{Y}_k\boldsymbol{R}_k^{-T}\boldsymbol{J}_k\dot{\boldsymbol{q}}_{k-1} \\ \dot{\boldsymbol{q}}_W &= \boldsymbol{0}\end{aligned} \tag{44}$$

coming from the solution (9), which is obtained without considering the hard inequalities (12). Therefore, in the following we call (44) the *unconstrained* solution. At each new saturation, these terms will be updated as

$$\begin{aligned}\dot{\bar{\boldsymbol{q}}}' &= \dot{\bar{\boldsymbol{q}}}' - \boldsymbol{b}_i\dot{\bar{\boldsymbol{q}}}'_i \\ \dot{\bar{\boldsymbol{q}}}'' &= \dot{\bar{\boldsymbol{q}}}'' - \boldsymbol{b}_i\dot{\bar{\boldsymbol{q}}}''_i \\ \dot{\bar{\boldsymbol{q}}}_W &= \dot{\bar{\boldsymbol{q}}}_W + \boldsymbol{b}_i\left(\dot{Q}_i - \dot{\boldsymbol{q}}_{k-1,i} - \dot{\bar{\boldsymbol{q}}}_{W,i}\right),\end{aligned} \tag{45}$$

where $\dot{Q}_i$ is the saturation value. The scaling factor s_k is given by Algorithm 2, 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 (43)). The null space projector is finally updated as

$$\boldsymbol{Z}_{A,k} = \boldsymbol{Z}_{A,k} - \boldsymbol{b}_i\boldsymbol{z}_i. \tag{46}$$

[6] In this section, in order to reduce notational burden, some intermediate quantities used only within the kth task loop will carry no k index.

C. The FastOpt-SNS algorithm

The fast algorithm described in Sec. VII-B yields in general only a suboptimal solution in terms of the QP problem (34). However, following the results of Sec. IV, we can improve also the Opt-SNS algorithm in a similar way by *i)* computing and updating more efficiently the multipliers (35), and *ii)* updating accordingly the solution when a joint command is removed from its saturation state. These are the core steps of the *FastOpt-SNS* algorithm presented next.

The multiplier associated to a new saturated joint velocity component of index i (the (n_s+1)-th going in saturation) is given by

$$\bar{\mu}_{k,i} = -\boldsymbol{b}_i^T \dot{\boldsymbol{q}}_k. \tag{47}$$

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

$$\bar{\mu}_{k,j} = \bar{\mu}_{k,j} - b_{j,i}\,\bar{\mu}_{k,i}, \qquad j = 1,\ldots,n_s, \tag{48}$$

where $b_{j,i}$ is the ith element of the update vector $\boldsymbol{b}_j$ for the jth saturated joint. Equation (48) shows that these vectors need to be stored and updated at each new saturation. This can be done with

$$\boldsymbol{b}_j = \boldsymbol{b}_j - \boldsymbol{b}_i b_{j,i}, \qquad j = 1,\ldots,n_s. \tag{49}$$

Since the n_s vectors $\boldsymbol{b}_j$ compose the matrix $\boldsymbol{B}$, eq. (49) executes the same update as eq. (39).

Using the multipliers, it is possible to identify a saturated joint command, say of index o, that should no longer saturate in the optimal solution. In this case, joint o has to be removed from the list of n_s saturated commands and the solution downgraded. The first step is to downgrade the update vectors associated to the n_s-1 joints that will remain in saturation:

$$\boldsymbol{b}_j = \boldsymbol{b}_j - \boldsymbol{b}_o \left(\frac{\boldsymbol{b}_o^T \boldsymbol{b}_j}{\boldsymbol{b}_o^T \boldsymbol{b}_o} \right), \qquad j = 1,\ldots,n_s. \tag{50}$$

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

$$\begin{aligned}
\dot{\tilde{\boldsymbol{q}}}' &= \dot{\tilde{\boldsymbol{q}}}' + \boldsymbol{b}_o \left(\dot{q}'_o - \sum_{j=1}^{n_s} b_{j,o}\,\dot{q}'_j \right) \\
\dot{\tilde{\boldsymbol{q}}}'' &= \dot{\tilde{\boldsymbol{q}}}'' + \boldsymbol{b}_o \left(\dot{q}''_o - \sum_{j=1}^{n_s} b_{j,o}\,\dot{q}''_j \right) \\
\dot{\tilde{\boldsymbol{q}}}_W &= \dot{\tilde{\boldsymbol{q}}}_W - \boldsymbol{b}_o \left(\dot{Q}_o - \sum_{j=1}^{n_s} b_{j,o} \left(\dot{Q}_j - \dot{q}_{k-1,j} \right) \right),
\end{aligned} \tag{51}$$

where $\dot{\boldsymbol{q}}'$ and $\dot{\boldsymbol{q}}''$ compose the unconstrained solution in eqs. (44) and the $\dot{Q}_j$'s are the saturation values. The null space projector has to restore the additional direction given by the joint of index o leaving the saturation set. Thus, $\boldsymbol{Z}_{A,k}$ is downgraded as

$$\boldsymbol{Z}_{A,k} = \boldsymbol{Z}_{A,k} - \boldsymbol{b}_0 \left(\tilde{\boldsymbol{z}}_o - \sum_{j=1}^{n_s} b_{j,o}\,\tilde{\boldsymbol{z}}_j \right), \tag{52}$$

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

$$\bar{\mu}_{k,j} = \bar{\mu}_{k,j} + b_{j,o}\,\bar{\mu}_{k,o}, \quad j = 1,\ldots,n_s, \quad j \neq o. \tag{53}$$

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

D. 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, the new 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 kth task at the previous time sample, e.g., matrix $\boldsymbol{J}_W$ is known. Following similar ideas as in [21], the new solution can be computed efficiently starting from the same previous set of saturated joints and proceeding as follows.

We first obtain the update vectors associated to all saturated joints, organized as a matrix

$$\boldsymbol{B} = \boldsymbol{Z}_{A,k} \left(\boldsymbol{J}_W \boldsymbol{Z}_{A,k} \right)^{\#}. \tag{54}$$

These update vectors allow to obtain the SNS solution from the unconstrained solution in (44)

$$\begin{aligned}
\dot{\tilde{\boldsymbol{q}}}' &= \dot{\boldsymbol{q}}' - \boldsymbol{B}\boldsymbol{J}_W \dot{\boldsymbol{q}}' \\
\dot{\tilde{\boldsymbol{q}}}'' &= \dot{\boldsymbol{q}}'' - \boldsymbol{B}\boldsymbol{J}_W \dot{\boldsymbol{q}}'' \\
\dot{\tilde{\boldsymbol{q}}}_W &= \dot{\boldsymbol{q}}_W + \boldsymbol{B}\boldsymbol{J}_W \dot{\boldsymbol{q}}_{N,k}.
\end{aligned} \tag{55}$$

The directions associated to the saturated joints are removed from the null space projector by

$$\boldsymbol{Z}_{A,k} = \boldsymbol{Z}_{A,k} - \boldsymbol{B}\boldsymbol{J}_W \boldsymbol{Z}_{A,k}, \tag{56}$$

and the multipliers associated to the saturated joints are obtained as

$$\bar{\boldsymbol{\mu}}_k = -\boldsymbol{B}^T \dot{\boldsymbol{q}}_k. \tag{57}$$

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

VIII. Numerical Results

The SNS method and each of the various proposed algorithms have been extensively tested in simulations using a kinematic model of the KUKA LWR IV robot ($n = 7$). From the data sheet, the joint range limits and the bounds on joint velocities are all symmetric and equal to

$$\boldsymbol{Q}_{max} = -\boldsymbol{Q}_{min} = (170, 120, 170, 120, 170, 120, 170) \text{ [deg]}$$
$$\boldsymbol{V}_{max} = -\boldsymbol{V}_{min} = (100, 110, 100, 130, 130, 180, 180) \text{ [deg/s]}.$$

Acceleration bounds $\boldsymbol{A}_{max} = -\boldsymbol{A}_{min} = 300 \cdot \boldsymbol{I}$ [deg/s^2] have been chosen, equal and symmetric for all joints. A sampling time $T = 1$ [ms] is used, also for shaping the joint velocity constraints (12).

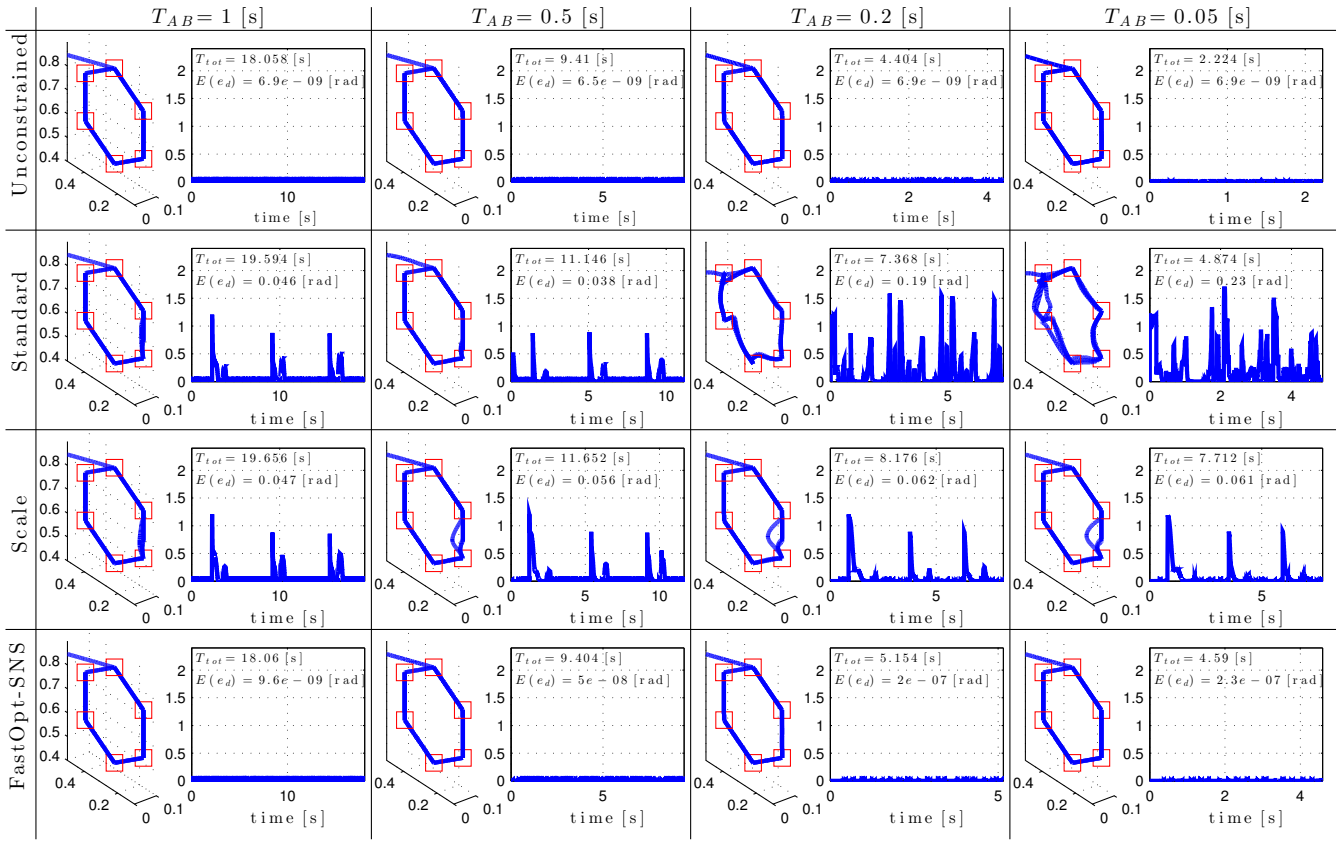

Fig. 5. Execution of a multi-point cyclic task trajectory with a KUKA LWR 4 for different timings $T_{AB} = 1$, 0.5, 0.2, and 0.05 s on each elementary path segment. [First row] Ideal case with pseudoinverse solution $\dot{\boldsymbol{q}} = \boldsymbol{J}^{\#}(\boldsymbol{q})\dot{\boldsymbol{x}}$ and without any constraints. [Second row] Standard pseudoinverse solution when including the presence of constraints. [Third row] Pseudoinverse solution followed by classical task scaling. [Fourth row] Solution with the FastOpt-SNS algorithm.

In the numerical results reported here, a single task of dimension $m_1 = 3$ has been specified (the degree of redundancy for this task is then $n - m_1 = 4$). The robot end-effector position $\boldsymbol{x}_1 = \boldsymbol{p}_{EE} = \boldsymbol{f}_1(\boldsymbol{q})$ should cycle three times through a series of six Cartesian points $\boldsymbol{X}_i$, $i = 1, \ldots, 6$, connected by linear segments, with the KUKA LWR starting from $\boldsymbol{q}(0) = (0, 45, 45, 45, 0, 0, 0)$ [deg] (i.e., off path, with an initial error). The Cartesian points are vertices of an hexagon inscribed in a circle lying in the (Y, Z) vertical plane, with center $\boldsymbol{C} = \begin{pmatrix} 0.1 & 0.35 & 0.6235 \end{pmatrix}^T$ [m] and radius $r = 0.2$ [m]. Between two generic successive Cartesian points $\boldsymbol{X}_A \rightarrow \boldsymbol{X}_B$ in the sequence, a rest-to-rest trajectory starts at time t_A, lasts T_{AB} seconds, and is given by

$$\begin{aligned} \boldsymbol{X}(t) &= \boldsymbol{X}_A + (\boldsymbol{X}_B - \boldsymbol{X}_A)\,\gamma(\tau) \\ \tau &= \frac{t - t_A}{T_{AB}} \in [0, 1] \\ \gamma(\tau) &= 6\tau^5 - 15\tau^4 + 10\tau^3 \\ \boldsymbol{v}(t) &= \dot{\boldsymbol{X}}(t) = \frac{\boldsymbol{X}_B - \boldsymbol{X}_A}{T_{AB}} \left(30\tau^4 - 60\tau^3 + 30\tau^2\right). \end{aligned} \tag{58}$$

The nominal time for one cycle is thus $6\,T_{AB}$, while for the complete motion $T_{tot} = 18 T_{AB}$. Because we need feedback control to recover the initial error or any following transient errors for staying as close as possible to the desired trajectory, the desired task velocity is specified as

$$\dot{\boldsymbol{x}}_1 = \boldsymbol{v}(t) + K_P\,(\boldsymbol{X}(t) - \boldsymbol{x}_1)\,, \tag{59}$$

where $K_P > 0$ is a control gain. When the norm of the position error with respect to the end point of a segment is below a small positive threshold, $\|\boldsymbol{X}_B - \boldsymbol{x}_1\| < \epsilon$, the task planner switches to the next desired point in the sequence.

As a measure of the directional error in executing the task, we use the absolute value e_d of the angle δ between the desired and the actual Cartesian velocity, both normalized:

$$\delta = \arccos\left(\frac{\boldsymbol{X}_B - \boldsymbol{x}_1}{\|\boldsymbol{X}_B - \boldsymbol{x}_1\|} \cdot \frac{\boldsymbol{J}_1(\boldsymbol{q})\dot{\boldsymbol{q}}}{\|\boldsymbol{J}_1(\boldsymbol{q})\dot{\boldsymbol{q}}\|}\right), \quad e_d = |\delta|. \tag{60}$$

Figure 5 shows the comparative results obtained with four different redundancy resolution methods for four increasing average desired velocities to complete the cyclic task (specified through decreasing times T_{AB} for moving though a single path segment). The control gain in (59) was set to $K_P = 100$ and the threshold for switching path points to $\epsilon = 10^{-6}$ [m]. The first row refers to the ideal case, used for reference, with joint velocities obtained by pseudoinversion of the task Jacobian $\boldsymbol{J}_1(\boldsymbol{q})$ and without assuming any limit in the joint space (*unconstrained* case). In the second row, the same pseudoinversion control method is applied but without considering the joint constraints that are now present (*standard* case). The third row presents again the pseudoinverse solution, followed by a

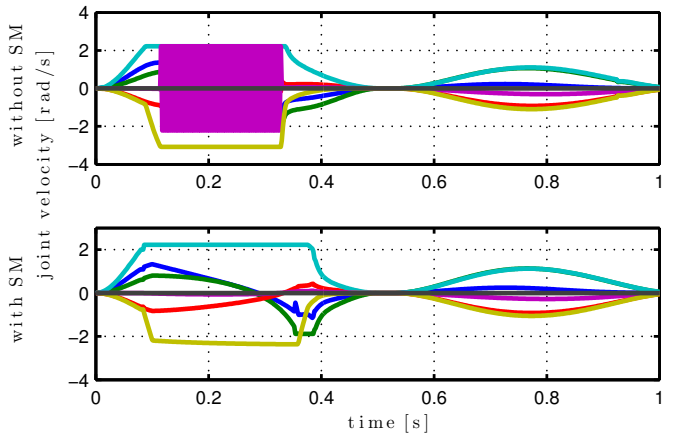

Fig. 6. Joint velocities obtained with the FastOpt-SNS algorithm for the task in Fig. 5 with $T_{AB} = 0.5$ s, without (top) and with (bottom) the introduction of a scale margin $s_m = 0.1$. Only the initial first second of motion is plotted to focus on the problem of discontinuities and their removal.

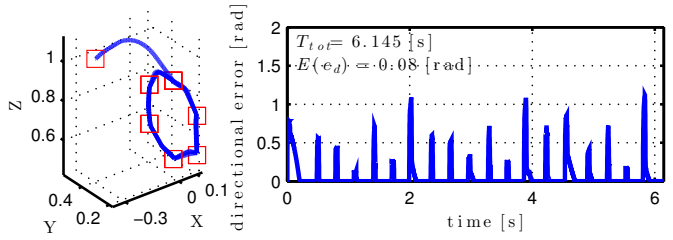

Fig. 7. Execution of the task in Fig. 5 with $T_{AB} = 0.05$ s, with a QP solver that minimizes the task error norm when the task is unfeasible

classical task scaling to recover feasibility (*scale* case). Finally, the last row shows the results obtained with the *FastOpt-SNS* algorithm, using a margin $s_m = 0.1$ (see Sec. V). Each plot displays also the values of the actual task completion time T_{tot} and of the mean directional error $E(e_d)$ over the complete robot motion.

At slow execution speed, the SNS method performs as well as the reference unconstrained case. On the other hand, both the standard solution and pseudoinversion followed by classical scaling show already some trajectory errors and an associated increase in execution time. Note that all schemes are local in nature and follow in general different joint trajectories. Moreover, neither the Standard nor the Scale method take into account joint range limits. The reaching of joint position limits during task execution explains the relative performance in slow motion (up to $T_{AB} = 0.5$ [s]) between the Scale method and the Standard method, where the latter happens to behave slightly better in terms of task accuracy, contrary to what expected. For faster motions, the situation is reversed.

The typical behaviors of each method become more evident and are enhanced dramatically when the task is more demanding. Note in particular that at high task speeds, in order to recover feasibility the classical task scaling slows down considerably the execution time ($T_{tot} = 7.712$ s in the fourth column), as opposed to the use of the SNS method ($T_{tot} = 4.59$ s). Nonetheless, some deformation of the desired path is still present with the classical method since a hard joint limit is reached (see the path error on the right lower part of the hexagon). On the other hand, the SNS solution is practically error free while still undergoing saturation of the joint velocity constraints (12). We can conclude that the SNS method, here in its fast and optimal implementation, is able to exploit at best the available motion capabilities of the robot. More numerical results of similar nature can be found in [23] and [25], as well as in [24] for the multiple task case.

Figure 6 shows the velocities of the seven robot joints obtained when executing the desired task for $T_{AB} = 0.5$ s using the FastOpt-SNS algorithm, with and without a scale margin. The multiple discontinuities that occur when using the algorithm as such disappear altogether thanks to the scale margin workaround (practically, with no extra motion time needed). Finally, Figure 7 refers to the execution of the task with $T_{AB} = 0.05$ s, as obtained when using a competing QP solver that minimizes the norm of the task error when the task is unfeasible[7] instead of resorting to the task scaling embedded in the SNS method (see also Sec. V). The task deformation is quite evident and the total motion time is longer than the one obtained with the FastOpt-SNS algorithm (compare with the last row/column of Fig. 5).

IX. Experimental Results

The effectiveness of the proposed SNS method has been tested also in experiments with the KUKA LWR 4 robot shown in Fig. 1.

First experiment. The single Cartesian task is specified again by eqs. (58–59). We consider the same hexagonal trajectory for the end-effector position $\boldsymbol{p}_{EE}$ and the same control parameters as in the simulations. However, the center of the hexagon is now placed at $\boldsymbol{C} = \begin{pmatrix} -0.3 & 0.35 & 0.8 \end{pmatrix}^T$ [m], so that the task error in the initial robot configuration is zero. When $T_{AB} = 1$ s, the task performed with the Jacobian pseudoinverse is feasible, and no constraints become active. Therefore, *any* QP solver that minimizes the joint velocity norm gives exactly the same solution. The results of such an experiment are reported in Fig 8, where we show also the evolution of auxiliary tasks that are not considered here but only in the second set of experiments.

Second experiment. We introduce an additional second and third task, with priority order. The second (scalar) task x_2 is specified by a point-to-point trajectory for the y-coordinate of the position $\boldsymbol{p}_{EL}$ of the center of the KUKA LWR 4 elbow (on the 4th joint axis), moving from 0 to 0.2 m and using again the time profile (58) with $T_{AB} = 6$ s. The desired task velocity $\dot{x}_2$ is given by (59), with an error feedback gain $K_P = 100$. The third (scalar) task prescribes no displacement of the elbow center along the x direction, thus $\dot{x}_3 = \dot{p}_{EL,x} = 0$.

Figures 9 and 10 show the experimental results obtained with pseudoinversion followed by classical task scaling and with the FastOpt-SNS algorithm, respectively. It can be seen that the third task is not consistent with the other two of higher priority, and therefore its error cannot be kept to zero.

[7]This result is obtained using the Matlab implementation of the HQP method [21], as released in [22].

When using the scaling method, the secondary task displays some errors and, even more critically, also the primary task is deformed geometrically, despite of the large increase in total motion time ($T_{tot} = 21.15$ s). On the contrary, the FastOpt-SNS algorithm executes correctly the second priority task. Moreover, the primary task is completely unaffected by the presence of the second and third tasks. As a matter of fact, the task completion time ($T_{tot} = 18.032$ s) is practically the same obtained when considering the first task alone (Fig. 8). These conclusions are supported by the mean values of the error indices used for each task: $E(e_d)$ for the first task, $E(\|\boldsymbol{e}_2\|)$ for the second, and $E(|\dot{x}_3|)$ for the velocity of the third task.

We remark that, in this multiple tasks case, the cyclic end-effector motion specified as primary task could *not* be completed when using standard pseudoinversion alone, although the secondary task is still executed. The same behavior was observed when including the classical task scaling for a larger control gain K_P. Instead, the FastOpt-SNS algorithm allows to set arbitrarily large values for the gains. Thanks to the preemptive strategy and the embedded task scaling, joint velocity commands will saturate but remain always within the prescribed hard constraints.

In the accompanying video, other comparative experiments are presented. The first experiment above is shown for $T_{AB} = 1$, 0.5, and 0.2 s. In the faster case, the FastOpt-SNS algorithm succeeds whereas the Scale method is blocked by the KUKA controller protection. Another cyclic experiment involves a secondary task with a desired sinusoidal motion of the robot elbow. The FastOpt-SNS algorithm is always superior in performance. More experiments for single or multiple tasks, including also a CS task, are reported in [24], while an application of the SNS method to a Cartesian collision avoidance task has been shown in [23]. The videos of the related experiments are available in IEEE Xplore.

X. Performance Evaluation

In order to evaluate the real-time performance of the *Opt-SNS*, *Fast-SNS*, and *FastOpt-SNS* algorithms, we have considered a high-dimensional planar snake robot with a parametric number n of revolute joints and links of unitary lengths, as shown in Fig. 1. From a computational point of view, the planar case shares similar properties of a 3D situation. It is used here only to simplify the generation of a parametric form for the robot kinematics. All SNS versions have been developed in C++, using the Eigen library [36] for algebraic computations. Numerical simulations were performed using the ROS environment [37] on a Intel Core i7-2600 CPU 3.4GHz, with 8Gb of RAM.

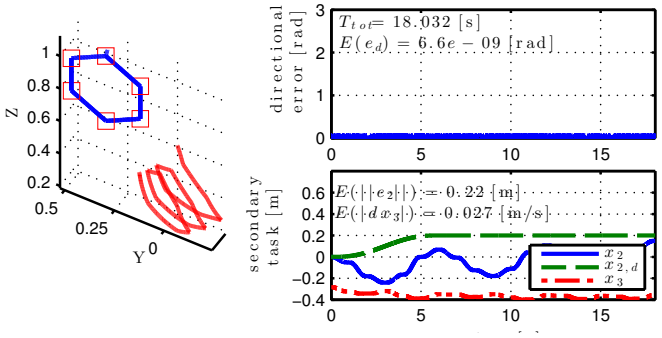

Fig. 8. Execution of a single task using a generic redundancy resolution algorithm. [Left] Cartesian 3D view of the executed end-effector (blue) and elbow (red) paths. [Right, top] Directional task error e_d, with task completion time and mean directional error. [Right, bottom] Components x (dot-dashed, green) and y (solid, blue) of the robot elbow trajectory.

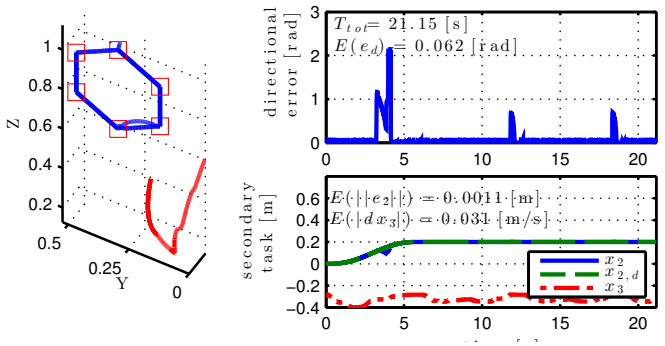

Fig. 9. Execution of the three tasks using pseudoinversion and classical task scaling. [Left] Cartesian 3D view of the executed end-effector (blue) and elbow (red) paths. [Right, top] Directional task error e_d, with task completion time and mean directional error. [Right, bottom] Evolution of the desired (dashed, green) and actual (solid, blue) second task and of the third task, both related to the robot elbow trajectory. The mean values of error norms for two lower priority tasks are also shown.

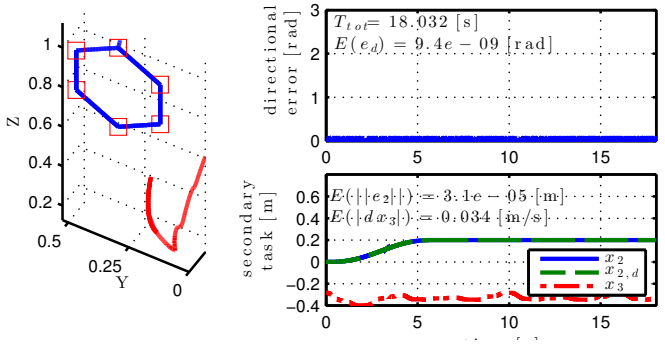

Fig. 10. Execution of the three tasks using the FastOpt-SNS algorithm. [Left] Cartesian 3D view of the executed end-effector (blue) and elbow (red) paths. [Right, top] Directional task error e_d, with task completion time and mean directional error. [Right, bottom] Evolution of the desired (dashed, green) and actual (solid, blue) second task and of the third task, both related to the robot elbow trajectory. The mean values of error norms for two lower priority tasks are also shown.

First test. The single task considered for the n-DOF snake robot is a planar positioning of its end-effector ($m_1 = 2$). Symmetric joint limits (11) have been chosen as

$$\begin{aligned} Q_{max,i} &= -Q_{min,i} = 90 \text{ [deg]}, \\ V_{max,i} &= 1 \text{ [deg/s]}, \quad A_{max,i} = 3 \text{ [deg/s}^2\text{]}, \end{aligned} \tag{61}$$

for $i = 1, \ldots, n$. The (non-time based) task velocity is defined by

$$\dot{\boldsymbol{x}} = V_C \sin\left(\left(1 - \frac{\|\boldsymbol{x} - \boldsymbol{x}_d\|}{\|\boldsymbol{x}_0 - \boldsymbol{x}_d\|}\right)\pi + \varepsilon\right) \frac{\boldsymbol{x} - \boldsymbol{x}_d}{\|\boldsymbol{x}_0 - \boldsymbol{x}_d\|}, \tag{62}$$

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 $\boldsymbol{q}_0 = \mathbf{0}$, corresponding to $\boldsymbol{x}_0 = \begin{pmatrix} n & 0 \end{pmatrix}^T$. The desired Cartesian position of the tip of a generic link $r \in \{1, \ldots, n\}$ is specified by

$$\boldsymbol{x}_d(r) = \begin{pmatrix} \frac{\sqrt{2}}{2} r & \frac{\sqrt{2}}{2} r \end{pmatrix}^T, \tag{63}$$

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

Figure 11 shows in semi-logarithmic scale the execution times needed in the worst case by different algorithms proposed in this paper, when increasing the number n of robot DOFs. We compared these also with *qpOASES* [38], a state-of-the-art QP active-set solver with warm start. Both the QP solver and the Opt-SNS algorithm become too slow when n increases. Assuming 10 ms as the upper limit on execution time for on-line uses, Opt-SNS and qpOASIS are thus not adequate for a hyper-redundant robot with more than 40 DOFs, while the Fast-SNS and the FastOpt-SNS algorithms can handle up to 100 DOFs.

Second test. The number of DOFs of the robot has been 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 a link r in the kinematic chain, as given by eq. (63). 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 (61) of the first test have been used, with the same type of desired velocity law (62) for each task.

The execution times needed in the worst case by the different algorithms are shown in Fig. 12. Execution time grows linearly with the number of tasks for the state-of-the-art QP solver, while it increases only slightly for all versions of the SNS method. In particular, the two *Fast* versions run in the worst case in 2 ms even when 10 tasks are considered,

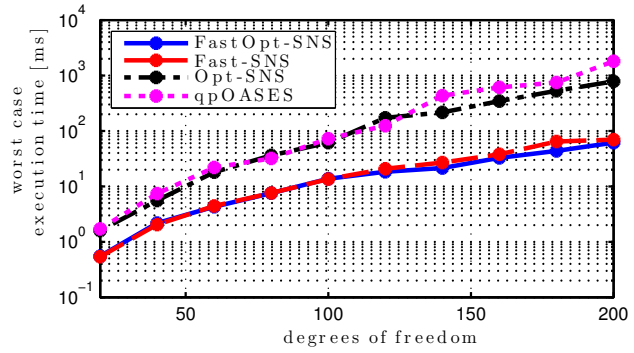

Fig. 11. Worst-case execution times to accomplish with different algorithms a single two-dimensional task for the n-DOF snake robot of Fig. 1, when varying n from 20 to 200. The problem consists of $m_1 = 2$ task equalities and an equivalent of $4n$ box inequalities. Semilog scale is used in view of the differences by one or more order of magnitude.

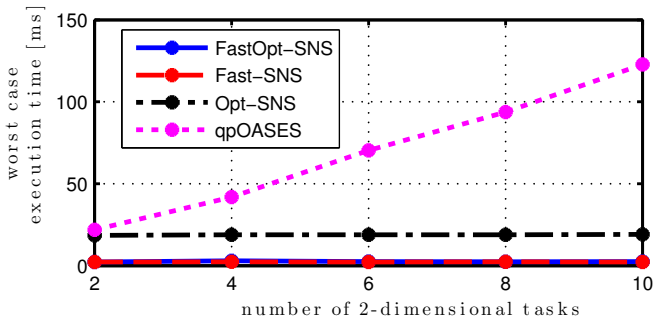

Fig. 12. Worst-case execution times to accomplish with different algorithms l prioritized two-dimensional tasks with the snake robot of Fig. 1 having $n = 50$ DOFs, when varying l from 2 to 10. The problem consists of a total of $2l$ task equalities and an equivalent of 200 box inequalities.

thus being eligible for real-time control applications[8].

XI. Conclusions

We have presented a local, possibly sensor-based approach for solving the inverse differential task kinematics of a redundant robot under hard joint constraints, the Saturation in the Null Space (SNS) method. In the SNS framework, different algorithms of slightly increasing complexity have been introduced in an incremental way, leading to optimal joint velocity solutions with guaranteed performance and numerically efficient implementations. In case of multiple tasks ordered by priority, the SNS realizes a correct preemptive strategy, i.e., tasks of higher priority use in the best way the feasible robot capabilities they need, while lower priority tasks are accommodated with the residual capability and do not interfere with the execution of higher priority tasks. A main feature of all the presented algorithms is the automatic integration of a (multi-task) least possible scaling strategy, when some of the original tasks are found to be unfeasible for the robot capabilities. Except for physically impossible tasks (because of spatial/geometric or temporal discontinuity), task directionality will always be preserved, which is extremely important in several robotic applications, e.g., in sensor-based human-robot collision avoidance.

The effectiveness and performance of the algorithms have been assessed through several numerical simulations and experiments, showing that the method allows real-time control of robots with high-dimensional configuration spaces executing a large number of prioritized tasks, and with a large number of hard joint constraints that may saturate during motion. The method has been presented here using joint velocity commands, but its extension to acceleration (see [23]) or even torque commands is rather straightforward.

The performance improvements obtained with the fast, optimal version of the SNS algorithm with respect to the use of state-of-the-art, general (and possibly hierarchical) QP solvers addressing the same constrained optimization problem deserves a final comment. The SNS method treats the hard joint constraints separately from the hierarchy of (equality)

[8]A video showing animations of the 50-DOF snake robot for $l = 1$, $l = 3$, and $l = 5$ tasks with priority when using the FastOpt-SNS algorithm is available in IEEE Xplore, associated to [26].

tasks. In this way, the operations involved in saturating and desaturating commands are largely simplified. On the other hand, in order to guarantee the same behavior with [20]–[22], the hard inequality constraints on joint motion have to be assigned the highest priority in the stack of tasks. Therefore, any change in the active set of inequality constraints will be reflected in a re-computation through the whole stack. Moreover, the case of unfeasible tasks is directly and clearly addressed with the SNS method, while it requires forcing some modifications in QP methods. On the other hand, QP methods allow more general formulations including also unilateral task space constraints, out of which the joint level constraints are only a particular case. In this framework, the SNS approach has to resort to less elegant approximations in order to handle general unilateral constraints (see, e.g. [39]).

Finally, in presenting our results, we took advantage of the extensive experience gained in the last two years or so. The SNS method was in fact implemented seamlessly on three different robotic systems in Stanford, in Rome, and at CNRS-LAAS in Toulouse (within the SAPHARI European project), proving further the robustness of the proposed solution.

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] 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.

[4] C. Samson, M. L. Borgne, and B. Espiau, *Robot Control: The Task Function Approach*. Clarendon, 1991.

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

[6] A. S. Deo and I. D. Walker, "Minimum effort inverse kinematics for redundant manipulators," *IEEE Trans. on Robotics and Automation*, vol. 13, no. 5, pp. 767–775, 1997.

[7] P. Chiacchio and S. Chiaverini, "Coping with joint velocity limits in first-order inverse kinematics algorithms: Analysis and real-time implementation," *Robotica*, vol. 13, no. 9, pp. 515–519, 1995.

[8] G. Antonelli, S. Chiaverini, and G. Fusco, "A new on-line algorithm for inverse kinematics of robot manipulators ensuring path tracking capability under joint limits," *IEEE Trans. on Robotics and Automation*, vol. 19, no. 1, pp. 162–167, 2003.

[9] R. V. Dubey, J. A. Euler, and S. M. Babcock, "Real-time implementation of an optimization scheme for seven-degree-of freedom redundant manipulators," *IEEE Trans. on Robotics and Automation*, vol. 7, no. 5, pp. 579–588, 1991.

[10] 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.

[11] G. Antonelli, S. Chiaverini, and G. Fusco, "Kinematic control of redundant manipulators with on-line end-effector path tracking capability under velocity and acceleration constraints," in *Proc. 6th IFAC Symp. on Robot Control*, 2000, pp. 609–614.

[12] F. Chaumette and E. Marchand, "A redundancy-based iterative scheme for avoiding joint limits: Application to visual servoing," *IEEE Trans. on Robotics and Automation*, vol. 17, no. 5, pp. 719–730, 2001.

[13] D. Omrcen, L. Zlajpah, and B. Nemec, "Compensation of velocity and/or acceleration joint saturation applied to redundant manipulator," *Robotics and Autonomous Systems*, vol. 55, no. 4, pp. 337–344, 2007.

[14] 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.

[15] 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.

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

[17] P. Gill, W. Murray, and M. Wright, *Practical Optimization*. Emerald, 1982.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] ——, "Hierarchical quadratic programming: Fast online humanoid-robot motion generation," *The International Journal of Robotics Research,*, vol. 7, no. 33, pp. 1006–1028, 2014.

[23] 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.

[24] 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.

[25] 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. 3969–3975.

[26] ——, "Fast redundancy resolution for high-dimensional robots executing prioritized tasks under hard bounds in the joint space," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2013, pp. 2500–2506.

[27] 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.

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

[29] 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.

[30] 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.

[31] 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.

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

[33] O. Kanoun, "Real-time prioritized kinematic control under inequality constraints for redundant manipulators," in *Robotics: Science and Systems VII*, H. Durrant-Whyte, N. Roy, and P. Abbeel, Eds. MIT Press, 2012, pp. 145–152.

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

[35] H. W. Kuhn and A. Tucker, "Nonlinear programming," in *Proc. 2nd Berkeley Symp. on Mathematical Statistics and Probability*, 1951, pp. 481–492.

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

[37] 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.

[38] 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.

[39] 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.