Prioritized Multi-Task Motion Control of Redundant Robots under Hard Joint Constraints

Fabrizio Flacco* Alessandro De Luca* Oussama Khatib**

Abstract— **We present an efficient method for motion control of redundant robots performing multiple prioritized tasks in the presence of hard bounds on joint range, velocity, and acceleration/torque. This is an extension of our recently proposed SNS (Saturation in the Null Space) algorithm developed for single tasks. The method is defined at the level of acceleration commands and proceeds by successively discarding one at a time the commands that would exceed their bounds for a task of given priority, and reintroducing them at their saturated levels by projection in the null space of a suitable Jacobian associated to the already considered tasks. When processing all tasks in their priority order, a correct preemptive strategy is realized in this way, i.e., a task of higher priority uses in the best way the feasible robot capabilities it needs, while lower priority tasks are accommodated with the residual capability and do not interfere with the execution of higher priority tasks. The algorithm automatically integrates a multi-task least possible scaling strategy, when some ordered set of original tasks is found to be unfeasible. Simulation and experimental results on a 7-dof lightweight KUKA LWR IV robot illustrate the good performance of the method.**

I. Introduction

The capability of handling multiple tasks is one of the most appealing features of kinematically redundant robots. Simultaneous control of a series of prioritized tasks is typically achieved using generalized inversion (most often, by pseudoinversion of the Jacobian) of the differential task kinematics, combined with projections in suitable null spaces so as to preserve as much as possible the execution of higher priority tasks [1]–[3]. On-line (and thus local) solutions that lend themselves to sensor-based control without the need of future information are preferred, e.g., for dealing with the time-varying and unpredictable nature of physical Human-Robot Interaction (pHRI) tasks.

In this framework, *hard constraints* imposed in the joint space (bounds on the joint range, velocity, acceleration, or even torque) are barely taken into account, at least explicitly. In fact, such hard bounds are typically converted into soft ones, resolving redundancy by task constrained optimization of suitable objective functions (e.g., keeping the joints closer to their range centers [4], [5], or using the joint ranges to weight the pseudoinversion [6] or to define an infinity norm to be minimized at the velocity level [7]). However, the commanded joint motion may still saturate some of the bounds, producing then an unpredictable robot motion. Furthermore, in case of multiple tasks, it may happen that the command contribution needed to execute lower priority tasks leads to exceeding some bounds. Their saturation destroys also the execution of tasks in the correct priority.

A simple way to recover feasibility with respect to the given bounds is by task scaling, i.e., reducing the speed/acceleration of the (single or multiple) task commands. Task relaxation by time scaling has been used for satisfying joint velocity [8] and/or acceleration [9] bounds. In [10], the velocity term in the 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., [11]), using again the mechanism of projection in the null space of the task Jacobians.

Nonetheless, before resorting to task scaling, it would be useful to verify whether we can generate alternative joint motions that still execute the original task while satisfying the hard joint constraints (and preserving prioritization in case of multiple tasks). This obviously requires a smart exploitation of the null space of the task Jacobian(s). A method that explicitly handles joint velocity or acceleration bounds in a redundant robot performing a single task has been introduced in [12]. All joint commands exceeding their bounds are simultaneously pushed back at their saturation levels. This effect is then compensated by the selection of a null space contribution intended to satisfy the task. However, no solution is given in case of unfeasible tasks and the method is not extended to multiple tasks. A similar approach had been proposed for the case of multiple tasks in the animation of avatars [13], where only joint range limits were considered. Feasibility with respect to these bounds is verified only after adding the contributions of all tasks, and if not all tasks can be executed within the bounds, the resulting task deformation is spanned to all tasks and not just to the low priority tasks that produced the violation of bounds.

In [14], we proposed a new method, named *SNS* (Saturation in the Null Space), for controlling the motion of a redundant robot performing a single task under hard bounds on the joint range, joint velocity, and joint acceleration. Commands at the velocity or acceleration level were considered. The *SNS* algorithm disables successively only one exceeding command at the time, reintroducing it at its saturated level through the projection in the null space of the task Jacobian. Moreover, the algorithm automatically integrates the use of the least possible task scaling, only when the original task is found to be unfeasible.

The main goal of this paper is to extend the *SNS* approach to the multi-task case, considering task priorities and the

*Dipartimento di Ingegneria informatica, automatica e gestionale Antonio Ruberti, Università di Roma "La Sapienza", Via Ariosto 25, 00185 Rome, Italy {fflacco,deluca}@dis.uniroma1.it). **Artificial Intelligence Laboratory, Stanford University, Stanford, CA 94305, USA khatib@cs.stanford.edu. The work of the first two authors is supported by the European Community, within the FP7 ICT-287513 SAPHARI project.

same previous set of hard joint constraints. In doing so, we follow the idea of *preemptive* prioritization: A higher priority task should use all the feasible robot capabilities it needs, while a lower priority task must preserve the execution of high priority tasks using only the residual capabilities not used by all tasks of higher priority. This reasonable requirement is very well addressed by our solution method. As a matter of fact, the *SNS* algorithm typically ends up with using a smaller number of saturated commands after satisfying each task in the priority scale and this leaves more room for the additional satisfaction of lower priority tasks. Moreover, automatic scaling of task(s) is again seamlessly embedded in the algorithm only when needed.

The considered problem can also be recast and tackled as a constrained minimization of a quadratic objective function under linear equality/inequality constraints with different priorities, as done in [15], [16]. However, and in contrast to these works, the inequality constraints imposed at the joint level (including those on the commands) cannot and will never be violated in our approach. These constraints, which define what we call *robot capabilities*, are handled separately and do not need to be allocated in the stack of prioritized tasks. To guarantee feasibility, we include instead the possibility of task scaling, which is not considered in [15], [16]. Moreover, the inequality constraints in our problem have the form of elementary bounds (box constraints). This problem structure, as well as the activation of one joint constraints at the time in the SNS algorithm, is exploited so as to lead to a computationally efficient numerical solution.

The paper is organized as follows. The redundancy formalism used throughout the paper is introduced in Sect. II. Section III presents a simple motivating example where the effects of joint acceleration saturation in a multi-task scenario are correctly handled by our method. Section IV recalls the *SNS* algorithm proposed in [14] for a single task. This method is extended to multiple tasks in Sect. V. The special case of (lower priority) tasks specified in the whole configuration space of the robot is presented in Sect. VI. The effectiveness of the *SNS* algorithm is shown by Matlab$^{\mathrm{TM}}$ simulations and experiments on a 7-dof KUKA LWR IV robot, respectively in Sect. VII and Sect. VIII.

II. Notation and Background

Let $\boldsymbol{q} \in \mathbb{R}^n$ be the vector of generalized (joint) coordinates of a robot, $\boldsymbol{x} \in \mathbb{R}^m$ the vector of variables describing a generic m-dimensional task, with $m < n$, and $\boldsymbol{J}(\boldsymbol{q})$ the associated $m \times n$ task Jacobian matrix. At a given $(\boldsymbol{q}, \dot{\boldsymbol{q}})$, the direct second-order differential relation and its (minimum norm) inverse are

$$\ddot{\boldsymbol{x}} = \boldsymbol{J}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}(\boldsymbol{q})\dot{\boldsymbol{q}}, \qquad \ddot{\boldsymbol{q}} = \boldsymbol{J}^{\#}(\boldsymbol{q})\left(\ddot{\boldsymbol{x}} - \dot{\boldsymbol{J}}(\boldsymbol{q})\dot{\boldsymbol{q}}\right), \quad (1)$$

where $(\cdot)^{\#}$ denotes pseudoinversion.

Consider l acceleration tasks $\ddot{\boldsymbol{x}}_k$, $k = 1 \dots l$, each of dimension $m_k < n$ and ordered by priority, i.e., task i has a higher priority than task j if $i < j$. The multi-task motion control with priority can be described using the recursive approach proposed in [1]. We have (dropping dependencies):

$$\begin{aligned} \ddot{\boldsymbol{q}}_0 &= \boldsymbol{0} \\ \ddot{\boldsymbol{q}}_k &= \ddot{\boldsymbol{q}}_{k-1} + \boldsymbol{a}_k \\ &= \ddot{\boldsymbol{q}}_{k-1} + (\boldsymbol{J}_k\boldsymbol{P}_{k-1})^{\#}\left(\ddot{\boldsymbol{x}}_k - \dot{\boldsymbol{J}}_k\dot{\boldsymbol{q}} - \boldsymbol{J}_k\ddot{\boldsymbol{q}}_{k-1}\right), \end{aligned} \quad (2)$$

In (2), $\boldsymbol{J}_k$ is the Jacobian associated to task k and $\boldsymbol{P}_k$ is the projector operator in the null space of the (augmented) Jacobian of the first k tasks

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

The generalized joint acceleration $\ddot{\boldsymbol{q}}_k$ performs the first k tasks with the given priority, while $\boldsymbol{a}_k$ is the modification of the acceleration needed to perform also task k, starting from a solution for the first $k-1$ tasks. The joint acceleration addressing all l tasks is $\ddot{\boldsymbol{q}} = \ddot{\boldsymbol{q}}_l$.

The following recursive formula, useful for obtaining the projector $\boldsymbol{P}_k$ without recomputing the null space of the augmented Jacobian for each additional task, has been proposed in [13]:

$$\begin{aligned} \boldsymbol{P}_0 &= \boldsymbol{I} \\ \boldsymbol{P}_k &= \boldsymbol{P}_{k-1} - (\boldsymbol{J}_k\boldsymbol{P}_{k-1})^{\#}\boldsymbol{J}_k\boldsymbol{P}_{k-1}, \end{aligned} \quad (3)$$

where $\boldsymbol{I}$ is the $n \times n$ identity matrix. To deal with singularities, it is customary to use damped pseudoinversion, with a selective damping on the lowest singular values (e.g., the numerical filtering method of [17]).

III. Illustrative Example

Consider a planar 4R manipulator with equal links of unitary length performing a primary task specified by a desired acceleration $\ddot{\boldsymbol{x}}_1 \in \mathbb{R}^2$ ($m_1 = 2$) for its end-effector and commanded by the joint acceleration $\ddot{\boldsymbol{q}} \in \mathbb{R}^4$ ($n = 4$). The degree of redundancy for this task is $n - m = 2$. Without loss of generality assume the robot at rest, i.e., $\dot{\boldsymbol{q}} = \boldsymbol{0}$. Suppose that the joint accelerations are bounded as $|\ddot{q}_i| \le A_i$, $i = 1, \dots, 4$, with $A_1 = A_2 = 2$, $A_3 = A_4 = 4$ [rad/s^2].

The 2×4 Jacobian $\boldsymbol{J}_1(\boldsymbol{q})$ in the differential map (1) evaluated at $\boldsymbol{q} = \begin{pmatrix} \pi/2 & -\pi/2 & \pi/2 & -\pi/2 \end{pmatrix}^T$ is

$$\boldsymbol{J}_1 = \begin{pmatrix} -2 & -1 & -1 & 0 \\ 2 & 2 & 1 & 1 \end{pmatrix}.$$

For a desired task acceleration $\ddot{\boldsymbol{x}}_1 = \begin{pmatrix} -3 & -1.5 \end{pmatrix}^T$, the minimum norm joint acceleration solution is

$$\ddot{\boldsymbol{q}}_1 = \boldsymbol{J}_1^{\#}\ddot{\boldsymbol{x}}_1 = \begin{pmatrix} 1.9091 & -1.7727 & 0.9545 & -2.7273 \end{pmatrix}^T,$$

which is within the joint acceleration bounds and thus executable by the robot.

Consider a secondary scalar task ($m_2 = 1$) specified by a desired acceleration $\ddot{x}_2 = 1$ along the y direction for the tip of the second link. At the given configuration $\boldsymbol{q}$, the 1×4 Jacobian $\boldsymbol{J}_2$ associated to this secondary task is

$$\boldsymbol{J}_2 = \begin{pmatrix} 1 & 1 & 0 & 0 \end{pmatrix}.$$

By applying the task priority algorithm (2–3), the secondary task is projected in the null space of the primary task obtaining the joint acceleration

$$\begin{aligned}\ddot{\boldsymbol{q}} &= \ddot{\boldsymbol{q}}_1 + (\boldsymbol{J}_2\boldsymbol{P}_1)^{\#}(\ddot{x}_2 - \boldsymbol{J}_2\ddot{\boldsymbol{q}}_1)\\ &= \begin{pmatrix} 2.125 & -1.125 & -0.125 & -3.375 \end{pmatrix}^T,\end{aligned}$$

which exceeds the acceleration bound at the first joint. Indeed, if the robot is commanded in this way, the actual joint acceleration applied to the robot would have the saturated value $\ddot{q}_1 = A_1 = 2$ in place of the computed one and, as a result, neither the first nor the second task will be achieved.

Since the higher priority task is by itself executable, the standard solution that would preserve the priority of the two tasks is to consider modified acceleration bounds for the task of lower priority, taking into account the acceleration $\ddot{\boldsymbol{q}}_1$ already requested by the primary task, and then to scale the desired acceleration for the secondary task so as to satisfy the new bounds. We scale then the acceleration of the secondary task by a factor $s_2 \in (0,1)$ so that the joint acceleration modification (see eq. (2))

$$\boldsymbol{a}_2 = (\boldsymbol{J}_2\boldsymbol{P}_1)^{\#}(s_2\ddot{x}_2 - \boldsymbol{J}_2\ddot{\boldsymbol{q}}_1),$$

satisfies the modified bounds $-A_i - \ddot{q}_{1,i} \le a_{2,i} \le A_i - \ddot{q}_{1,i}$, for $i = 1,\ldots,4$. This results in a scale factor $s_2 = 0.5$ and the final joint acceleration $\ddot{\boldsymbol{q}}_1 + \boldsymbol{a}_2$ is

$$\ddot{\boldsymbol{q}}_{scaled} = \begin{pmatrix} 2 & -1.5 & 0.5 & -3 \end{pmatrix}^T.$$

The primary task is fully executed, while only 50% of the secondary task can be performed.

On the other hand, applying our *SNS* method (**Algorithm 3** in Sect. V), it is still possible to find a joint acceleration

$$\ddot{\boldsymbol{q}}_{SNS} = \begin{pmatrix} 2 & -1 & 0 & -3.5 \end{pmatrix}^T$$

that realizes exactly both original tasks while satisfying the given joint acceleration bounds. In fact,

$$\boldsymbol{J}_1\ddot{\boldsymbol{q}}_{SNS} = \begin{pmatrix} -3 & -1.5 \end{pmatrix}^T = \ddot{\boldsymbol{x}}_1, \quad \boldsymbol{J}_2\ddot{\boldsymbol{q}}_{SNS} = 1 = \ddot{x}_2.$$

IV. Single Task SNS

In this section we recall the *SNS* method proposed in [14], rewritten explicitly at the acceleration control level and for a single task.

A. Shaping the joint acceleration bounds

We define the robot capabilities through the following bounds on the joint ranges, joint velocities, and joint accelerations:

$$\begin{gathered}\boldsymbol{Q}_{min} \le \boldsymbol{q} \le \boldsymbol{Q}_{max}, \quad -\boldsymbol{V}_{max} \le \dot{\boldsymbol{q}} \le \boldsymbol{V}_{max},\\ \boldsymbol{A}_{min} \le \ddot{\boldsymbol{q}} \le \boldsymbol{A}_{max}.\end{gathered} \tag{4}$$

All the above inequalities are intended component-wise. Joint ranges need not to be symmetric, while velocity bounds typically are. If the accelerations bounds come from pure kinematic reasoning, it is $\boldsymbol{A}_{min} = -\boldsymbol{A}_{max}$. If these bounds arise instead from symmetric actuator torque bounds, i.e., $|\tau_i| \le T_{max,i}$, for $i = 1,\ldots,n$, then at the robot state $(\boldsymbol{q}.\dot{\boldsymbol{q}})$

$$\boldsymbol{A}_{min} = -\boldsymbol{M}^{-1}(\boldsymbol{q})\,(\boldsymbol{T}_{max} + \boldsymbol{n}(\boldsymbol{q},\dot{\boldsymbol{q}}))$$

and

$$\boldsymbol{A}_{max} = \boldsymbol{M}^{-1}(\boldsymbol{q})\,(\boldsymbol{T}_{max} - \boldsymbol{n}(\boldsymbol{q},\dot{\boldsymbol{q}})),$$

where $\boldsymbol{M}$ is the robot inertia matrix and $\boldsymbol{n}$ collects centrifugal, Coriolis, and gravity terms.

In control implementations, the joint acceleration command is kept constant at the computed value $\ddot{\boldsymbol{q}} = \ddot{\boldsymbol{q}}_h = \ddot{\boldsymbol{q}}(t_h)$ for a sampling time of duration T. Suppose that at $t_h = hT$ the current joint position $\boldsymbol{q} = \boldsymbol{q}_h$ and velocity $\dot{\boldsymbol{q}} = \dot{\boldsymbol{q}}_h$ are both feasible. The next joint velocity and position

$$\dot{\boldsymbol{q}}_{h+1} \simeq \dot{\boldsymbol{q}}_h + \ddot{\boldsymbol{q}}\,T, \quad \boldsymbol{q}_{h+1} \simeq \boldsymbol{q}_h + \dot{\boldsymbol{q}}_h T + \frac{1}{2}\ddot{\boldsymbol{q}}\,T^2$$

have to be kept within their bounds. Thus, we obtain

$$-\frac{\boldsymbol{V}_{max} + \dot{\boldsymbol{q}}_h}{T} \le \ddot{\boldsymbol{q}} \le \frac{\boldsymbol{V}_{max} - \dot{\boldsymbol{q}}_h}{T} \tag{5}$$

and

$$\frac{2\,(\boldsymbol{Q}_{min} - \boldsymbol{q}_h - \dot{\boldsymbol{q}}_h T)}{T^2} \le \ddot{\boldsymbol{q}} \le \frac{2\,(\boldsymbol{Q}_{max} - \boldsymbol{q}_h - \dot{\boldsymbol{q}}_h T)}{T^2}. \tag{6}$$

Considering the constraints given by the third set of inequalities in (4), and those in (5) and (6), we obtain a box constraint for the command $\ddot{\boldsymbol{q}}$ at time $t = t_h$ (see also [14])

$$\ddot{\boldsymbol{Q}}_{min}(t_h) \le \ddot{\boldsymbol{q}} \le \ddot{\boldsymbol{Q}}_{max}(t_h). \tag{7}$$

B. The SNS algorithm for a single task

Consider a robot with n joints performing a single m_1-dimensional desired acceleration task $\ddot{\boldsymbol{x}}_1$, with $m_1 < n$. At a given $(\boldsymbol{q},\dot{\boldsymbol{q}})$, the *SNS* algorithm for realizing the task at the acceleration level under the box constraints (7) is given in pseudocode form by **Algorithm 1**.

Therein, the $n \times n$ selection matrix $\boldsymbol{W}_1 = \mathrm{diag}\{W_{1,ii}\}$ with 0/1 elements is used to specify which joints are currently enabled or disabled: if $W_{1,ii} = 0$, then the acceleration of joint i is set at its saturation level and the joint is disabled (for norm minimization purposes). The algorithm is initialized with $\boldsymbol{W}_1 = \boldsymbol{I}$ (the identity matrix), a null-space vector $\ddot{\boldsymbol{q}}_{N,1} = \boldsymbol{0}$, and two scaling factors $s_1 = 1$ and $s_1^* = 0$. Also, we denote by $\bar{\ddot{\boldsymbol{q}}}_1$ (with an additional bar) the current guess of joint acceleration. The core (and final) joint acceleration command computed by this algorithm uses the *SNS* projection equation

$$\ddot{\boldsymbol{q}}_{SNS} = \ddot{\boldsymbol{q}}_{N,1} + (\boldsymbol{J}_1\boldsymbol{W}_1)^{\#}\left(s_1\ddot{\boldsymbol{x}}_1 - \dot{\boldsymbol{J}}_1\dot{\boldsymbol{q}} - \boldsymbol{J}_1\ddot{\boldsymbol{q}}_{N,1}\right). \tag{8}$$

Another important aspect is the integrated use of the task scaling factor s_1, which is eventually chosen as the largest possible one (i.e., equal to 1 if the task is feasible) that is compatible with the box constraints (7). If some of the joint accelerations (8) exceed their bounds, **Algorithm 2** is called to evaluate the best task scaling factor only among the enabled joints. If the jth joint is the most critical for task execution, i.e., its acceleration needs the largest relative decrease to stay within the bounds, this acceleration is saturated and we set $W_{1,jj} = 0$. Algorithm 1 stops when $\mathrm{rank}(\boldsymbol{J}_1\boldsymbol{W}_1) < m_1$, providing as output the best feasible solution found. Further analysis of the properties of this algorithm and of the obtained solution is provided in [14].

Algorithm 1 (*SNS* at the acceleration level for a single task)

$\boldsymbol{W}_1 = \boldsymbol{I}$, $\ddot{\boldsymbol{q}}_{N,1} = \boldsymbol{0}$, $s_1 = 1$, $s_1^* = 0$
repeat
  limit_exceeded = FALSE
  $\ddot{\bar{\boldsymbol{q}}}_1 = \ddot{\boldsymbol{q}}_{N,1} + (\boldsymbol{J}_1\boldsymbol{W}_1)^{\#}\left(\ddot{\boldsymbol{x}}_1 - \dot{\boldsymbol{J}}_1\dot{\boldsymbol{q}} - \boldsymbol{J}_1\ddot{\boldsymbol{q}}_{N,1}\right)$
  if $\left\{ \begin{array}{l} \exists\ i \in [1:n] : \\ \ddot{\bar{q}}_{1,i} < \ddot{Q}_{min,i} \text{ .OR. } \ddot{\bar{q}}_{1,i} > \ddot{Q}_{max,i} \end{array} \right\}$ **then**
    limit_exceeded = TRUE
    $\boldsymbol{a} = (\boldsymbol{J}_1\boldsymbol{W}_1)^{\#}\,\ddot{\boldsymbol{x}}_1$
    $\boldsymbol{b} = \ddot{\bar{\boldsymbol{q}}}_1 - \boldsymbol{a}$
    getTaskScalingFactor($\boldsymbol{a}$, $\boldsymbol{b}$) (∗**call Algorithm 2**∗)
    if {task scaling factor} $> s_1^*$ **then**
      $s_1^* =$ {task scaling factor}
      $\boldsymbol{W}_1^* = \boldsymbol{W}_1$, $\ddot{\boldsymbol{q}}_{N,1}^* = \ddot{\boldsymbol{q}}_{N,1}$
    end if
    $j =$ {the most critical joint}
    $W_{1,jj} = 0$
    $\ddot{q}_{N,1,j} = \left\{ \begin{array}{ll} \ddot{Q}_{max,j} & \text{if } \ddot{\bar{q}}_{1,j} > \ddot{Q}_{max,j} \\ \ddot{Q}_{min,j} & \text{if } \ddot{\bar{q}}_{1,j} < \ddot{Q}_{min,j} \end{array} \right.$
    if rank$(\boldsymbol{J}_1\boldsymbol{W}_1) < m_1$ **then**
      $s_1 = s_1^*$, $\boldsymbol{W}_1 = \boldsymbol{W}_1^*$, $\ddot{\boldsymbol{q}}_{N,1} = \ddot{\boldsymbol{q}}_{N,1}^*$
      $\ddot{\bar{\boldsymbol{q}}}_1 = \ddot{\boldsymbol{q}}_{N,1} + (\boldsymbol{J}_1\boldsymbol{W}_1)^{\#}\left(s_1\ddot{\boldsymbol{x}}_1 - \dot{\boldsymbol{J}}_1\dot{\boldsymbol{q}} - \boldsymbol{J}_1\ddot{\boldsymbol{q}}_{N,1}\right)$
      limit_exceeded = FALSE *(∗outputs solution∗)*
    end if
  end if
until limit_exceeded = TRUE
$\ddot{\boldsymbol{q}}_{SNS} = \ddot{\bar{\boldsymbol{q}}}_1$

Algorithm 2 (Task scaling factor at the acceleration level)

function getTaskScalingFactor($\boldsymbol{a}$, $\boldsymbol{b}$)
for $i = 1 \rightarrow n$ **do**
  $S_{min,i} = \left(\ddot{Q}_{min,i} - b_i\right)/a_i$
  $S_{max,i} = \left(\ddot{Q}_{max,i} - b_i\right)/a_i$
  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 = $\text{argmin}_i \{S_{max,i}\}$
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

V. Multiple Tasks SNS

Consider l acceleration tasks $\ddot{\boldsymbol{x}}_k$, of dimension $m_k < n$ and with task Jacobian matrix $\boldsymbol{J}_k$, for $k = 1, \ldots, l$. Tasks are ordered from the highest to the lowest priority. We extend the algorithm proposed in Sect. IV to this situation, by imposing a *preemptive* prioritization strategy: 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 3** is the pseudocode of the acceleration-level *SNS* method for the multi-task case.

In this algorithm, we denote with $\ddot{\bar{\boldsymbol{q}}}_k$ the joint acceleration that satisfies at best the first $k-1$ tasks (and is *feasible* w.r.t. the joint constraints) and includes the current joint acceleration guess for addressing task k. At each loop over tasks ($k = 1, \ldots, l$), the initializations of matrix $\boldsymbol{W}_k$, of null-space vector $\ddot{\boldsymbol{q}}_{N,k}$, 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 $\bar{\boldsymbol{P}}_k = \boldsymbol{P}_{k-1}$.

The joint acceleration $\ddot{\bar{\boldsymbol{q}}}_1$ that satisfies the first (highest priority) task is the same obtained with Algorithms 1 and 2, since $\bar{\boldsymbol{P}}_1 = \boldsymbol{P}_0 = \boldsymbol{I}$. When attacking the generic task k, the joint acceleration command is computed with the *SNS* projection equations

$$\begin{aligned} \ddot{\boldsymbol{q}}_{N,k} &= \left((\boldsymbol{I} - \boldsymbol{W}_k)\,\boldsymbol{P}_1\right)^{\#}\ddot{\boldsymbol{q}}_{N,k} \\ \ddot{\tilde{\boldsymbol{q}}}_k &= \ddot{\bar{\boldsymbol{q}}}_k + \ddot{\boldsymbol{q}}_{N,k} \\ \ddot{\bar{\boldsymbol{q}}}_k &= \ddot{\tilde{\boldsymbol{q}}}_k + \left(\boldsymbol{J}_k\bar{\boldsymbol{P}}_k\right)^{\#}\left(s_k\ddot{\boldsymbol{x}}_k - \dot{\boldsymbol{J}}_k\dot{\boldsymbol{q}} - \boldsymbol{J}_k\ddot{\tilde{\boldsymbol{q}}}_k\right). \end{aligned} \tag{9}$$

Similarly to the single task algorithm, we check first if the task can be executed within the joint acceleration bounds (7). If not, the task scaling factor and the most critical joint are computed using again Algorithm 2. When the obtained scaling factor is the largest computed so far, the current solution parameters (s_k, $\boldsymbol{W}_k$, $\ddot{\boldsymbol{q}}_{N,k}$, $\bar{\boldsymbol{P}}_k$) are saved. At this point, the most critical joint j is disabled for the execution of the current task ($W_{k,jj} = 0$), and the acceleration contribution of this saturated joint is assigned as null-space vector component $\ddot{q}_{N,k,j}$. The auxiliary null-space projector is then obtained using eq. (3)

$$\bar{\boldsymbol{P}}_k = \left(\boldsymbol{I} - \left((\boldsymbol{I} - \boldsymbol{W}_k)\,\boldsymbol{P}_{k-1}\right)^{\#}\right)\boldsymbol{P}_{k-1}, \tag{10}$$

by considering $\ddot{\boldsymbol{q}}_{N,k}$ as an auxiliary task (at the configuration space level), and thus with 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 acceleration command is provided as output by (9). Otherwise, the joint acceleration is recomputed with the current parameters and the process is repeated. Note that when (9) is used with saturated commands, the auxiliary null-space vector $\ddot{\bar{\boldsymbol{q}}}_{N,k}$ 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 $\ddot{\boldsymbol{q}}_{SNS} = \ddot{\bar{\boldsymbol{q}}}_l$ of the algorithm is obtained after processing the (last) task l.

Algorithm 3 (*SNS* at the acceleration level for multiple tasks)

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

Remark 1: Equation (9) collapses into the kth step of the prioritized multi-task motion control scheme (2), 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 3 is equivalent to eqs. (2–3). Moreover, matrix $\boldsymbol{W}_k$ is rebuilt independently of the obtained matrices $\boldsymbol{W}_i$ at steps $i < k$. As a result, even if the acceleration 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 acceleration away from its saturation level.

Remark 2: An expensive operation in the multi-task *SNS* algorithm is the computation of $\left(\left(\boldsymbol{I} - \boldsymbol{W}_k\right)\boldsymbol{P}_{k-1}\right)^{\#}$, which has to be done every time a new command saturates. Computational savings are obtained by considering the $r \times n$ matrix $\bar{\boldsymbol{W}}_k$ composed only by the rows of $\left(\boldsymbol{I} - \boldsymbol{W}_k\right)$ whose diagonal element is 1, being r the number of saturated joints. We have then

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

so that only the inversion of a nonsingular $r \times r$ sub-matrix of $\boldsymbol{P}_{k-1}$ is needed rather than pseudoinversion of the original rank-deficient $n \times n$ matrix.

VI. Configuration Space Tasks

As a special case, we analyze tasks that request to control robot behavior directly in the configuration space (CS). This is of interest when the redundant robot is commanded at the acceleration level, in order to damp otherwise uncontrolled self-motion velocities or to include a Projected Gradient optimization of auxiliary criteria (e.g., manipulability).

The multi-task *SNS* algorithm provides a computationally simple and effective solution that executes only that part of a CS task which preserves higher priority tasks without violating the constraints on the joint acceleration commands. At the acceleration level, a CS task of priority k is specified simply by a desired $\ddot{\boldsymbol{q}}_{CS}$. We have then

$$\ddot{\boldsymbol{x}}_k = \ddot{\boldsymbol{q}}_{CS} \quad \Rightarrow \quad \boldsymbol{J}_k = \boldsymbol{I}. \tag{11}$$

It is easy to check that the special structure (11) simplifies considerably some steps of the multi-task Algorithm 3 (we leave the details to the reader). We present here as **Algorithm 4** an even simpler version of the *SNS* algorithm for a CS task. This algorithm just replaces the kth loop of Algorithm 3 when task k is of the CS type.

Algorithm 4 (Simplified *SNS* for task k of the CS type)

$\boldsymbol{W}_{CS} = \boldsymbol{I}$
for $i = 1 \rightarrow n$ **do**
  if $\ddot{\bar{q}}_{k-1,i} = \ddot{Q}_{min,i}$ .OR. $\ddot{\bar{q}}_{k-1,i} = \ddot{Q}_{max,i}$ **then**
    $\boldsymbol{W}_{CS,ii} = 0$
  end if
end for
$\bar{\boldsymbol{P}}_{CS} = \left(\boldsymbol{I} - \left(\left(\boldsymbol{I} - \boldsymbol{W}_{CS}\right)\boldsymbol{P}_{k-1}\right)^{\#}\right)\boldsymbol{P}_{k-1}$
$\boldsymbol{a} = \bar{\boldsymbol{P}}_{CS}\ddot{\boldsymbol{q}}_{CS}$
$\boldsymbol{b} = \ddot{\bar{\boldsymbol{q}}}_{k-1}$
getTaskScalingFactor($\boldsymbol{a}$, $\boldsymbol{b}$) (∗**call Algorithm 2**∗)
$s_{CS} =$ {task scaling factor}
$\ddot{\bar{\boldsymbol{q}}}_k = \ddot{\bar{\boldsymbol{q}}}_{k-1} + s_{CS}\bar{\boldsymbol{P}}_{CS}\ddot{\boldsymbol{q}}_{CS}$

In this simplified version, we take into account that CS tasks are typically used at a low priority level, so as to shape the joint self-motion whenever still possible. Therefore, all

acceleration commands that are saturated up to task $k-1$ are disabled together for task k. By calling Algorithm 2 only once, the whole CS task is scaled by a common factor in order to fulfill the bounds (4).

We provide two examples of typical CS tasks. Suppose that one of the robot tasks is to maximize a configuration-dependent performance criterion $\boldsymbol{H}(\boldsymbol{q})$. For this, the Projected Gradient (PG) method [4] specifies a joint velocity along the gradient direction

$$\dot{\boldsymbol{q}} = \alpha \nabla_{\boldsymbol{q}} \boldsymbol{H}(\boldsymbol{q}), \tag{12}$$

with a scalar stepsize $\alpha > 0$, to be then projected in the null space of a suitable Jacobian (depending on the priority of this CS task). The associated CS task acceleration will be

$$\ddot{\boldsymbol{q}}_{CS} = \alpha \nabla^2_{\boldsymbol{q}} \boldsymbol{H}(\boldsymbol{q})\dot{\boldsymbol{q}} + \beta\left(\alpha \nabla_{\boldsymbol{q}} \boldsymbol{H}(\boldsymbol{q}) - \dot{\boldsymbol{q}}\right), \tag{13}$$

with a gain parameter $\beta > 0$. For stabilizing undesired self-motion velocities in any acceleration control scheme [18], the CS acceleration task is formulated instead as

$$\ddot{\boldsymbol{q}}_{CS} = -k_d \dot{\boldsymbol{q}}, \tag{14}$$

with a gain parameter $k_d > 0$. We note that the choice of design parameters in (13) and (14) is not critical. Thanks to the embedded task scaling feature of the *SNS* algorithm, they can be set at arbitrarily large values. The algorithm will automatically scale them to the largest values compatible with the robot joint constraints.

VII. Simulation Results

The method has been tested in simulation using a kinematic model of the KUKA LWR IV robot ($n = 7$). From the data sheet, all joint range limits are symmetric $\boldsymbol{Q}_{max} = (170, 120, 170, 120, 170, 120, 170)$ [deg] $= -\boldsymbol{Q}_{min}$ and the maximum joint velocities are $\boldsymbol{V}_{max} = (100, 110, 100, 130, 130, 180, 180)$ [deg/s]. Further, a maximum acceleration $\boldsymbol{A}_{max} = 300 \cdot \boldsymbol{I}$ [deg/s^2] has been chosen, equal for all joints. A sampling time $T = 1$ [ms] is used, also for shaping the joint acceleration constraints (7).

In the first simulation, only a primary task of dimension $m_1 = 3$ is specified. The robot end-effector position $\boldsymbol{x}_1 = \boldsymbol{f}_1(\boldsymbol{q})$ should cycle twice through a series of six Cartesian points connected by linear paths, starting from $\boldsymbol{q}(0) = (0, 45, 45, 45, 0, 0, 0)$ [deg]. The points are vertices of an hexagon inscribed in a circle lying in the (Y, Z) vertical plane, and having center in $\begin{pmatrix} 0.1 & 0.35 & 0.6235 \end{pmatrix}^T$ [m] and radius 0.2 [m]. At the time instant $t_h = hT$, the desired velocity $\dot{\boldsymbol{x}}_{1,h} = \dot{\boldsymbol{x}}_1(t_h)$ is chosen so as to head toward the next desired Cartesian point, say $\boldsymbol{x}_r$, with speed V_h, i.e.,

$$\begin{aligned} \dot{\boldsymbol{x}}_{1,h} &= V_h \frac{\boldsymbol{x}_r - \boldsymbol{f}_1(\boldsymbol{q}_h)}{\|\boldsymbol{x}_r - \boldsymbol{f}_1(\boldsymbol{q}_h)\|} \\ V_h &= k_P \|\boldsymbol{x}_r - \boldsymbol{f}_1(\boldsymbol{q}_h)\| - k_D \|\boldsymbol{J}_1(\boldsymbol{q}_{h-1})\dot{\boldsymbol{q}}_{h-1}\|, \end{aligned} \tag{15}$$

where $k_P = 10$, $k_D = 0.1$, and $\boldsymbol{J}_1(\boldsymbol{q})$ is the 3×7 Jacobian associated to the robot end-effector velocity. The desired acceleration $\ddot{\boldsymbol{x}}_{1,h} = \ddot{\boldsymbol{x}}_1(t_h)$ is then obtained by discrete time differentiation of (15) as

$$\ddot{\boldsymbol{x}}_{1,h} = \frac{\dot{\boldsymbol{x}}_{1,h} - \boldsymbol{J}_1(\boldsymbol{q}_{h-1})\dot{\boldsymbol{q}}_{h-1}}{T}. \tag{16}$$

Note that this task trajectory is particularly demanding at the vertices $\boldsymbol{x}_r$ (reached within a tolerance ε) of the hexagon, where large accelerations are required to change suddenly direction. As a measure of the directional error in executing the task, we use the angle between the desired and the obtained velocity direction, both normalized:

$$e_d = \arccos\left(\frac{\boldsymbol{x}_r - \boldsymbol{f}_1(\boldsymbol{q})}{\|\boldsymbol{x}_r - \boldsymbol{f}_1(\boldsymbol{q})\|} \cdot \frac{\boldsymbol{J}_1(\boldsymbol{q})\dot{\boldsymbol{q}}}{\|\boldsymbol{J}_1(\boldsymbol{q})\dot{\boldsymbol{q}}\|}\right). \tag{17}$$

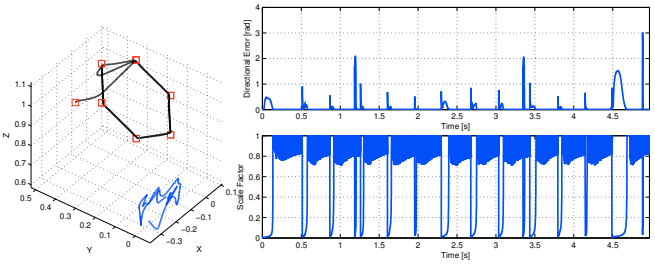

Fig. 1. Simulation 1. Single task performed using standard task scaling: [left] 3D plot of the end-effector (*black*) and elbow (*blue*) trajectory; [right] directional error (*top*) and task scaling factor (*bottom*)

Figure 1 shows the result of task execution at the acceleration level using pseudoinversion, see (1), and a standard task scaling method. In this case, only the limits $\pm\boldsymbol{A}_{max}$ on joint acceleration have been considered. The actual path executed by the end-effector is deformed, in particular at $t = 4.5$ s, as can be also evaluated on the error angle e_d. The narrow peeks in this directional error are due to the discontinuous change of direction when a desired point $\boldsymbol{x}_r$ is reached. Accordingly, each time the end-effector departs from a point $\boldsymbol{x}_r$ (including the starting point $\boldsymbol{f}_1(\boldsymbol{q}(0))$), the task acceleration needs to be scaled (see the right-bottom plot in Fig. 1). In addition, uncontrolled self-motion velocities in the null space of the task Jacobian result in a Cartesian drift of the robot elbow position (the blue trajectory in the 3D plot of Fig. 1).

In the second simulation, the same task is performed with the *SNS* algorithm under the full set of box constraints (4) on joint accelerations. From the results in Fig. 2 we can see that the desired task trajectory is much better reproduced. The narrow peeks on the directional error are still there, due to the discontinuities of the desired task velocity, but the error e_d is eliminated away from the hexagon vertices $\boldsymbol{x}_r$. On the other hand, when self-motion velocities are not damped, the elbow will drift and the robot will approach a bad manipulability configuration. Thus, higher joint acceleration will be needed to perform the task and this may violate the bounds (4). Such a situation occurs in fact just before $t = 2.5$ s, producing a large directional error even if the desired task direction is continuous at this time. By introducing self-motion velocity damping as a secondary task, i.e., as the CS task (14) with $k_d = 1000$ (as mentioned, this value can be chosen very

large), a considerable reduction in both elbow displacement and elbow joint velocity is obtained.

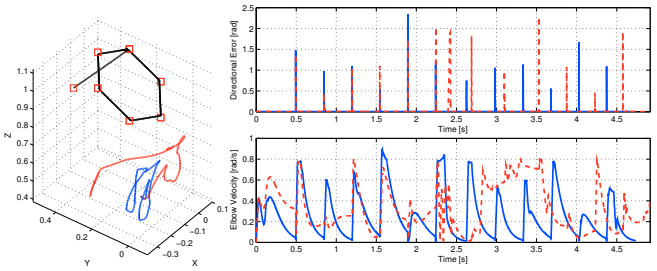

Fig. 2. Simulation 2. Single task performed using the *SNS* algorithm: [left] 3D plot of the end-effector (*black*) and of the elbow trajectory with (*blue*) and without (*red*) self-motion damping; [right] directional error (*top*) and elbow velocity (*bottom*) with (*solid blue*) and without (*dashed red*) self-motion damping

In the third and fourth simulations, we consider two Cartesian tasks with priority. Beside the same previous primary task for the robot end-effector, the secondary task requires to keep the robot elbow close to the (X, Z) vertical plane so as to reduce the workspace occupation by the robot arm while performing the primary task. This second requirement is formulated as a task on the position $x_2 = f_2(\boldsymbol{q})$ of the robot elbow along the (vertical) y-direction ($m_2 = 1$), with an associated 1×7 Jacobian $\boldsymbol{J}_2(\boldsymbol{q})$. At the time instant $t_h = hT$, the desired task velocity is

$$\dot{x}_{2,h} = -k_e f_2(\boldsymbol{q}_h), \tag{18}$$

with gain $k_e = 50$, and the associated desired acceleration for this second task is obtained as in (16):

$$\ddot{x}_{2,h} = \frac{\dot{x}_{2,h} - \boldsymbol{J}_2(\boldsymbol{q}_h)\dot{\boldsymbol{q}}_h}{T}. \tag{19}$$

Figure 3 shows the results obtained in the execution of the two tasks using the task priority scheme (2) and a standard task scaling method. Both tasks are badly performed, and in particular the primary task has been deteriorated by the presence of the secondary one (compare the directional error with that for the single task in Fig. 1). Figure 4 shows the execution of the two tasks with the *SNS* algorithm. In this case, the secondary task does not modify the execution of the primary task and is also correctly executed.

Table I summarizes quantitative measures of task execution in all performed simulations, using the *SNS* method (rows 2, 3, and 5) or not (rows 1 and 4). T_{tot} is the total time needed to go twice through the six desired Cartesian points. The other three columns provide *average values* of the directional error (quality of execution of the single or primary task), of the norm of the Cartesian velocity $\dot{\boldsymbol{x}}_{el}$ of the elbow (related to the presence of self-motion in the null space of the primary task in simulations S1 and S2), and of the absolute value of the y-position of the elbow (representing the quality of execution of the secondary task in S3 and S4).

VIII. Experimental Results

A number of experiments have been performed with a 7-dof KUKA LWR IV commanded at the acceleration level.

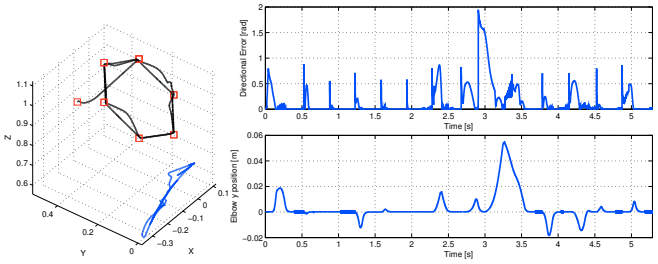

Fig. 3. Simulation 3. Two tasks with priority performed using standard task scaling: [left] 3D plot of the end-effector (*black*) and elbow (*blue*) trajectory; [right] directional error for the primary task (*top*) and secondary error on the elbow y-position x_2 (*bottom*)

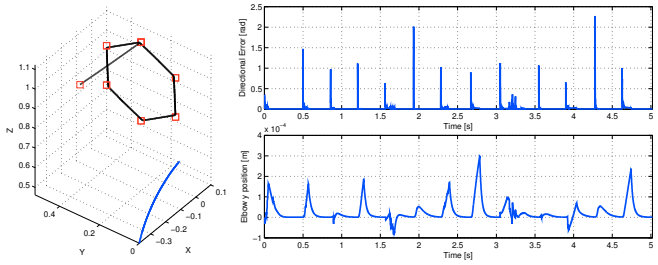

Fig. 4. Simulation 4. Two tasks with priority performed using the *SNS* algorithm: [left] 3D plot of the end-effector (*black*) and elbow (*blue*) trajectory; [right] directional error for the primary task (*top*) and secondary error on the elbow y-position x_2 (*bottom*)

TABLE I

Summary of the Simulation Results

	time T_{tot} [s]	average e_d [rad]	average $\|\dot{\boldsymbol{x}}_{el}\|$ [m/s]	average $\|x_2\|$ [m]
S1	4.972	0.0769	0.2764	0.0348
S2 (*SNS*)	4.917	0.0157	0.3287	0.1430
S2 (*SNS* + damp)	4.740	0.0066	0.2513	0.0445
S3	5.286	0.1376	0.3121	0.0045
S4 (*SNS* + elbow)	5.018	0.0138	8.8×10^{-4}	2.6×10^{-5}

The experiments are illustrated also by the accompanying video.

In the first experiment, the primary task requires to move the end-effector along linear paths with constant speed V, passing through 20 equidistant points distributed uniformly on a circle in the vertical (Y, Z) plane, having center at $\begin{pmatrix} -0.5 & 0 & 0.5 \end{pmatrix}^T$ [m] and radius 0.3 [m]. As secondary task, the robot should be attracted to a preferred configuration $\boldsymbol{q}_0$ (having good manipulability) using the Projected Gradient method. The performance criterion to be maximized is thus

$$\boldsymbol{H}(\boldsymbol{q}) = -\frac{1}{2}(\boldsymbol{q} - \boldsymbol{q}_0)^T(\boldsymbol{q} - \boldsymbol{q}_0),$$

and the desired CS task acceleration is given by eq. (13).

Figure 5 shows the results obtained using the *SNS* method, with $\boldsymbol{q}_0 = \begin{pmatrix} 0 & 1.0472 & 0 & 1.5708 & 0 & 0 & 0 \end{pmatrix}^T$ [rad], $\alpha\beta = 0.5$, $\alpha + \beta = 10$, and $V = 0.3$ [m/s]. After an initial approaching transient, the primary task is correctly executed. This can be seen also from the overlap of the desired and actual end-effector velocities, despite the several saturations

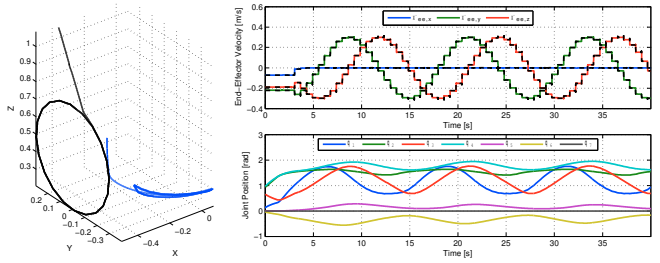

Fig. 5. Experiment 1. Two tasks with priority performed using the *SNS* algorithm: [left] 3D plot of the end-effector (*black*) and elbow trajectory (*blue*); [top right] end-effector desired (*dashed black*) and actual (*solid*) velocity; [bottom right] joint evolutions

of the acceleration command occuring during motion. The good attractive effect on joint configurations realized by the PG method as a secondary task can be appreciated in the plot at the bottom right of Fig. 5.

In the second experiment, the primary task is specified as before, but only through 3 desired end-effector positions $\boldsymbol{x}_r$. For this trajectory, joint limits are reached when using a simple pseudoinverse acceleration control, as well as when adding a configuration-attracting PG scheme in the null space of the primary task. Figures 6 and 7 show the good results obtained instead with the same control method and parameters used in the first experiment. As in the simulations, the narrow peeks (now smaller) in the directional error are caused by the discontinuity of the desired velocity when moving out of each $\boldsymbol{x}_r$ point. The joint positions remains within their ranges, despite joint 4 reaches several times its upper limit (as shown also in the fourth frame of Fig. 7).

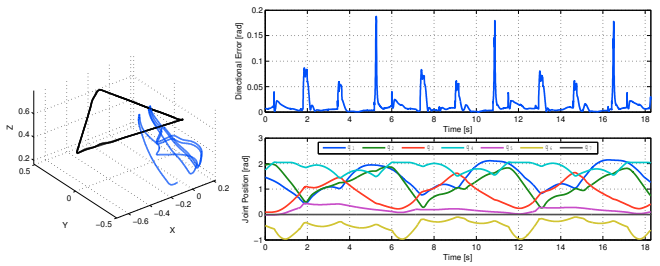

Fig. 6. Experiment 2. Two tasks with priority performed using the *SNS* algorithm: [left] 3D plot of the end-effector (*black*) and elbow trajectory (*blue*); [right] directional error (*top*) and joint evolutions (*bottom*)

Fig. 7. Snapshots from Experiment 2 with the KUKA LWR IV

IX. Conclusions

We have extended our single-task *SNS* algorithm [14] for motion control of redundant robots under hard constraints on joint variables/commands to the case of multiple prioritized tasks. The basic idea is an efficient search in the task Jacobian(s) null space(s), obtained by saturating one at the time the acceleration commands, compensating their effect in the null space, and possibly introducing task scaling when strictly needed. In the multi-task case, the *SNS* algorithm realizes conveniently a preemptive prioritization strategy, letting first the higher priority tasks use all the robot capabilities they need at the joint level. Finally, our method can be used with simple modifications also at the generalized force control level within the operational space framework [2].

References

[1] 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.

[2] O. Khatib, "The operational space framework," *JSME Int. J. Ser. C: Dynamics, Control, Robotics, Design and Manufacturing*, vol. 36, no. 3, pp. 277–287, 1993.

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

[4] 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.

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

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

[7] 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.

[8] P. Chiacchio and S. Chiaverini, "Coping with joint velocity limits in first-order inverse kinematics algorithms: Analysis and real-time implementation," *Int. J. of Robotics Research*, vol. 13, no. 5, pp. 515–519, 1995.

[9] 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*, vol. 19, no. 1, pp. 162–167, 2003.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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

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

[16] 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.

[17] 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.

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