2012 IEEE International Conference on Robotics and Automation
RiverCentre, Saint Paul, Minnesota, USA
May 14-18, 2012

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

Fabrizio Flacco* Alessandro De Luca* Oussama Khatib**

Abstract— **We present a novel efficient method addressing the inverse differential kinematics problem for redundant manipulators in the presence of different hard bounds (joint range, velocity, and acceleration limits) on the joint space motion. The proposed *SNS* (Saturation in the Null Space) iterative algorithm proceeds by successively discarding the use of joints that would exceed their motion bounds when using the minimum norm solution and reintroducing them at a saturated level by means of a projection in a suitable null space. The method is first defined at the velocity level and then moved to the acceleration level, so as to avoid joint velocity discontinuities due to the switching of saturated joints. Moreover, the algorithm includes an optimal task scaling in case the desired task trajectory is unfeasible under the given joint bounds. We also propose the integration of obstacle avoidance in the Cartesian space by properly modifying on line the joint bounds. Simulation and experimental results reported for the 7-dof lightweight KUKA LWR IV robot illustrate the properties and computational efficiency of the method.**

I. Introduction

Inversion of first-order (velocity level) or second-order (acceleration level) differential kinematics is the standard way to execute a desired Cartesian motion of redundant robots [1]. Control methods typically use a generalized inverse (most often, the pseudoinverse) of the manipulator Jacobian in order to convert velocity or acceleration commands from the task space to the joint space, where actuation takes place. Kinematic redundancy is exploited for collision avoidance, for joint motion optimization, or by augmenting the primary task with multiple additional ones, possibly prioritized [2]. These approaches can be casted as the selection of suitable joint commands in the null space of the Jacobian matrix.

An important problem of all the above methods in their simplest form is that constraints in the joint space (e.g., bounds on the joint range, velocity, acceleration, or even torque) are not taken explicitly into account. 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. 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. Simple scaling of the task commands recovers motion feasibility but may no longer satisfy the primary intention, e.g., avoid a collision with a fast moving human. Before doing so, it would be useful to verify whether there exist alternative joint motions satisfying the hard bounds in the joint space and still executing the original task command. Stated in this way, the problem is intrinsically local to the current robot state, i.e., needs to be solved on line with no future information.

A common on-line approach to deal with limited joint ranges is to convert these hard bounds into soft ones, resolving redundancy by optimization (e.g., based on the Projected Gradient algorithm) of an objective function that keeps the joints closer to the center of their ranges (see [3], [4]). However, since the Cartesian task has always the highest priority, satisfaction of joint limits is not guaranteed. A method that enhances the capability of avoiding joint limits by using a suitably weighted pseudoinverse has been proposed in [5]. Other similar solutions for handling the presence of joint ranges have been proposed in the context of visual servoing tasks (e.g., [6], [7]).

In the presence of joint velocity or joint acceleration bounds, simple software saturation of the joint-space command results in the lack of execution of the desired motion in the task space. In [8], a redundancy resolution method was proposed that minimizes the infinity-norm (i.e., the maximum absolute value of the components) of the command vector in the joint space, with the norm being weighted by the available actuation ranges. Nonetheless, satisfaction of the original hard bounds is again not guaranteed. In [9], the velocity term in the one-dimensional null space of a 7-dof robot is scaled so as to satisfy the joint velocity bounds, if at all possible. On the other hand, task relaxation by time scaling can be used for satisfying joint velocity [10] and/or acceleration [11] bounds. This approach has been extended to the redundant case, specifically in [12] to a team of mobile robots executing multiple tasks with priority. Lower-priority task velocity commands are scaled in the null space of higher-priority Jacobians so that the hierarchy of task priorities is preserved despite actuator saturations.

It should be mentioned that the considered problem fits into the framework of constrained minimization of a quadratic objective function under linear equality and inequality constraints, possibly having different priorities. Once the problem is properly formulated, the use of a general-purpose optimization algorithm is suggested in [13].

*Dipartimento di Ingegneria Informatica, Automatica e Gestionale, 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).

However, since the inequality constraints are here only in the form of elementary bounds (box constraints) on the commands at the joint level, the problem structure could be further exploited so as to lead to a more efficient solution.

A method that explicitly handles joint velocity or acceleration bounds in a n-dof redundant robot performing an m-dimensional task (with $m < n$) has been introduced in [14]. The saturation of the *whole set of* $s \leq (n-m)$ *joints that exceed their bounds* (called *overdriven* by the authors) in the unconstrained solution is simultaneously compensated by using the remaining joints, which still have motion capabilities. The method works by selecting columns from the Jacobian null-space projection matrix and proceeds by pseudoinversion of the resulting non-square $s \times n$ matrix. The associated redistribution of joint motion may however lead to further saturations, in which case computations need to be repeated.

In this paper, we propose an improved method, leading to the *SNS* algorithm (Saturation in the Null Space), based on a concept similar to [14]. However, the *SNS* algorithm is computationally more efficient, integrates the minimum task scaling (including no scaling) needed to accomplish the Cartesian task with the given constraints, and considers in a unified framework all joint motion constraints (joint range limits, velocity and acceleration bounds). The basic idea is to disable only *one joint at a time* (the most critical one, according to some criterion) out of the set for which the requested motion would exceed the capabilities, and to reintroduce the saturated contribution of this joint in the null space of a suitable Jacobian matrix. When designed at the velocity level, the final outcome is a *feasible* joint command that can be put in the standard form

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{\#}(\boldsymbol{q})\, s\dot{\boldsymbol{x}} + \Big(\boldsymbol{I} - \boldsymbol{J}^{\#}(\boldsymbol{q})\boldsymbol{J}(\boldsymbol{q})\Big)\,\dot{\boldsymbol{q}}_N, \qquad (1)$$

with the pseudoinverse of the task Jacobian, and both a task scaling factor $s \in (0,1]$ and a null-space velocity vector $\dot{\boldsymbol{q}}_N$ that are provided by the *SNS* algorithm. The combination of the two final choices s and $\dot{\boldsymbol{q}}_N$ guarantees the satisfaction of all joint constraints. If possible, s is kept to 1 (full task preservation), otherwise it is reduced as least as possible to recover feasibility. Moreover, at each iterative step of our method, pseudoinversion of only of a $m \times (n-s)$ submatrix of the original Jacobian is required. Moreover, this is a rank one modification of the pseudoinverse computed at the previous step, so that the updated matrix is efficiently obtained without a new full SVD operation. When compared to [14], the proposed method leads in general to a reduced number of saturated joint commands. This is a relevant property, since the more are the saturation events, the less smooth is the resulting robot motion.

The paper is organized as follows. The handling of saturation of the joint commands by our method is illustrated through a simple motivating example in Sect. II. In Sec. III the *SNS* algorithm is proposed and analyzed at the level of velocity commands. The extension of the method at the acceleration level is presented in Sec. IV. Furthermore, in Sec. V we show how to include the additional presence of Cartesian constraints, e.g., coming from obstacle avoidance requirements, with a suitable local modification of the joint bounds. The effectiveness of the various versions of the *SNS* algorithm is shown by Matlab$^{\text{TM}}$ simulations and by experiments on a 7-dof KUKA LWR IV robot (Sect. VI).

II. Illustrative Example

Consider a planar 4R manipulator with equal links of unitary length performing a task specified by a desired end-effector linear velocity $\dot{\boldsymbol{x}} \in \mathbb{R}^2$ (i.e., $m = 2$) and commanded by the joint velocity $\dot{\boldsymbol{q}} \in \mathbb{R}^4$ (i.e., $n = 4$). The degree of robot redundancy for this task is $n - m = 2$. Suppose that the joint velocities are bounded as $|\dot{q}_i| \leq V_i$, $i = 1, \ldots, 4$, with $V_1 = V_2 = 2$, $V_3 = V_4 = 4$ [rad/s].

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

$$\boldsymbol{J} = \begin{pmatrix} J_1 & J_2 & J_3 & J_4 \end{pmatrix} = \begin{pmatrix} -2 & -1 & -1 & 0 \\ 2 & 2 & 1 & 1 \end{pmatrix}. \qquad (2)$$

For a desired task velocity $\dot{\boldsymbol{x}} = \begin{pmatrix} -4 & -1.5 \end{pmatrix}^T$, the minimum norm joint velocity solution is

$$\dot{\boldsymbol{q}}_{PS} = \boldsymbol{J}^{\#}\dot{\boldsymbol{x}} = \begin{pmatrix} 2.4545 & -2.1364 & 1.2273 & -3.3636 \end{pmatrix}^T \qquad (3)$$

which exceeds the bounds on the first and second joint. Note that the ratio $|\dot{q}_i|/V_i$ is larger for joint $i = 1$.

A natural solution that tries to preserve the original task velocity would bring back the exceeding joint values at their closest saturation level, i.e., $\dot{q}_1 = V_1 = 2$, $\dot{q}_2 = -V_2 = -2$, and redefine the remaining joint velocities so as to still satisfy the task, if possible. From

$$\dot{\boldsymbol{x}}' = \dot{\boldsymbol{x}} - J_1 V_1 + J_2 V_2 = \begin{pmatrix} J_3 & J_4 \end{pmatrix} \begin{pmatrix} \dot{q}_3 \\ \dot{q}_4 \end{pmatrix}, \qquad (4)$$

we see that a solution can be found since the square Jacobian sub-matrix $\begin{pmatrix} J_3 & J_4 \end{pmatrix}$ is nonsingular. We obtain

$$\begin{aligned} \dot{\boldsymbol{q}}' &= \Big(V_1 \quad -V_2 \quad \begin{pmatrix} J_3 & J_4 \end{pmatrix}^{-1} \dot{\boldsymbol{x}}' \Big) \\ &= \begin{pmatrix} 2 & -2 & 2 & -3.5 \end{pmatrix}^T, \end{aligned} \qquad (5)$$

which satisfies in this case the bounds on all joints. This feasible solution, which is also the outcome of the algorithm presented in [14], has norm $\|\dot{\boldsymbol{q}}'\| = 4.9244$.

However, a better solution can be found by applying our *SNS* algorithm, as described in full in Sect. III. We saturate only the most violating velocity, in this case $\dot{q}_1 = V_1 = 2$, and redefine the remaining three joint velocities from

$$\dot{\boldsymbol{x}}_{SNS} = \dot{\boldsymbol{x}} - J_1 V_1 = \begin{pmatrix} J_2 & J_3 & J_4 \end{pmatrix} \begin{pmatrix} \dot{q}_2 \\ \dot{q}_3 \\ \dot{q}_4 \end{pmatrix}$$

obtaining

$$\begin{aligned} \dot{\boldsymbol{q}}_{SNS} &= \Big(V_1 \quad \begin{pmatrix} J_2 & J_3 & J_4 \end{pmatrix}^{\#} \dot{\boldsymbol{x}}'_{SNS} \Big) \\ &= \begin{pmatrix} 2 & -1.8333 & 1.8333 & -3.6667 \end{pmatrix}^T \end{aligned} \qquad (6)$$

by pseudoinversion. As before, all bounds are satisfied by this alternative feasible solution, which uses less saturated joints and has also a norm $\|\dot{\boldsymbol{q}}'_{SNS}\| = 4.9160$ that is smaller than in the previous case —perhaps not surprisingly.

Suppose now that the bound on the second joint velocity is reduced to $V_2 = 1$ [rad/s], with all other operative conditions remaining the same. Saturating the two overdriven joints 1 and 2 of the pseudoinverse solution (3) and solving again as in (4) provides in this case

$$\dot{\boldsymbol{q}}' = \begin{pmatrix} 2 & -1 & 1 & -4.5 \end{pmatrix}^T, \tag{7}$$

which has now the fourth velocity out of bound. The algorithm in [14] ends at this stage, since saturating also joint 4 would leave a single velocity (of joint 3) available for satisfying the two-dimensional task, which is clearly insufficient. Note that the same result (7) is obtained also when saturating one of the overdriven joints at the time (in any priority order) and then repeating the procedure as needed (i.e., twice). As a result, a task scaling procedure is certainly needed to recover feasibility of the joint velocity commands under the reduced joint velocity bound.

If we scale the original task velocity so as to recover a feasible joint velocity (with at least one saturated component) using just Jacobian pseudoinversion as in (3), we compute a downscaling factor as small as $s = 0.4681$ and need to reduce $\dot{\boldsymbol{x}}$ to $s\dot{\boldsymbol{x}}$. Accordingly, we obtain from (3)

$$\dot{\boldsymbol{q}}_s = s\,\dot{\boldsymbol{q}} = \begin{pmatrix} 1.1489 & -1 & 0.5745 & -1.5745 \end{pmatrix}^T.$$

On the other hand, if we scale the outcome of the saturated joint velocity (7), a larger task scaling value $s' = 0.8889$ is found and the solution is

$$\dot{\boldsymbol{q}}'_s = s'\dot{\boldsymbol{q}}' = \begin{pmatrix} 1.7778 & -0.8889 & 0.8889 & -4 \end{pmatrix}^T,$$

where again only one joint velocity component is left at its saturated level (the fourth one in this case). Instead, by using our *SNS* algorithm, we obtain the largest possible reducing factor $s_{SNS} = 0.9091$ (with less than a 10% reduction of the original task speed $\dot{\boldsymbol{x}}$) and the globally optimal solution

$$\dot{\boldsymbol{q}}_{SNS,s} = \begin{pmatrix} 2 & -1 & 0.6364 & -4 \end{pmatrix}^T.$$

Note that this joint velocity command has three saturated values and can be rewritten in the form (1) with $s = s_{SNS}$ and $\dot{\boldsymbol{q}}_N = \begin{pmatrix} -0.4913 & 0.8537 & -0.6093 & -0.9007 \end{pmatrix}^T$.

III. Velocity-level control

We introduce here our *SNS* method at the velocity level. We show first how to shape the joint velocity bounds so as to take into account also joint range limits and acceleration bounds (Sect. III-A). Then we present the general *SNS* algorithm, including task scaling, at the velocity level (Sect. III-B), followed by simulation results on the 7-dof KUKA LWR IV robot (Sect. . A discussion of the properties of the method and on computational issues concludes this section (Sect. III-D).

A. Shaping the joint velocity bounds

We assume that the following bounds exists on the joint ranges, joint velocities, and joint accelerations:

$$\begin{aligned} Q_{min,i} &\le q_i \le Q_{max,i} \\ -V_{max,i} &\le \dot{q}_i \le V_{max,i} \qquad i = 1,\dots,n. \\ -A_{max,i} &\le \ddot{q}_i \le A_{max,i} \end{aligned} \tag{8}$$

Note that the joint ranges need not to be symmetric, while the differential bounds typically are. By $\boldsymbol{Q}_{min}$, $\boldsymbol{Q}_{max}$, $\boldsymbol{V}_{max}$, and $\boldsymbol{A}_{max}$ we denote the vectors containing the above scalar limits. In the control implementation, the joint velocity command $\dot{\boldsymbol{q}}$ is kept constant at the computed value $\dot{\boldsymbol{q}}_k = \dot{\boldsymbol{q}}(t_k)$ for a sampling time of duration T, where $t_k = kT$. Suppose that at $t = t_k$ the current joint position $\boldsymbol{q} = \boldsymbol{q}_k$ is feasible.The next position $\boldsymbol{q}_{k+1} \simeq \boldsymbol{q}_k + \dot{\boldsymbol{q}}_k T$ needs still to be within the joint range limits, and thus

$$\frac{\boldsymbol{Q}_{min} - \boldsymbol{q}_k}{T} \le \dot{\boldsymbol{q}}_k \le \frac{\boldsymbol{Q}_{max} - \boldsymbol{q}_k}{T}. \tag{9}$$

The acceleration bounds in (8) can be similarly transferred to limits for $\dot{\boldsymbol{q}}_k$, by approximating $\dot{\boldsymbol{q}}_k \simeq \dot{\boldsymbol{q}}_{k-1} + \ddot{\boldsymbol{q}}_k T$, leading to

$$-\boldsymbol{A}_{max}T + \dot{\boldsymbol{q}}_{k-1} \le \dot{\boldsymbol{q}}_k \le \boldsymbol{A}_{max}T + \dot{\boldsymbol{q}}_{k-1}.$$

However, the term on the left-hand side in this chain of inequalities can be either positive or negative, leading to a more complex design of a task scaling scheme when this is needed (in fact, a sufficiently reduced task velocity would not automatically guarantee the satisfaction of all constraints in this case). Therefore, we prefer to resort to a different idea for including acceleration bounds at the velocity level. Suppose that we would need to stop the robot motion in the fastest possible way, namely by maximally decelerating a joint i which is moving at $\dot{q}_i > 0$ (or maximally accelerating it if $\dot{q}_i < 0$) so as to stay within the available joint range. For a generic $t \ge t_k$, we have for the ith joint position and velocity subject to $-A_{max,i}$ (the following reasoning is specular for the case of maximum acceleration)

$$\begin{aligned} q_i(t) &= q_{k,i} + \dot{q}_{k,i}(t - t_k) - \frac{A_{max.i}}{2}(t - t_k)^2 \\ \dot{q}_i(t) &= \dot{q}_{k,i} - A_{max.i}(t - t_k). \end{aligned}$$

The most critical situation is when we reach the upper limit $Q_{max,i}$ at some $t = t_i^* > t_k$ with the joint velocity being $\dot{q}_i(t_i^*) = 0$ —the joint stops right at the boundary of its range. It is then easy to check that the largest positive velocity of joint i that we can tolerate at $t = t_k$ is upper bounded as

$$\dot{q}_{k,i} \le \sqrt{2A_{max,i}\left(Q_{max,i} - q_{k,i}\right)}, \tag{10}$$

and similarly for the largest negative velocity which is lower bounded as

$$-\sqrt{2A_{max,i}\left(q_{k,i} - Q_{min,i}\right)} \le \dot{q}_{k,i}. \tag{11}$$

Putting together all the constraints given by the second set of inequalities in (8), by (9), as well as by (10) and (11) for

$i = 1, \dots, n$, we finally obtain the following box constraint for the command $\dot{\boldsymbol{q}}$ at time $t = t_k$

$$\dot{\boldsymbol{Q}}_{min}(t_k) \leq \dot{\boldsymbol{q}} \leq \dot{\boldsymbol{Q}}_{max}(t_k) \qquad (12)$$

where, for $i = 1, \dots, n$,

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

is negative and

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

is positive.

B. The SNS algorithm at the velocity level

Consider a manipulator with n joints performing a m-dimensional desired velocity task $\dot{\boldsymbol{x}}$, with $m < n$. At a given $\boldsymbol{q}$, the *SNS* algorithm for realizing the task at the velocity level under the box constraints (12) is presented below in pseudocode form.

Algorithm 1 (*SNS* at the velocity level)

```
$\boldsymbol{W} = \boldsymbol{I}$, $\dot{\boldsymbol{q}}_N = \boldsymbol{0}$, $s = 1$, $s^* = 0$
repeat
  limit_exceeded = FALSE
  $\dot{\boldsymbol{q}}_{SNS} = \dot{\boldsymbol{q}}_N + (\boldsymbol{J}\boldsymbol{W})^{\#} (s\dot{\boldsymbol{x}} - \boldsymbol{J}\dot{\boldsymbol{q}}_N)$
  if $\exists\ i \in [1\!:\!n] : \dot{q}_i < \dot{Q}_{min,i}$ .OR. $\dot{q}_i > \dot{Q}_{max,i}$ then
    limit_exceeded = TRUE

    call Algorithm 2
    if {task scaling factor} $> s^*$ then
      $s^*$ = {task scaling factor}
      $\boldsymbol{W}^* = \boldsymbol{W}$, $\dot{\boldsymbol{q}}_N^* = \dot{\boldsymbol{q}}_N$
    end if

    $j$ = {the most critical joint}
    $W_{jj} = 0$
    $\dot{q}_{N,j} = \begin{cases} \dot{Q}_{max,j} & \text{if } \dot{q}_j > \dot{Q}_{max,j} \\ \dot{Q}_{min,j} & \text{if } \dot{q}_j < Q_{min,j} \end{cases}$
    if rank($\boldsymbol{J}\boldsymbol{W}$) $< m$ then
      $s = s^*$, $\boldsymbol{W} = \boldsymbol{W}^*$, $\dot{\boldsymbol{q}}_N = \dot{\boldsymbol{q}}_N^*$
      $\dot{\boldsymbol{q}}_{SNS} = \dot{\boldsymbol{q}}_N + (\boldsymbol{J}\boldsymbol{W})^{\#} (s\dot{\boldsymbol{x}} - \boldsymbol{J}\dot{\boldsymbol{q}}_N)$
      limit_exceeded = FALSE    (*outputs solution*)
    end if

  end if
until limit_exceeded = TRUE
```

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

$$\dot{\boldsymbol{q}}_{SNS} = \dot{\boldsymbol{q}}_N + (\boldsymbol{J}\boldsymbol{W})^{\#} (s\dot{\boldsymbol{x}} - \boldsymbol{J}\dot{\boldsymbol{q}}_N). \qquad (13)$$

Note that (13) collapses to the usual pseudoinverse (minimum norm) solution $\boldsymbol{J}^{\#}\dot{\boldsymbol{x}}$ with the given algorithm initialization.

We check first if the original ($s = 1$) task can be performed with the joint velocity (13) being within the box constraints (12). If not, Algorithm 2 is called to evaluate the minimum positive task scaling factor, say $\overline{s}$, *only* among the enabled joints (i.e., those corresponding to $W_{ii} = 1$). If $\overline{s}$ is larger than the current factor s^*, we store current $\boldsymbol{W}^* = \boldsymbol{W}$ and $\dot{\boldsymbol{q}}_N^* = \dot{\boldsymbol{q}}_N$ and set $s^* = \overline{s}$. These values provide the closest task velocity that can be executed so far along the original desired direction. If the jth joint is the most critical for task execution, i.e., its velocity needs the smallest task scaling to stay within the bounds, we saturate the velocity of joint j by setting $W_{jj} = 0$. If the task cannot be executed with this new parameters, If now the rank of $\boldsymbol{J}\boldsymbol{W}$ is strictly less than m, the algorithm stops with the best parameters that have been found ($\boldsymbol{W} = \boldsymbol{W}^*$, $\dot{\boldsymbol{q}}_N = \dot{\boldsymbol{q}}_N^*$, and $s = s^*$), and the joint velocity command is provided as output by (13). Otherwise, the joint velocity is recomputed with the current parameters using again (13) and the process is repeated.

An important aspect in the proposed method is the integrated use of a task scaling factor, as given by Algorithm 2 presented in pseudocode form below.

Algorithm 2 (Task scaling factor at the velocity level)

```
$\boldsymbol{a} = (\boldsymbol{J}\boldsymbol{W})^{\#}\, \dot{\boldsymbol{x}}$
$\boldsymbol{b} = \dot{\boldsymbol{q}}_N - (\boldsymbol{J}\boldsymbol{W})^{\#}\, \boldsymbol{J}\dot{\boldsymbol{q}}_N$
for $i = 1 \to n$ do
  $S_{min,i} = \left(\dot{Q}_{min,i} - b_i\right) / a_i$
  $S_{max,i} = \left(\dot{Q}_{max,i} - b_i\right) / a_i$
  if $S_{min,i} > S_{max,i}$ then
    {switch $S_{min,i}$ and $S_{max,i}$}
  end if
end for
$s_{max} = \min_i \{S_{max,i}\}$
$s_{min} = \max_i \{S_{min,i}\}$
if $s_{min} > s_{max}$ .OR. $s_{max} < 0$ .OR. $s_{min} > 1$ then
  task scaling factor = 0
else
  task scaling factor = $\min\{s_{max}, 1\}$
end if
```

As illustrated by the simple example in Sect. II, there is a basic difference between task velocity and joint velocity scaling. In the standard case they are equivalent, but this correspondence is no longer valid in the presence of the saturated velocity components in $\dot{\boldsymbol{q}}_N$. As a matter of fact, in the *SNS* algorithm the projection of $\dot{\boldsymbol{q}}_N$ in a suitable null space allows a greater task scaling factor (possibly 1).

C. Simulation results

The method has been tested in simulation using a kinematic model of the KUKA LWR IV robot. From the data sheet, joint range limits are all symmetric ($Q_{min,i} =$

$-Q_{max,i}$) and $\boldsymbol{Q}_{max} = (170, 120, 170, 120, 170, 120, 170)$ [deg]. The maximum joint velocities are $\boldsymbol{V}_{max} = (100, 110, 100, 130, 130, 180, 180)$ [deg/s]. To shape the joint velocity bounds, we have considered a maximum acceleration of 300 [deg/s^2] for all joints[1], with a simulation sampling time $T = 1$ [ms].

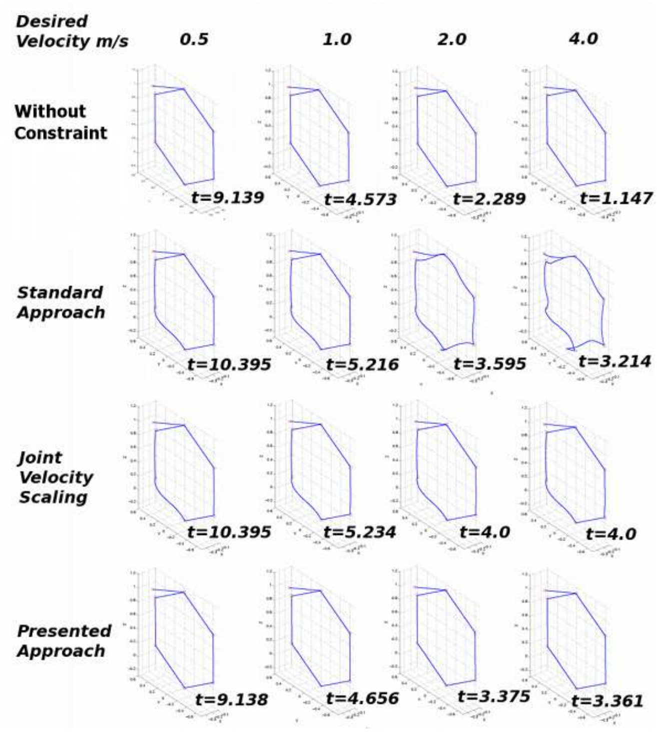

Fig. 1. Execution of a multi-point task at constant Cartesian speed with the KUKA LWR IV. The results refer to the ideal case of $\dot{\boldsymbol{q}} = \boldsymbol{J}^{\#}(\boldsymbol{q})\dot{\boldsymbol{x}}$ without joint constraints (first row) and in the presence of constraints (second row), possibly followed by joint velocity scaling (third row), compared to the *SNS* algorithm (fourth row). Each simulation test is repeated at the four desired task speeds $V = 0.5$, 1.0, 2.0, and 4.0 [m/s]. Task completion times are also displayed for each realization

We present the case of a task specified by moving the robot end-effector through six desired Cartesian positions with linear paths at constant speed V. For the generic next position $\boldsymbol{x}_r$, the task velocity is given by

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

The robot starts at $\boldsymbol{q}(0) = (0, 45, 45, 45, 0, 0, 0)$ [deg]. Figure 1 shows the results obtained with four different redundancy resolution methods at four different constant speeds. The first row represents the ideal case, using Jacobian pseudoinversion and discarding the joint constraints. In the second row, the simulation includes all joint constraints but the control law does not consider them (standard approach). The third control row is again the pseudoinverse solution, followed by task velocity scaling to recover feasibility (this is equivalent to joint scaling here). Finally, the results obtained with the proposed *SNS* algorithm are shown in the last row. The task completion times are displayed for each realization. It can be noticed that the standard approach is generally slower than *SNS* for a low task speed. When the task speed increases, the completion time is slightly faster, but at the expense of a highly deformed trajectory due to joint velocity saturations, as opposed to the *SNS* case. On the other hand, task velocity scaling to recover feasibility with a pseudoinverse solution preserves the desired task path but the completion time is longer than with our method.

[1]In the simulations at velocity level we do not consider explicitly bounds on the joint acceleration.

D. Properties and computational issues

We provide some comments on the structure of the projection equation (13) which leads to further properties of the *SNS* method. 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 and (13) will not update the joint velocity $\dot{q}_i$ after Algorithm 1 has saturated its value. At a generic iteration of the algorithm, let n_s be the number of velocity commands that are in saturation (or disabled) while the remaining $n_e = n - n_s \geq m$ are enabled (and yet to be defined). Possibly 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. The vector $\dot{\boldsymbol{q}}_s \in \mathbb{R}^{n_s}$ contains the saturated joint velocities. Thus, we have

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

and equation (13) can be re-expressed as

$$\dot{\boldsymbol{q}}_{SNS} = \begin{pmatrix} \boldsymbol{0} \\ \dot{\boldsymbol{q}}_s \end{pmatrix} + \begin{pmatrix} \boldsymbol{J}_e^{\#} \\ \boldsymbol{O} \end{pmatrix} (s\dot{\boldsymbol{x}} - \boldsymbol{J}_s\dot{\boldsymbol{q}}_s). \tag{14}$$

It is easy to show that the computed $\dot{\boldsymbol{q}}_e$ in (14) 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

$$\min \frac{1}{2}\dot{\boldsymbol{q}}_e^T \dot{\boldsymbol{q}}_e, \quad \text{s.t.} \quad \boldsymbol{J}_e\dot{\boldsymbol{q}}_e = s\dot{\boldsymbol{x}} - \boldsymbol{J}_s\dot{\boldsymbol{q}}_s, \tag{15}$$

for a given set of saturated joints $\dot{\boldsymbol{q}}_s$. Indeed, we can also rewrite (13) in a form containing explicitly a projection matrix, and then proceed as above:

$$\begin{aligned} \dot{\boldsymbol{q}}_{SNS} &= (\boldsymbol{JW})^{\#} s\dot{\boldsymbol{x}} + \left(\boldsymbol{I} - (\boldsymbol{JW})^{\#}\boldsymbol{J}\right)\dot{\boldsymbol{q}}_N & (16) \\ &= \begin{pmatrix} \boldsymbol{J}_e^{\#} \\ \boldsymbol{O} \end{pmatrix} s\dot{\boldsymbol{x}} + \begin{pmatrix} -\boldsymbol{J}_e^{\#}\boldsymbol{J}_s \\ \boldsymbol{I}_{n_s} \end{pmatrix} \dot{\boldsymbol{q}}_s. & (17) \end{aligned}$$

One possible computational drawback of the presented approach, as opposed to [14], is the need to recompute $(\boldsymbol{JW})^{\#}$ (actually $\boldsymbol{J}_e^{\#}$, as just seen) each time a new joint is saturated (and $\boldsymbol{W}$ is modified) during the iterations of the *SNS* algorithm. However, since only one element at a time is modified in the diagonal of $\boldsymbol{W}$ and thus only one column is zeroed in the next $\boldsymbol{JW}$, the new pseudoinverse $(\boldsymbol{JW})^{\#}$

can be obtained as a rank-one variation of the previous matrix. Simple formulas are given in [15] to compute the new pseudoinverse without an additional SVD operation.

A more critical problem is the presence of discontinuities in the obtained joint velocity command, even if the desired task trajectory has no discontinuities in space and time. This behavior is generated in the *SNS* algorithm by the enable/disable switching of joints in the presence of task scaling. Consider for instance a linear point-to-point motion for the KUKA LWR IV robot between the initial point $\boldsymbol{x}_0 = (-0.3712, 0.3015, 1.1235)$ [m] and the final point $\boldsymbol{x}_1 = (-0.7, -0.15, 0.2)$ [m], with a constant desired speed $V = 0.1$ [m/s]. The only discontinuity of the desired task velocity occurs at $t = 0$. The joint velocity $\dot{\boldsymbol{q}}$ obtained with the *SNS* algorithm at the velocity level is shown in Fig. 2. Beside the expected discontinuity at $t = 0$, the resulting joint velocity commands have also an undesired jump at $t = 0.1$. This discontinuity at the joint velocity level would not be feasible for the real robot, due to its limited acceleration capabilities, and would result in a task trajectory error.

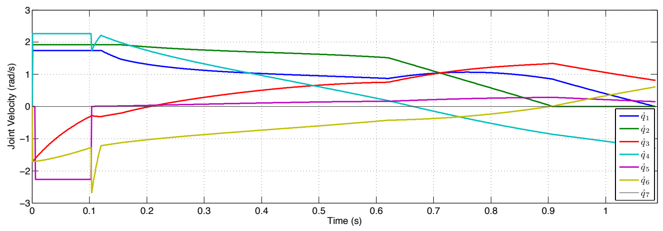

Fig. 2. Joint velocity $\dot{\boldsymbol{q}}$ of the KUKA LWR IV robot obtained with the *SNS* algorithm for a point-to-point task

IV. Acceleration-Level Control

Discontinuities in the joint velocity command are caused by the switching of saturated joints possibly required by the algorithm during robot motion. Even though this problem can be reduced by suitably choosing the order in which the joints are disabled, the natural solution is to move the *SNS* algorithm at the acceleration level so as to yield continuous joint velocity profiles. Moreover, at the second-order level the actual joint acceleration bounds are directly considered and the norm of the acceleration of the enabled joints will be minimized. As a result, a smoother joint motion is obtained which is also closer to the real dynamic characteristics of the robot.

A. Shaping the joint acceleration bounds

Consider again the bounds (8) of Sect. III-A. In the control implementation, the joint acceleration command $\ddot{\boldsymbol{q}}$ is now kept constant at the computed value $\ddot{\boldsymbol{q}}_k = \ddot{\boldsymbol{q}}(t_k)$ for a sampling time of duration T. Suppose that at $t = t_k = kT$ the current joint position $\boldsymbol{q} = \boldsymbol{q}_k$ and velocity $\dot{\boldsymbol{q}} = \dot{\boldsymbol{q}}_k$ are both feasible. The next joint velocity and position

$$\dot{\boldsymbol{q}}_{k+1} \simeq \dot{\boldsymbol{q}}_k + \ddot{\boldsymbol{q}}_k T, \qquad \boldsymbol{q}_{k+1} \simeq \boldsymbol{q}_k + \dot{\boldsymbol{q}}_k T + \frac{1}{2}\ddot{\boldsymbol{q}}_k T^2$$

need still to be kept within their bounds. Thus, we obtain

$$-\frac{\boldsymbol{V}_{max} + \dot{\boldsymbol{q}}_k}{T} \leq \ddot{\boldsymbol{q}}_k \leq \frac{\boldsymbol{V}_{max} - \dot{\boldsymbol{q}}_k}{T} \tag{18}$$

and

$$\frac{2\left(\boldsymbol{Q}_{min} - \boldsymbol{q}_k - \dot{\boldsymbol{q}}_k T\right)}{T^2} \leq \ddot{\boldsymbol{q}}_k \leq \frac{\boldsymbol{Q}_{max} - \boldsymbol{q}_k}{T^2}. \tag{19}$$

Similarly to (12), combining the constraints given by the third set of inequalities in (8), by (18) and (19), we obtain the following box constraint for the command $\ddot{\boldsymbol{q}}$ at time $t = t_k$

$$\ddot{\boldsymbol{Q}}_{min}(t_k) \leq \ddot{\boldsymbol{q}} \leq \ddot{\boldsymbol{Q}}_{max}(t_k) \tag{20}$$

where, for $i = 1, \ldots, n$,

$$\ddot{Q}_{min,i} = \max\left\{\frac{2\left(Q_{min,i} - q_{k,i} - \dot{q}_{k,i}T\right)}{T^2}, -\frac{V_{max,i} + \dot{q}_{k,i}}{T}, -A_{max,i}\right\}$$

and

$$\ddot{Q}_{max,i} = \min\left\{\frac{2\left(Q_{max,i} - q_{k,i} - \dot{q}_{k,i}T\right)}{T^2}, \frac{V_{max,i} - \dot{q}_{k,i}}{T}, A_{max,i}\right\}.$$

However, additional caution should be used here. In fact, for $\ddot{Q}_{min,i} < 0$ to hold, it should be $\dot{q}_{k,i} \geq \left(Q_{min,i} - q_{k,i}\right)/T$, i.e., the value of the ith joint velocity is not too large negative (the term on the right of the inequality is negative by assumption). Similarly, for $\ddot{Q}_{max,i} > 0$ to hold, the inequality $\dot{q}_{k,i} \leq \left(Q_{max,i} - q_{k,i}\right)/T$ should be satisfied, i.e., $\dot{q}_{k,i}$ is not too large positive. To avoid that one of these conditions is violated, the bounds in (18) can be suitably modified.

B. The SNS algorithm at the acceleration level

The transposition of the *SNS* algorithm at the acceleration is easily obtained using still Algorithm 1. From the direct and inverse relations on acceleration

$$\ddot{\boldsymbol{x}} = \boldsymbol{J}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}(\boldsymbol{q})\dot{\boldsymbol{q}}, \qquad \ddot{\boldsymbol{q}} = \boldsymbol{J}^{\#}(\boldsymbol{q})\left(\ddot{\boldsymbol{x}} - \dot{\boldsymbol{J}}(\boldsymbol{q})\dot{\boldsymbol{q}}\right) \tag{21}$$

the *SNS* projection equation becomes

$$\ddot{\boldsymbol{q}}_{SNS} = \ddot{\boldsymbol{q}}_N + (\boldsymbol{J}\boldsymbol{W})^{\#}\left(s\ddot{\boldsymbol{x}}_d - \dot{\boldsymbol{J}}\dot{\boldsymbol{q}} - \boldsymbol{J}\ddot{\boldsymbol{q}}_N\right). \tag{22}$$

The *SNS* algorithm at the acceleration level is obtained by replacing velocity with acceleration and using (22) in Algorithm 1. Similarly, the task scaling factor is obtained with Algorithm 2 using acceleration in place of velocity and redefining $\boldsymbol{b} = \ddot{\boldsymbol{q}}_N - (\boldsymbol{J}\boldsymbol{W})^{\#}\left(\dot{\boldsymbol{J}}\dot{\boldsymbol{q}} + \boldsymbol{J}\ddot{\boldsymbol{q}}_N\right)$. Note that the affine nature of the second-order differentials map in (21), due to the presence of $\dot{\boldsymbol{J}}\dot{\boldsymbol{q}}$, implies that a scaling of joint acceleration does not produce a task scaling by the same factor.

C. Simulation results

We consider the same point-to-point task presented in Sec. III-D. The desired task acceleration $\ddot{\boldsymbol{x}}$ is simply obtained from the desired velocity as

$$\ddot{\boldsymbol{x}} = \frac{1}{T}\left(V\frac{\boldsymbol{x}_1 - \boldsymbol{x}_0}{\|\boldsymbol{x}_1 - \boldsymbol{x}_0\|} - \dot{\boldsymbol{x}}\right).$$

Figure 3 shows the Cartesian path of the robot end-effector during task execution. T indeed, with the proposed algorithm the robot goes straight to the final point, outperforming a classical pseudoinversion at the acceleration level which does not consider the presence of joint constraints. Also the motion execution time is improved since the task is completed in 1.27 s as opposed to 2.06 s.

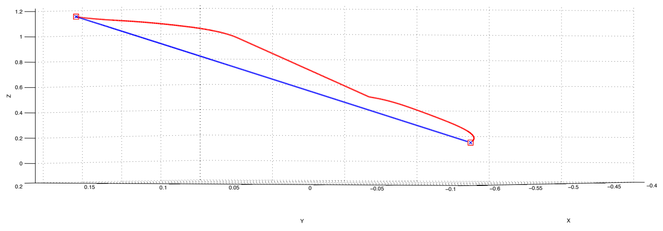

Fig. 3. End-effector trajectory for a point-to-point task. Using pseudoinversion as in (21) (red), or the *SNS* algorithm at the acceleration level (blue)

Figure 4 shows the joint trajectories obtained in the point-to-point task using the *SNS* algorithm at the acceleration level. No discontinuities are present, not even at $t = 0$, resulting in a fully feasible motion for the robot.

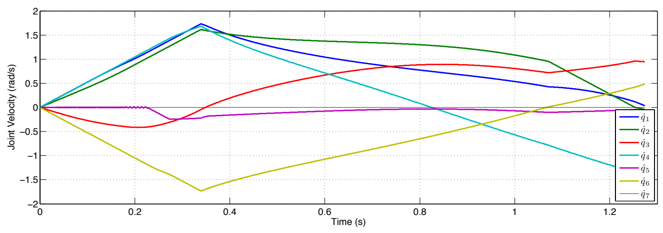

Fig. 4. Joint velocity $\dot{\boldsymbol{q}}$ of the KUKA LWR IV robot obtained with the *SNS* algorithm at the acceleration level for the point-to-point task of Fig. 3

V. Incorporating Cartesian Constraints

We propose here a simple method for incorporating the presence of constraints on robot motion from the Cartesian space to the joint space and then using the *SNS* algorithm to fulfill them. Let $\boldsymbol{C} = \begin{pmatrix} x_c & y_c & z_c \end{pmatrix}^T$ be a control point on the robot whose Cartesian displacement is limited, and $\boldsymbol{J}_C(\boldsymbol{q})$ be its associated Jacobian. For the sake of illustration, consider a simple constraint of the form $\boldsymbol{C} \geq \boldsymbol{C}_L$, where $\boldsymbol{C}_L$ is a fixed Cartesian limit. When $\boldsymbol{d} = \boldsymbol{C}(\boldsymbol{q}) - \boldsymbol{C}_L \geq 0$, we associate to the distance $\|\boldsymbol{d}\| \geq 0$ of $\boldsymbol{C}$ from the limit $\boldsymbol{C}_L$ a suitable function that goes rapidly to zero when this distance is greater than a range γ, and approaches 1 when the distance is close to zero. The sigmoid function

$$f(\boldsymbol{d}) = \frac{1}{1 + e^{\left(\|\boldsymbol{d}\|\frac{2}{\gamma} - 1\right)\alpha}},$$

can be used to this purpose, being $\alpha > 0$ is a shaping factor. Define now the vector $\boldsymbol{D} = f(\boldsymbol{d})\boldsymbol{d}/\|\boldsymbol{d}\|$, which can be interpreted as a repulsive force from the Cartesian constraint and convert it then in the joint space as

$$\boldsymbol{s} = \boldsymbol{J}^T(\boldsymbol{q})\boldsymbol{D}$$

The component s_i of $\boldsymbol{s}$ represents a "degree of influence" of the Cartesian constraint on joint i. Its sign (or, more in general, its value) can be used to reshape the velocity bounds of this joint in order to fulfill the Cartesian constraint. For this, we define

$$\begin{array}{ll} \text{if } s_i \geq 0 & \dot{Q}_{max,i} = \dot{Q}_{max,i}\,(1 - f(\boldsymbol{d})) \\ \text{else} & \dot{Q}_{min,i} = \dot{Q}_{min,i}\,(1 - f(\boldsymbol{d}))\,, \end{array} \tag{23}$$

for $i = 1, \ldots, n$. In practice, joint motions that would be in contrast with the Cartesian constraint are slowed down; moreover, when the Cartesian limit is too close, joint motions that are not compatible with this constraint will be denied. Multiple Cartesian constraints can be taken into account by considering, for each joint i, the minimum factor $1 - f(\boldsymbol{d}_j)$ obtained from all the constraints to be applied in (23).

A. Simulation results

Consider the same point-to-point task of the previous simulations. Additionally, we would like that the height z_{el} of the KUKA LWR robot elbow, i.e., the coordinate z of the 4-th link base, is always greater than a constant value z_{obs}, which emulates the presence of a horizontal obstacle (e.g., a table). The distance is then $\boldsymbol{d} = \begin{pmatrix} 0 & 0 & z_{el} - z_{obs} \end{pmatrix}^T$. The end-effector trajectory and the elbow trajectories with and without considering the Cartesian constraint are shown in Fig. 5. These were obtained using the *SNS* algorithm at the acceleration level, the parameters $\gamma = 0.14$ and $\alpha = 4$, and $z_{obs} = 0.4$ [m].

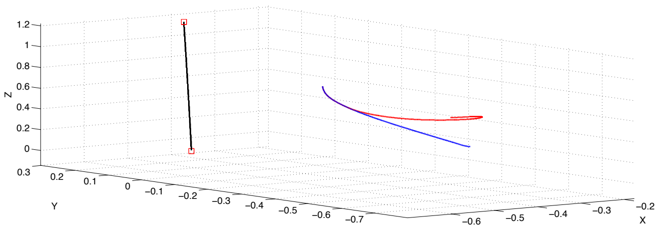

Fig. 5. End-effector trajectory (black) and elbow trajectories with (red) and without (blue) the Cartesian constraint $z_{el} \geq 0.4$ for a point-to-point task commanded at the acceleration level with the *SNS* algorithm

In both cases the task is correctly executed, and the Cartesian constraint is also satisfied when taken into account. This is more evident in Fig. 6, where the trajectory of the elbow height is shown in the two cases.

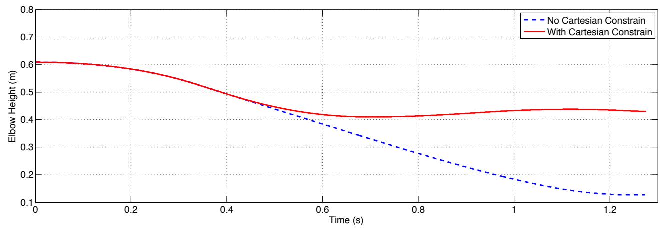

Fig. 6. Elbow height z_{el} in the point-to-point task of Fig. 5, with (red/continuous) and without (blue/dashed) the constraint $z_{el} \geq 0.4$

VI. Experimental results

A number of experiments have been realized with a 7-dof KUKA LWR IV commanded at the velocity level. The desired end-effector trajectory is a circular trajectory centered at $(0, -0.5, 0.32)$ [m] with radius is 0.3 [m], passing thus very close to the robot supporting table: the minimum nominal height of the end effector is only 2 cm. While a careful trajectory planning is usually needed to accomplish this kind of tasks, with our method the constraint is considered directly in the kinematic controller. Figure 7 shows the result of the experiment. The reader is referred to the accompanying video clip for a better appreciation of the robot motion.

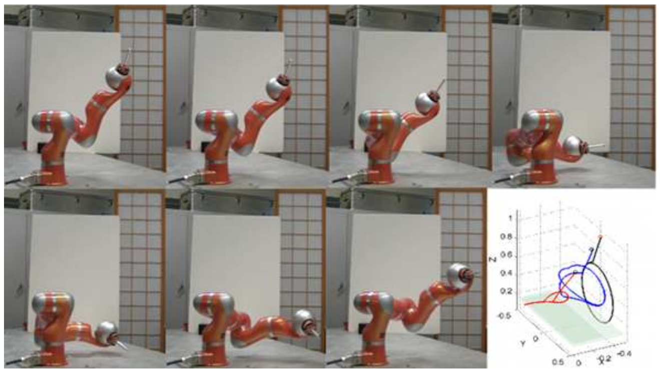

Fig. 7. Snapshots from an experiment with the KUKA LWR IV and plot of the relevant trajectories: end-effector (black), wrist (blue), elbow (red)

The Cartesian constraints were mapped as minimum height limits for the elbow and the wrist, respectively equal to 0.15 [m] and 0.12 [m]. Their satisfaction is shown in detail in Fig. 8, while the obtained modulation of the joint velocity bounds during the experiment is illustrated in Fig. 9.

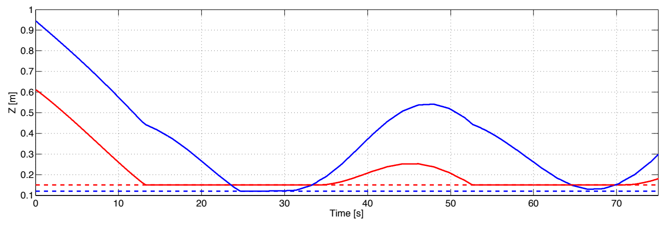

Fig. 8. Elbow (red) and wrist (blue) trajectories in the experiment, with dashed lines representing the associated limits

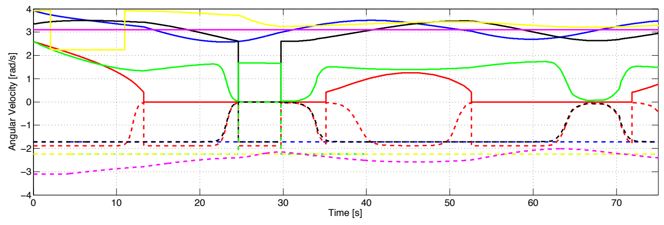

Fig. 9. Maximum (solid) and minimum (dashed) joint velocity bounds: $\dot{q}_1$ (blue), $\dot{q}_2$ (red), $\dot{q}_3$ (black), $\dot{q}_4$ (green), $\dot{q}_5$ (yellow), and $\dot{q}_6$ (magenta)

VII. Conclusions

We have presented a new redundancy resolution method that algorithmically explores the possibility of redistributing joint motion commands so as to: *i)* satisfy hard bounds on joint range, joint velocity, and joint acceleration; *ii)* guarantee task preservation, if at least one feasible solution exists; *iii)* use a reduced number of saturated commands; *iv)* achieve a minimum norm property for the remaining enabled (unsaturated) joints; *v)* automatically introduce a minimum task scaling, if and only if the original task is not feasible with the given bounds. The simultaneous achievement of all these requirements is a distinctive feature of the proposed method, as opposed to all previously existing works.

Acknowledgements

This work is supported by the European Community, within the FP7 ICT-287513 SAPHARI project.

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*. Reading, MA, USA: Addison-Wesley, 1991.
[3] A. Liegeois, "Automatic supervisory control of the configuration and behavior of multibody mechanisms," *IEEE Trans. Syst., Man, Cybern.*, vol. 7, pp. 245–250, 1977.
[4] C. Samson, M. L. Borgne, and B. Espiau, *Robot Control: The Task Function Approach*. Clarendon, Oxford, UK, 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*, vol. 11, no. 2, pp. 286–292, 1995.
[6] F. Chaumette and E. Marchand, "A new redundancy-based iterative scheme for avoiding joint limits: Application to visual servoing," in *Proc. IEEE Int. Conf. on Robotics and Automation*, 2000, pp. 1720–1725.
[7] N. Mansard and F. Chaumette, "Visual servoing sequencing able to avoid obstacles," in *Proc. IEEE Int. Conf. on Robotics and Automation*, 2005, pp. 3143 – 3148.
[8] 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.
[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] 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.
[11] 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.
[12] 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.
[13] 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.
[14] 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.
[15] C. D. Meyer, "Generalized inversion of modified matrices," *SIAM J. of Applied Mathematics*, vol. 24, no. 3, pp. 315–323, 1973.