2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

Redundancy resolution under hard joint constraints: a generalized approach to rank updates

Anton Ziese[1,2], Mario D. Fiore[1,⋆], Jan Peters[2], Uwe E. Zimmermann[1] and Jürgen Adamy.[2]

***Abstract*— The increasing interest in autonomous robots with a high number of degrees of freedom for industrial applications and service robotics have also increased the demand for efficient control algorithms. The unstructured environment these robots operate in often impose constraints on the joint motion, an important type being the joint limits of the robot itself. These circumstances demand control algorithms to handle multiple tasks as well as constraints efficiently. This paper shows that both kinematic and torque control of redundant robots under hard joint constraints can be formulated in a single framework as a constrained optimization problem. To solve said problem, a generalization of the Fast-SNS algorithm to weighted pseudoinverses is proposed, which fulfills our demand of efficiently and reliably handling joint constraints.**

I. Introduction

This paper makes contributions to the field of controlling redundant robots, i.e. robots with more degrees of freedom (DOF) than are needed to perform a given task. The main benefit of a redundant robot is that it offers multiple solutions for a given task (e.g. an end-effector trajectory), and thus a joint configuration can be chosen that also fulfills secondary objectives; often, obstacle avoidance is chosen as a secondary task, other important examples include self-collision or joint limit avoidance [1]. These secondary objectives are especially important for robots operating autonomously in unstructured environments, which have seen an increasing demand in industrial applications and service robotics in recent years. Also, since the tasks are subject to change during runtime, a control algorithm is needed that can find a redundancy resolution for prioritized tasks online. Task execution should not be deformed by saturation of the robot joints, thus, since the robot's reaction to the environment is not predictable, joint limit avoidance must also be handled online. Alternatively, one or more tasks may become unfeasible altogether due to some change in the environment, in which case the robot must retain robust and predictable behavior. It stands to reason to exploit contingent redundancies to find a solution that does not violate joint limits, or bring the robot to a controlled stop otherwise. When considering robots with a higher number of DOFs, the computational efficiency of these algorithms becomes increasingly important in light of real-time capability.

[1]Authors are with Corporate Research of KUKA Deutschland GmbH, Zugspitzstraße 140, 86165 Augsburg, DE

[2]Authors are with TU Darmstadt, Karolinenplatz 5, 64289 Darmstadt, DE

⋆ Corresponding author. Mario.Fiore@kuka.com

This work was partly supported by the German Federal Ministry of Education and Research (BMBF) through the project Hybr-iT (grant no. 01IS16026A).

Prioritized multitasking has been thoroughly explored in the literature, the most common method using the well known augmented projector [2] [3]. Joint limit avoidance is a less clear-cut subject, though the literature provides many approaches in the context of online local planning. A common one is to minimize a cost function that increases as a joint approaches a limit; the joint limit avoidance is then realized with the same [4] or lower [5] priority than the main task, meaning adherence to the joint limits is not guaranteed. Also, these approaches favor a joint configuration where each joint is in the center of its range, even though there might be a better solution. To find the optimal solution, the control problem can be formulated as a constrained optimization problem, which is often solved using quadratic programming (QP) [6] [7]; however, these typically present higher computation time [8] and do not feature a defined behavior in case a task is not feasible.

A family of algorithms known as SNS (Saturation in the Null Space) offer a solution: these use task scaling, first introduced in [9], as an additional strategy, while exploiting redundancy similarly to active set QP solvers. In essence, the SNS uses redundant DOFs to find a solution that requires the least scaling of a task to satisfy all joint limits. While the Opt-SNS guarantees optimal solutions, this requirement can be dropped in favor of computational speed, while still offering a far larger solution space than the other approaches detailed above. Especially for high number of DOFs, the Fast-SNS variant has proven very powerful [10], which, as the name suggests, is especially efficient. An overview of all SNS variants is given in [11].

Most of the above mentioned literature utilizes velocity-based redundancy resolution, i.e. describes the robot using a first order kinematic equation, because of its mathematical simplicity. However, moving to second-order algorithms, i.e. acceleration or torque level, promises some advantages, e.g. enabling the joint limit avoidance to also limit the maximum acceleration, improving the noise, vibration and harshness (NVH) behavior of the robot. Torque control additionally offers compliance, meaning the robot will yield at physical contact. Both of the aforementioned points are especially important in human-robot collaboration. The other obvious reason to use torque control is that a given task demands a certain force.

Discussing different strategies to control a redundant system raises the question if it isn't possible to find a more general description of the control problem. Therefore, the goal of this paper is twofold: first, we will show that all of the aforementioned control schemes (velocity, acceleration

and torque control) can be brought into a single framework and thus we can also design a single solver for all of them; second, we will present such a solver which generalizes the Fast-SNS algorithm to constrained optimization problems with arbitrary weighting matrices.

Secs. II and III give the mathematical background for these goals. Subsequently in sec. IV we will evaluate the generalized Fast-SNS on velocity, acceleration and torque level, showing how different control schemes can easily be implemented with our unified formulation. Finally, we discuss the results in sec. V.

II. Generalized control problem

As mentioned in the introduction, there are different approaches to controlling redundant robots depending on the equation used to describe the robot. Velocity [12] and acceleration-based [13] schemes describe the robot using a kinematic equation on velocity and acceleration level, respectively; the output of the solver is then a vector of joint velocities or accelerations, which are typically integrated to joint positions and given as input to underlying joint position controllers. A torque-based approach [14], as the name suggests, outputs a vector of joint torques instead. These are then directly sent to joint actuators and an actual control loop is closed using measured joint position and velocity. Velocity-based redundancy resolution is used for its mathematical simplicity, and torque control because it includes the robot dynamics, with acceleration-based algorithms somewhere in between in terms of complexity and abstraction. All of these schemes can be brought into the form:

$$\mathbf{A}\mathbf{u} = \mathbf{b} \tag{1}$$

where $\mathbf{u} \in \mathbb{R}^n$ is the output of the solver, $\mathbf{b} \in \mathbb{R}^m$ a vector describing the robot task, and $\mathbf{A} \in \mathbb{R}^{m\times n}$ a mapping between the two. The number of robot joints is given by n, and the dimension of the task by m. The definition of these variables for the different control schemes can be found in Tab. I. For sake of clarity, dependencies from the robot joint configuration are omitted. Inputting the content of a column in Eq. 1 will yield the basic equations found in [12], [13] or [14] respectively.

As stated in the previous section, the problem of controlling a redundant robot under joint constraints can be formulated as a constrained optimization problem:

$$\begin{aligned} &\min \mathbf{u}^{\mathrm{T}}\mathbf{N}\mathbf{u} \\ &\text{s.t. } \mathbf{A}\mathbf{u} = \mathbf{b},\ \mathbf{E}\mathbf{u} \leq \mathbf{d} \end{aligned} \tag{2}$$

where $\mathbf{N} \in \mathbb{R}^{n\times n}$ is an arbitrary, (semi) positive definite weighting matrix, $\mathbf{d} \in \mathbb{R}^c$ a vector describing the c joint constrains and $\mathbf{E} \in \mathbb{R}^{c\times n}$ a mapping between $\mathbf{u}$ and $\mathbf{d}$. Again the definitions for $\mathbf{E}$ and $\mathbf{d}$ can be found in Tab. I. The choice of the metric $\mathbf{u}^{\mathrm{T}}\mathbf{N}\mathbf{u}$ is left to the programmer, some typical examples are presented in sec. IV. Omitting the inequality constraints, the solution for Eq. 2 is given by the weighted pseudoinverse of $\mathbf{A}$ [12]:

$$\mathbf{u} = \mathbf{A}^{\dagger N}\mathbf{b}. \tag{3}$$

In this paper, the weighted pseudoinverse is computed as:

$$\mathbf{A}^{\dagger N} = \mathbf{N}^{-1}\mathbf{A}^{\mathrm{T}}\left(\mathbf{A}\mathbf{N}^{-1}\mathbf{A}^{\mathrm{T}}\right)^{-1}. \tag{4}$$

TABLE I: Definitions for Eq. 2 for velocity, acceleration and torque-based schemes with a desired task space trajectory $\mathbf{x} \in \mathbb{R}^m$; $\mathbf{q} \in \mathbb{R}^n$ is the vector of robot joint positions, $\mathbf{J} \in \mathbb{R}^{m\times n}$ the jacobian of the robot, $\boldsymbol{\tau} \in \mathbb{R}^n$ the vector of joint torques, $\mathbf{M} \in \mathbb{R}^{n\times n}$ the robot's mass matrix, $\mathbf{C} \in \mathbb{R}^n$ a vector compensating coriolis, centrifugal and gravitational forces, $\ddot{\mathbf{Q}}_{\max} \in \mathbb{R}^n$ and $\ddot{\mathbf{Q}}_{\min} \in \mathbb{R}^n$ are upper and lower acceleration bounds, and $\dot{\mathbf{Q}}_{\max} \in \mathbb{R}^n$ and $\dot{\mathbf{Q}}_{\min} \in \mathbb{R}^n$ upper and lower velocity bounds, respectively. $\mathbf{I} \in \mathbb{R}^{n\times n}$ is the identity matrix.

	velocity	**acceleration**	**torque**
$\mathbf{A}$	$\mathbf{J}$	$\mathbf{J}$	$\mathbf{J}\mathbf{M}^{-1}$
$\mathbf{u}$	$\dot{\mathbf{q}}$	$\ddot{\mathbf{q}}$	$\boldsymbol{\tau}$
$\mathbf{b}$	$\dot{\mathbf{x}}$	$\ddot{\mathbf{x}} - \dot{\mathbf{J}}\dot{\mathbf{q}}$	$\ddot{\mathbf{x}} - \dot{\mathbf{J}}\dot{\mathbf{q}} - \mathbf{J}\mathbf{M}^{-1}\mathbf{C}$
$\mathbf{E}$	$\begin{bmatrix}\mathbf{I} & -\mathbf{I}\end{bmatrix}^{\mathrm{T}}$	$\begin{bmatrix}\mathbf{I} & -\mathbf{I}\end{bmatrix}^{\mathrm{T}}$	$\begin{bmatrix}\mathbf{M}^{-1} & -\mathbf{M}^{-1}\end{bmatrix}^{\mathrm{T}}$
$\mathbf{d}$	$\begin{bmatrix}\dot{\mathbf{Q}}_{\max}^{\mathrm{T}} & \dot{\mathbf{Q}}_{\min}^{\mathrm{T}}\end{bmatrix}^{\mathrm{T}}$	$\begin{bmatrix}\ddot{\mathbf{Q}}_{\max}^{\mathrm{T}} & \ddot{\mathbf{Q}}_{\min}^{\mathrm{T}}\end{bmatrix}^{\mathrm{T}}$	$\begin{bmatrix}\ddot{\mathbf{Q}}_{\max}^{\mathrm{T}} & \ddot{\mathbf{Q}}_{\min}^{\mathrm{T}}\end{bmatrix}^{\mathrm{T}}$

A solution for multiple tasks specified with different priorities can be computed by using nullspace projections:

$$\begin{aligned} \mathbf{u}_k &= \mathbf{u}_{k-1} + \left(\mathbf{A}_k\mathbf{P}_{\mathrm{A},k-1}\right)^{\dagger N}\left(\mathbf{b}_k - \mathbf{A}_k\mathbf{u}_{k-1}\right) \\ \mathbf{P}_{\mathrm{A},0} &= \mathbf{I} \\ \mathbf{u}_0 &= \mathbf{0} \end{aligned} \tag{5}$$

where $\mathbf{u}_k$ denotes the solution for the k-th task (lower k denoting higher priority) and $\mathbf{P}_{\mathrm{A},k} \in \mathbb{R}^{n\times n}$ is the well known augmented projector from [2]. Finding a solution that fulfills also the inequality constraints is more complex however, and is the subject of the following chapter.

III. Generalized Fast-SNS

As remarked in sec. I, the SNS algorithm is a powerful tool for online control of redundant robots under hard joint constraints. In multiple iterations, it computes which joint requires the most scaling to be within its limits, and subsequently adds the inequality constraint of that limit to the equality constraints of the task. Then, a new scaling factor is computed for the task; this cycle is repeated until either a solution that does not require scaling is found, or all redundancies of the robot are exhausted. In the latter case, the solution that requires the least scaling of the task is returned.

In this section, we will show how to integrate the SNS into the unified control framework introduced in the previous section. For the regular SNS algorithm presented on velocity level in [11], this is easily achieved by substituting $\dot{\mathbf{q}}$, $\dot{\mathbf{x}}$ and $\mathbf{J}$ with $\mathbf{u}$, $\mathbf{b}$ and $\mathbf{A}$, and replacing the Moore-Penrose pseudoinverse with a weighted pseudoinverse (the former is a special case of a weighted pseudoinverse with $\mathbf{N} = \mathbf{I}$). The Fast-SNS, which we favor for its computational efficiency, requires more complex deliberations, as it exploits some properties of the Moore-Penrose pseudoinverse which are not so easily transferred to the general case. To start, the solution of Eq. 2 for the k-th task considering only the equality constraint is written as:

$$\mathbf{u}_k = \mathbf{u}_{k-1} + \left(\mathbf{A}_k\mathbf{P}_{\mathrm{A},k-1}\right)^{\dagger N}\left(s\cdot\mathbf{b}'_k - \mathbf{b}''_k - \mathbf{A}_k\mathbf{u}_{k-1}\right) \tag{6}$$

with $\mathbf{b}'_k \in \mathbb{R}^n$, $\mathbf{b}''_k \in \mathbb{R}^n$ defined as in Tab. II. As in [11], Alg. 1 is used to compute a task scaling factor s so that all joints are within their limits, as well as to determine the most

TABLE II: Definitions for $\mathbf{b}'$ and $\mathbf{b}''$ from Eq. 6 for velocity, acceleration and torque-based schemes

	velocity	acceleration	torque
$\mathbf{b}'$	$\dot{\mathbf{x}}$	$\ddot{\mathbf{x}}$	$\ddot{\mathbf{x}}$
$\mathbf{b}''$	$\mathbf{0}$	$\dot{\mathbf{J}}\dot{\mathbf{q}}$	$\dot{\mathbf{J}}\dot{\mathbf{q}} - \mathbf{J}\mathbf{M}^{-1}\mathbf{F}$

critical joint that requires the most scaling. The arguments $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are given as:

$$\begin{aligned} \boldsymbol{\alpha} &= \mathbf{E}_{\mathrm{n}} \left(\mathbf{A}_k \mathbf{P}_{\mathrm{A},k-1}\right)^{\dagger N} \mathbf{b}'_k \\ \boldsymbol{\beta} &= \mathbf{E}_{\mathrm{n}} \mathbf{u}_k - \boldsymbol{\alpha} \end{aligned} . \tag{7}$$

Where $\mathbf{E}_{\mathrm{n}} \in \mathbb{R}^{c/2 \times n}$ is equal to the first n rows of $\mathbf{E}$. Analogous to [10], the most critical joint is saturated to its extremal value by augmenting the current task with a one-dimensional saturation task:

$$\mathbf{u}_k = \mathbf{u}_{k-1} + \begin{bmatrix} \mathbf{A}_k \mathbf{P}_{\mathrm{A},k-1} \\ \mathbf{A}_{\mathrm{sat}} \mathbf{P}_{\mathrm{A},k-1} \end{bmatrix}^{\dagger N} \begin{bmatrix} s \cdot \mathbf{b}'_k - \mathbf{b}''_k - \mathbf{A}_k \mathbf{u}_{k-1} \\ \mathbf{b}_{\mathrm{sat}} - \mathbf{A}_{\mathrm{sat}} \mathbf{u}_{k-1} \end{bmatrix} \tag{8}$$

where $\mathbf{A}_{\mathrm{sat}} \in \mathbb{R}^{r \times n}$ contains the rows of $\mathbf{E}$ corresponding to the r saturated joints, and $\mathbf{b}_{\mathrm{sat}} \in \mathbb{R}^r$ the respective elements of $\mathbf{d}$, with a new row and element respectively added at each iteration. The new solution is then computed by rank update of the pseudoinverse, which is the main source of savings in computational time of the Fast-SNS; the reader is referred to [10] for further details. One of the main contributions of this paper is the solution by rank update for arbitrary weighting matrices, which has been derived from the algorithm in [15] for appending a column. When saturating the i-th joint, the solution is updated as:

$$\begin{aligned} \mathbf{u}_k &= \mathbf{u}_{k-1} + \begin{bmatrix} \mathbf{A}_k \mathbf{P}_{\mathrm{A},k-1} \\ \mathbf{A}_{\mathrm{sat},i} \mathbf{P}_{\mathrm{A},k-1} \end{bmatrix}^{\dagger N} \begin{bmatrix} s \cdot \mathbf{b}'_k - \mathbf{b}''_k - \mathbf{A}_k \mathbf{u}_{k-1} \\ b_{\mathrm{sat},i} - \mathbf{A}_{\mathrm{sat},i} \mathbf{u}_{k-1} \end{bmatrix} \\ &= \mathbf{u}_{k-1} + \begin{bmatrix} \mathbf{X}_k - \boldsymbol{\chi}_{i,k} \mathbf{A}_{\mathrm{sat},i} \mathbf{X}_k & \boldsymbol{\chi}_{i,k} \end{bmatrix} \\ &\quad \times \begin{bmatrix} s \cdot \mathbf{b}'_k - \mathbf{b}''_k - \mathbf{A}_k \mathbf{u}_{k-1} \\ b_{\mathrm{sat},i} - \mathbf{A}_{\mathrm{sat},i} \mathbf{u}_{k-1} \end{bmatrix} \\ \mathbf{X}_k &= \left(\mathbf{A}_k \mathbf{P}_{\mathrm{A},k-1}\right)^{\dagger N} \\ \boldsymbol{\chi}_{i,k} &= \left(\mathbf{A}_{\mathrm{sat},i} \mathbf{P}_{\mathrm{A},k}\right)^{\dagger N} . \end{aligned} \tag{9}$$

with the new update vector $\boldsymbol{\chi}_{i,k}$ for a weighted pseudoinverse, and $\mathbf{A}_{\mathrm{sat},i}$ being the i-th row of $\mathbf{E}$. Pseudocode for the entire algorithm is given in Alg. 2. The main benefit of the proposed solution is that for each iteration, only the (weighted) pseudoinverse of a vector must be computed for $\boldsymbol{\chi}_{i,k}$, instead of a matrix. This makes the algorithm very efficient. The initial computation of the pseudoinverse is also critical for computation time; the programmer must decide on a trade-off between speed and robustness. In this work, the QR-Decomposition is used to compute the inverse in Eq. 6, which focuses on speed. A more robust alternative is e.g. the singular value decomposition (SVD) [16].

An open question is the shaping of the joint limits, namely finding $\dot{Q}_{\mathrm{max/min}}$ (velocity-based redundancy resolution) or $\ddot{Q}_{\mathrm{max/min}}$ (acceleration and torque-based schemes). The core problem lies in finding velocity/acceleration bounds that also ensure position limits are adhered to. For the velocity-based solver, the boundaries for the i-th joint in the l-th time step

Algorithm 1 Task scaling factor

function GETTASKSCALINGFACTOR($\boldsymbol{\alpha}$,$\boldsymbol{\beta}$)
  for $i = 1 \rightarrow n$ **do**
    $S_{\mathrm{min},i} = \left(\ddot{Q}_{\mathrm{min},i} - \beta_i\right)/\alpha_i$
    $S_{\mathrm{max},i} = \left(\ddot{Q}_{\mathrm{max},i} - \beta_i\right)/\alpha_i$
    if $S_{\mathrm{min},i} > S_{\mathrm{max},i}$ **then**
      switch $S_{\mathrm{min},i}$ and $S_{\mathrm{max},i}$
    end if
  end for
  $s_{\mathrm{max}} = \min_i \{\mathbf{S}_{\mathrm{max}}\}$
  $s_{\mathrm{min}} = \max_i \{\mathbf{S}_{\mathrm{min}}\}$
  most critical joint $= \mathrm{argmin}_i \{\mathbf{S}_{\mathrm{max}}\}$
  if $s_{\mathrm{min}} > s_{\mathrm{max}} \vee s_{\mathrm{max}} < 0 \vee s_{\mathrm{min}} > 1$ **then**
    task scaling factor $= 0$
  else
    task scaling factor $= \min\{s_{\mathrm{max}}, 1\}$
  end if
end function

at the position $q_{l,i}$ are taken from [11]:

$$\begin{aligned} \dot{Q}_{\mathrm{min},i} &= \max\left\{(Q_{\mathrm{min},i} - q_{l,i})/\Delta T,\, V_{\mathrm{min},i},\, \dot{Q}_{\mathrm{min,acc}}\right\} \\ \dot{Q}_{\mathrm{max},i} &= \min\left\{(Q_{\mathrm{max},i} - q_{l,i})/\Delta T,\, V_{\mathrm{max},i},\, \dot{Q}_{\mathrm{max,acc}}\right\} \\ \dot{Q}_{\mathrm{min,acc},i} &= -\sqrt{2A_{\mathrm{min},i}\left(Q_{\mathrm{min},i} - q_{l,i}\right)} \\ \dot{Q}_{\mathrm{max,acc},i} &= \sqrt{2A_{\mathrm{max},i}\left(Q_{\mathrm{max},i} - q_{l,i}\right)} \end{aligned} \tag{10}$$

with the absolute position, velocity and acceleration bounds $Q_{\mathrm{max/min},i}$, $V_{\mathrm{max/min},i}$ and $A_{\mathrm{max/min},i}$. ΔT is the discrete time step. For acceleration/torque control, joint boundary computation from [17] is used. The concept of viability of a state is introduced, which demands that the robot can be brought to a full stop during the next time step. Boundaries from viability are obtained by replacing joint velocities and positions in Eq. 10 with the discrete difference equations w.r.t. to the joint acceleration $\ddot{q}_i$. An anticipation coefficient $h > \Delta T$ is used in place of the cycle time ΔT of the controller to provide some conservatism. The reader is referred to [17] for a detailed derivation.

IV. Experiments

In this section we will prove that the proposed generalized Fast-SNS solver retains the key features discussed in the previous sections: joint limit avoidance, task prioritization, task scaling and computational efficiency. Experiments are carried out on a KUKA LBR iiwa 7-DOF robot (Fig. 1), as well as on highly-redundant mobile dual-arm system (Fig. 9).

A. Experiments with LBRiiwa

In the first set of experiments the robot performs the same task with the generalized Fast-SNS solver working at velocity, acceleration and torque level. In the first two cases the output of the solver is integrated to obtain a joint position reference, which is then provided to an underlying

Algorithm 2 Generalized Fast-SNS algorithm for multiple tasks

$\mathbf{P}_{\mathrm{A},0} = \mathbf{I}$, $\mathbf{u}_0 = \mathbf{0}$, $s_k = 0$
for $k = 1 \rightarrow l$ **do**
  $\mathbf{X}_k = (\mathbf{A}_k \mathbf{P}_{\mathrm{A},k-1})^{\dagger N}$
  $\mathbf{u}'_k = \mathbf{X}_k \mathbf{b}'_k$
  $\mathbf{u}''_k = -\mathbf{X}_k \left(\mathbf{b}''_k + \mathbf{A}_k \mathbf{u}_{k-1}\right)$
  $\mathbf{u}_{\mathrm{W}} = \mathbf{0}$
  $\mathbf{P}_{\mathrm{A},k} = \mathbf{P}_{\mathrm{A},k-1} - \mathbf{X}_k \left(\mathbf{J}_k \mathbf{P}_{\mathrm{A},k-1}\right)$
  $\bar{\mathbf{P}}_{\mathrm{A},k-1} = \mathbf{P}_{\mathrm{A},k}$
  repeat
    limits_violated = FALSE
    $\mathbf{u}_k = \mathbf{u}_{k-1} + \mathbf{u}' + \mathbf{u}'' + \mathbf{u}_{\mathrm{W}}$
    $\boldsymbol{\alpha} = \mathbf{E}_{\mathrm{n}} \mathbf{u}'$, $\boldsymbol{\beta} = \mathbf{E}_{\mathrm{n}} \mathbf{u}_k - \boldsymbol{\alpha}$
    $\boldsymbol{\gamma} = \boldsymbol{\alpha} + \boldsymbol{\beta}$
    if $\exists i \in [1:n] : (\gamma_i < d_{\min,i}) \vee (\gamma_i > d_{\max,i})$ **then**
      limits_violated = TRUE
      getTaskScalingFactor($\boldsymbol{\alpha}$,$\boldsymbol{\beta}$) ▷ call alg. 1
      if task scaling factor $> s^*_k$ **then**
        $s^*_k =$ task scaling factor
        $\mathbf{u}^{*'} = \mathbf{u}'$, $\mathbf{u}^{*''} = \mathbf{u}''$, $\mathbf{u}^*_{\mathrm{W}} = \mathbf{u}_{\mathrm{W}}$
      end if
      $j =$ most critical joint
      $b_{\mathrm{sat},j} = \begin{cases} d_{\min,j}, & \gamma_j < d_{\min,j} \\ d_{\max,j}, & \gamma_j > d_{\max,j} \end{cases}$
      $\boldsymbol{\chi}_{j,k} = (\mathbf{A}_{\mathrm{sat},j} \mathbf{P}_{\mathrm{A},k})^{\dagger N}$
      $\mathbf{u}' = \mathbf{u}' - \boldsymbol{\chi}_{j,k} \mathbf{A}_{\mathrm{sat},j} \mathbf{u}'$
      $\mathbf{u}'' = \mathbf{u}'' - \boldsymbol{\chi}_{j,k} \mathbf{A}_{\mathrm{sat},j} \mathbf{u}''$
      $\mathbf{u}_{\mathrm{W}} = \mathbf{u}_{\mathrm{W}}$
        $+\boldsymbol{\chi}_{j,k} \left(\mathbf{b}_{\mathrm{sat}} - \mathbf{A}_{\mathrm{sat},j} \left(\mathbf{u}_{k-1} + \mathbf{u}_{\mathrm{W}}\right)\right)$
      $\bar{\mathbf{P}}_{\mathrm{A},k} = \mathbf{P}_{\mathrm{A},k} - \boldsymbol{\chi}_{j,k} \mathbf{A}_{\mathrm{sat},j} \bar{\mathbf{P}}_{\mathrm{A},k}$
      $\boldsymbol{\chi}_{j,k-1} = (\mathbf{A}_{\mathrm{sat},j} \mathbf{P}_{\mathrm{A},k-1})^{\dagger N}$
      $\bar{\mathbf{P}}_{\mathrm{A},k-1} = \bar{\mathbf{P}}_{\mathrm{A},k-1} - \boldsymbol{\chi}_{j,k-1} \mathbf{A}_{\mathrm{sat},j} \bar{\mathbf{P}}_{\mathrm{A},k-1}$
      if rank $\left(\mathbf{A}_k \bar{\mathbf{P}}_{\mathrm{A},k-1}\right) < m$ **then**
        $s_k = s^*_k$, $\mathbf{u}' = \mathbf{u}^{*'}$, $\mathbf{u}'' = \mathbf{u}^{*''}$, $\mathbf{u}_{\mathrm{W}} = \mathbf{u}^*_{\mathrm{W}}$
        $\mathbf{u}_k = \mathbf{u}_{k-1} + s_k \mathbf{u}' + \mathbf{u}'' + \mathbf{u}_{\mathrm{W}}$
        limits_violated = FALSE
      end if
    end if
  until limits_violated = FALSE
end for
$\mathbf{u}_{\mathrm{tot}} = \mathbf{u}_l$

position controller; in the last case the output of the algorithm is directly fed to the joint actuators and an actual control loop is implemented. The required task is to track a desired Cartesian trajectory $(\mathbf{x}_d, \dot{\mathbf{x}}_d, \ddot{\mathbf{x}}_d)$ with the center point of the robot end effector. Fig. 1 shows the Cartesian path used for the experiments. The robot is commanded to move on each segment of the star, returning every time to the center point, following a sinusoidal velocity profile. The total planned time is 10 seconds (1,25 seconds per segment). The initial end-effector orientation must be kept along the entire trajectory. Therefore, both Cartesian position and orientation of the robot are controlled and only one redundant degree of freedom exists. The initial configuration of the robot, which can be seen in the top-left corner of Fig. 1, places the second and the sixth joint very close to their respective upper position limit. Additional limitations set on joint velocity and acceleration can be seen in Tab. III, which gives an overview of the initial setup. The velocity-based solver uses the joint

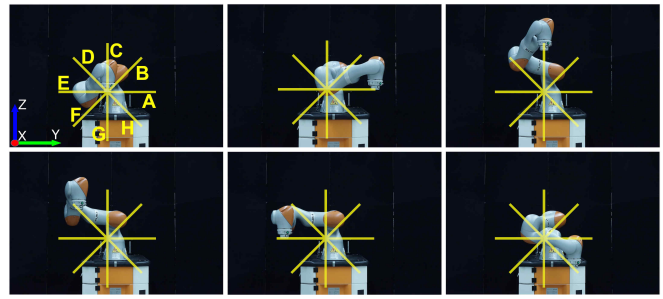

Fig. 1: LBRiiwa moving on a star-like path during the experiments. The path is defined on the YZ plane. Each star segment (indicated with a capital letter) has a length of 24 cm.

TABLE III: Initial configuration and joint limits for the LBRiiwa. The first joint is at the base of the robot.

Joint nr.	Position Lim. $[rad]$	Velocity Lim. $[rad/s]$	Acceleration Lim. $[rad/s^2]$	Initial Config. $[rad]$
1	± 2.9234	± 1.45	± 10	-0.7854
2	± 2.0508	± 1.45	± 10	$+2.0502$
3	± 2.9234	± 1.45	± 10	$+2.0721$
4	± 2.0508	± 1.45	± 10	-1.6563
5	± 2.9234	± 1.45	± 10	-2.0893
6	± 2.0508	± 1.45	± 10	$+2.0342$
7	± 3.0107	± 1.45	± 10	0

bounds in Eq. 10 and $\mathbf{N} = \mathbf{M}$. The reference Cartesian trajectory is defined over the time t on velocity level as $\dot{\mathbf{x}}_r(t) = \dot{\mathbf{x}}_d(t) + k_p \tilde{\mathbf{x}}(t)$, with k_p being a positive gain, which has been set to 50, and $\tilde{\mathbf{x}}$ a position error term. The output joint velocities are shown in Fig. 2 and have been integrated and differentiated, so as to obtain the corresponding joint position and acceleration. For sake of clarity, all the values are reported normalized w.r.t. the maximum/minimum limits defined in Tab. III. The occurrence of saturation can be easily identified, proving the effectiveness of the proposed algorithm in respecting the hard bounds imposed on joint positions and velocities. As it often occurs with velocity-based solvers, fulfillment of the maximum joint acceleration cannot be guaranteed ($A_{\min/\max}$ in Eq. 10 is only used for shaping of the joint bounds $\dot{Q}_{\min/\max}$ in the proximity of the position limits [11]). The moments in which saturation occur can also be identified in Fig. 3, where the trend of the task scaling factor and the number of executed iterations are shown. Saturation (on either joint position or velocity) requires an additional iteration. In most cases, the algorithm finds a solution after saturating one joint. However, this is not possible when the robot approaches segment E of the star. In this case, a scale factor $s < 1$ shows that the task has become unfeasible, i.e. it was not possible for the algorithm

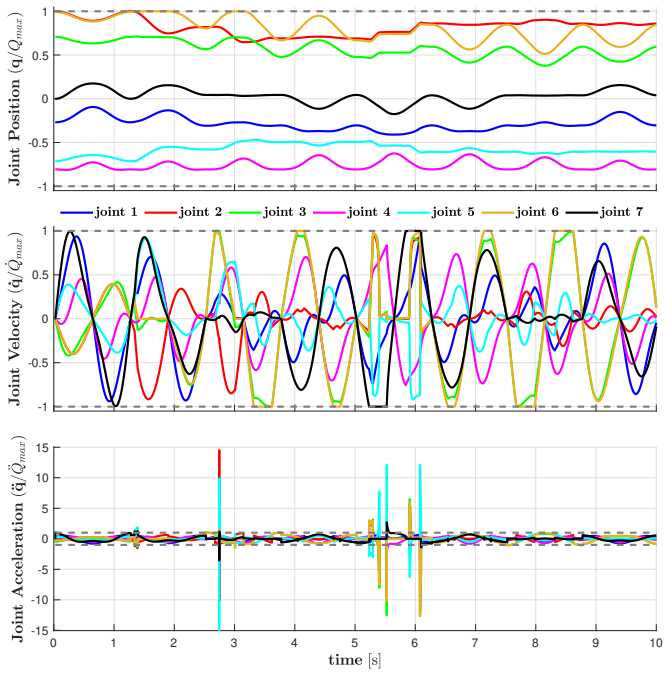

Fig. 2: Normalized joint position, velocity and acceleration produced by the velocity-based solver for the LBRiiwa.

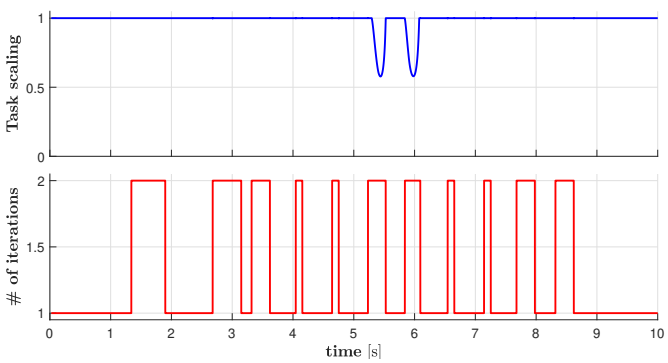

Fig. 3: Task scaling factor and number of iterations of the velocity-based solver for the LBRiiwa; as the robot has only one redundant degree of freedom, the algorithm can perform a maximum of two iterations.

to find a solution within the joint bounds. The task velocity reference $\dot{\mathbf{x}}_r(t)$ is then scaled down to ensure compliance to the joint limits.

To evaluate the computational efficiency of the algorithm, the same motion has been executed using a standard SNS solver [11], re-implemented using a weighted pseudo-inverse. The produced joint motion is, as expected, identical to the one shown in Fig. 2 and therefore not reported. The difference between the two algorithms lies in the computation time. The computation times per cycle are shown in Tab. IV. Both algorithms where executed in a VxWorks (32bit) real-time module running on one of the four cores of an Intel i5-45705 (2.89GHz) CPU with 432 MB of dedicated RAM. The cycle time of the real-time system is 1 ms. The advantage of the proposed algorithm is quite clear, as each additional iteration requires approximately half the time of the initial computation, whereas additional iterations of the regular SNS require about 88% the time required by the initial computation. Additionally, the computation time for the first iteration is significantly shorter due to the use of the QR-Decomposition, whereas an SVD decomposition is used in the standard SNS.

TABLE IV: Average computation time for solving a 6-DOF Cartesian task with a LBRiiwa; for the generalized Fast-SNS additional iterations are always significantly shorter; acceleration and torque based algorithms take slightly longer due to more complex equations. The average is computed over the entire execution time repeating each experiment five times.

	First iteration	Additional iteration
SNS vel. ($\mathbf{N} = \mathbf{M}$)	33μs	29μs
Gen. Fast-SNS vel.($\mathbf{N} = \mathbf{M}$)	17μs	9μs
Gen. Fast-SNS acc. ($\mathbf{N} = \mathbf{M}$)	18μs	10μs
Gen. Fast-SNS torq. ($\mathbf{N} = \mathbf{M}^{-1}$)	21μs	11μs

The acceleration-based solver uses $\mathbf{N} = \mathbf{M}$ and the joint bounds in [17] with $h = 4\Delta T$. The reference Cartesian trajectory is defined as $\ddot{\mathbf{x}}_r(t) = \ddot{\mathbf{x}}_d(t) + k_d\dot{\tilde{\mathbf{x}}} + k_p\tilde{\mathbf{x}}(t)$, with k_p, k_d two positive gains, which have been set to 400 and 40, respectively. A secondary joint space task $-k_d\dot{\mathbf{q}}(t)$ (with k_d a positive gain set to 20) is also added to damp nullspace motions. The generated joint motion is shown in Fig. 4: compared to Fig. 2, the constraint on the maximum joint acceleration is also respected. In return, it is more difficult for the algorithm to find a feasible solution throughout the Cartesian trajectory and a task scaling smaller than one is observed more often (Fig. 5), compared to Fig. 3.

The torque-based solver uses the same reference Cartesian trajectory and joint bounds as the acceleration controller. A secondary task, this time defined as $-\mathbf{M}(k_d\dot{\mathbf{q}}(t))$, with k_d set to 20, is again included to damp nullspace motions. The choice $\mathbf{N} = \mathbf{M}^{-1}$ should produce the same joint motion as the acceleration-based solver. However, imprecise estimation of the complete robot dynamic model results in a different motion (Fig. 6); for the same reason, small inaccuracies can be noticed on the joint velocity saturation, as well as a slightly different trend of the task scaling factor (Fig. 7).

The Cartesian tracking error (Fig. 8) is also larger when the robot is torque-controlled. The other cases present a considerably smaller error, except when a task scaling $s < 1$ is observed.

B. Experiments with mobile dual-arm system

This experiment highlights the multitasking capability and the computational efficiency of the proposed algorithm. A redundancy resolution is computed online for a mobile dual-arm robot with 17 DOFs and three 3-dimensional prioritized tasks. A velocity-based solver is used with $\mathbf{N} = \mathbf{I}$. The first task is to move the omnidirectional mobile platform sideways, along the y-axis in Fig. 9, while keeping its x-position and its orientation about the z-axis constant. The

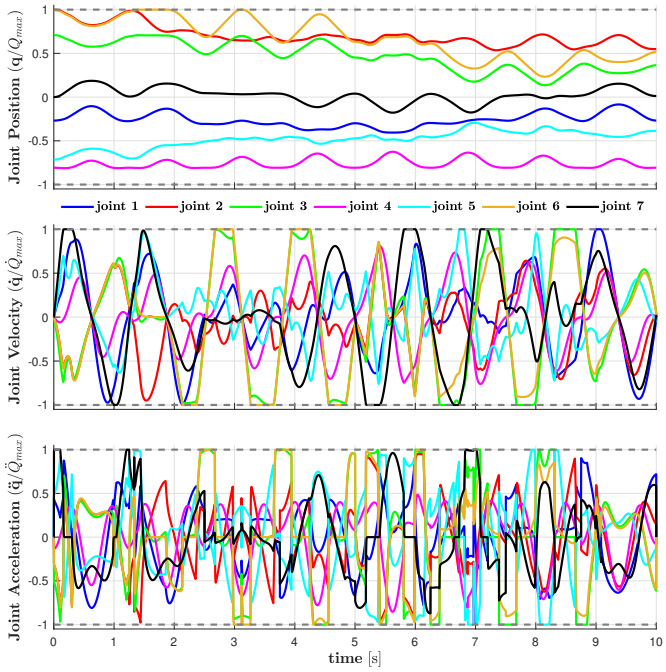

Fig. 4: Normalized joint position, velocity and acceleration produced by the acceleration-based solver for the LBRiiwa.

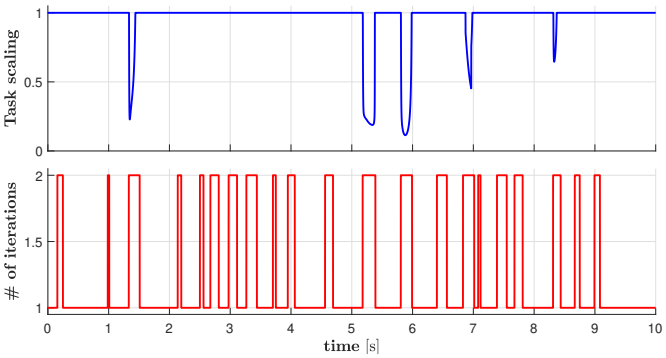

Fig. 5: Task scaling factor and number of iterations of the acceleration-based solver for the LBRiiwa.

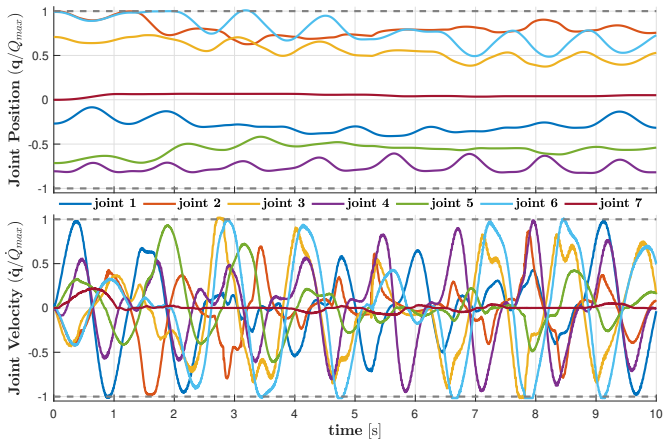

Fig. 6: Joint motion produced by the torque-based solver for the LBRiiwa; as a measure of the joint acceleration is not available, only (normalized) joint position and velocity are reported.

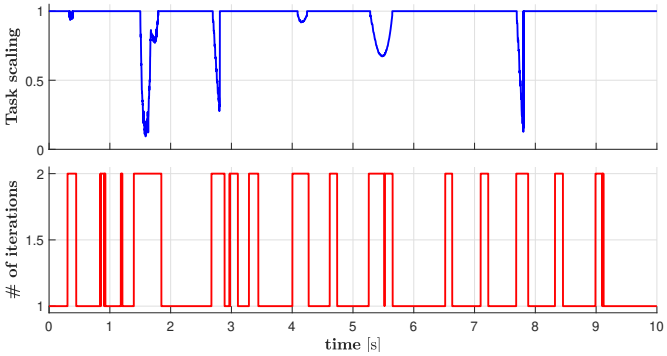

Fig. 7: Task scaling factor and number of iterations of the torque-based solver for the LBRiiwa.

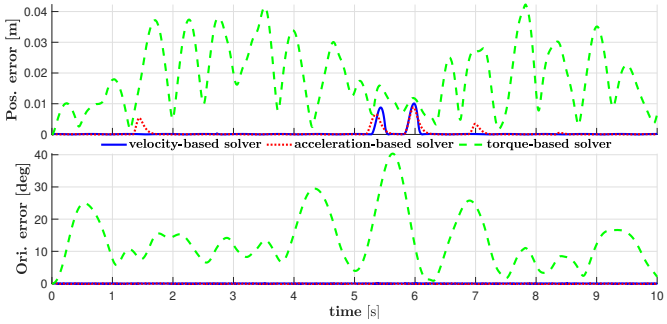

Fig. 8: Cartesian tracking error for the LBRiiwa; position error (in norm) is presented above; orientation error, reported as the angle extracted from the quaternion error, is reported below.

second and third tasks consist in keeping the end-effectors of both arms in place, controlling only their position. As can be seen in Fig. 11, the lowest priority task (i.e. task 3) is sacrificed first when it comes into conflict with the first task, followed by task 2. Finally, the robot comes to a halt when reaching a virtual limit in direction of the y-axis, which is modeled as a joint limit for one of the translational DOF of the mobile platform (joint 2 in Fig. 10). The computation time T required by the solver is also reported in the lower graph of Fig. 11 for a different number of total iterations. The computational resources are identical to the previous experiment.

V. Discussion

In this paper we have shown that both kinematic and torque control of redundant robots can be expressed in a single framework, as well as presented an efficient algorithm which exploits redundancy to fulfill a task under hard joint constraints. Experiment A shows the versatility and effectiveness in respecting joint limits of the solver, as well as the benefit of task scaling; the multitasking capability and computational efficiency in light of a high number of DOFs is proven in experiment B. Additionally, the experiments have hinted at the potential of using a weighted pseudoinverse to influence the generated joint motion according to the chosen metric. Some open issues regarding acceleration and torque

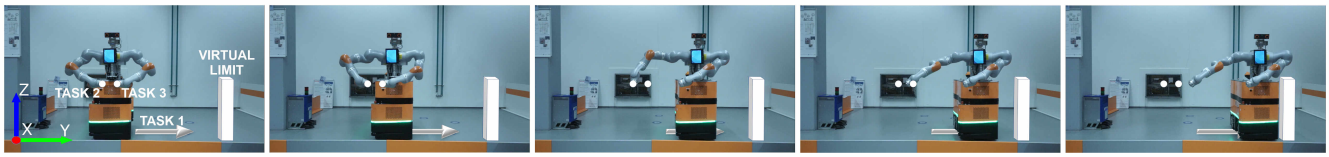

Fig. 9: Mobile dual-arm robot moving along the y-axis while keeping end-effector positions in place; the tasks are dropped in decreasing numerical order when they become unfeasible.

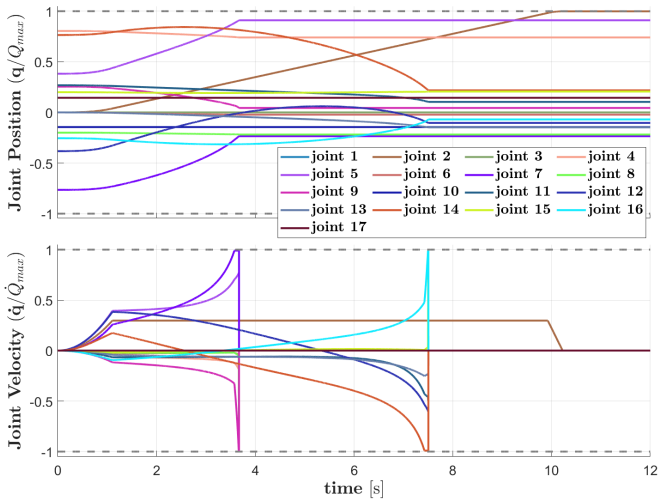

Fig. 10: Normalized joint position and velocity produced by the velocity-based solver for the mobile dual-arm robot.

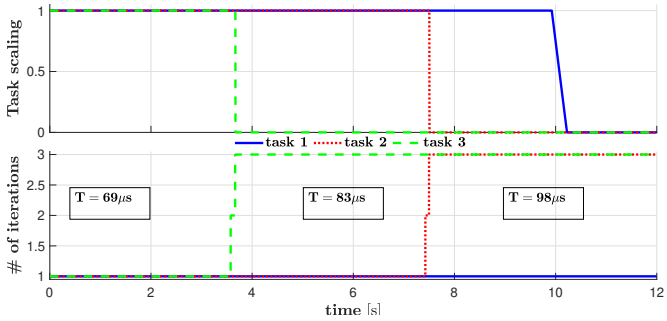

Fig. 11: Task scaling factor and number of iterations per task of the velocity-based solver for the mobile dual-arm robot. Execution times are given for 3, 4, and 7 total iterations

control remain, however. Foremost, the need for precise acceleration commands at joint limits makes saturation under torque control difficult. Additionally, task scaling alone is not sufficient for the robot to come to a full stop, as merely the acceleration is guaranteed to be scaled to zero; a joint-space damping task is a simple, but only partial solution to this problem. Finally, the used acceleration bounds present some drawbacks, which is also acknowledged in [17].

Nonetheless, the proposed controller presents a solid foundation on which different metrics and joint boundary computations can easily be tested.

References

[1] C. Scheurer, M.D. Fiore, S. Sharma and C. Natale, "Industrial implementation of a multi-task redundancy resolution at velocity level for highly redundant mobile manipulators," *Proceedings of ISR 2016: 47st International Symposium on Robotics* , pp. 109-117, 2016.

[2] B. Siciliano and J.-J. Slotine, "A general framework for managing multiple tasks in highly redundant robotic systems," *Fifth International Conference on Advanced Robotics 'Robots in Unstructured Environments*, pp. 1211–1216 vol.2, 1991.

[3] H. Sadeghian, L. Villani, M. Keshmiri, and B. Siciliano, "Dynamic multi-priority control in redundant robotic systems," *Robotica*, vol. 31, no. 07, pp. 1155–1167, 2013.

[4] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, pp. 500–505, 1985.

[5] A. Liégeois, "Automatic supervisory control of the configuration and behavior of multibody mechanisms," *IEEE Transactions on Man Machine Systems*, no. 12, pp. 868–871, 1977.

[6] A. Escande, N. Mansard, and P.-B. Wieber, "Hierarchical quadratic programming: Fast online humanoid-robot motion generation," *The International Journal of Robotics Research*, vol. 33, no. 7, pp. 1006–1028, 2014.

[7] O. Kanoun, "Real-time prioritized kinematic control under inequality constraints for redundant manipulators," *Robotics: Science and Systems 2011* , June 27-30, 2011.

[8] F. Flacco and A. de Luca, "Optimal redundancy resolution with task scaling under hard bounds in the robot joint space," in *2013 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3969–3975, 2013.

[9] J. M. Hollerbach, "Dynamic scaling of manipulator trajectories," in *1983 American Control Conference*, pp. 752–756, 1983.

[10] F. Flacco and A. de Luca, "Fast redundancy resolution for high-dimensional robots executing prioritized tasks under hard bounds in the joint space," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2500–2506, 2015.

[11] F. Flacco, A. de Luca, and O. Khatib, "Control of redundant robots under hard joint constraints: Saturation in the null space," *IEEE Transactions on Robotics*, vol. 31, no. 3, pp. 637–654, 2015.

[12] D. Whitney, "Resolved motion rate control of manipulators and human prostheses," *IEEE Transactions on Man Machine Systems*, vol. 10, no. 2, pp. 47–53, 1969.

[13] B. Siciliano, A. de Luca, and G. Oriolo, "Robot redundancy resolution at the acceleration level," *Laboratory Robotics and Automation*, pp. 97–106, 1992.

[14] O. Khatib, "Robot manipulator control in operational space," *Porc. CNRS Symposium on Mathematical Rools for Modeling and Control of Robots*, pp. 367–391, 1986.

[15] G.-r. Wang and Y.-l. Chen, "A recursive algorithm for computing the weighted moore-penrose inverse $a\_MN^{\dagger}$," *Journal of Computational Mathematics*, vol. 1, no. 4, pp. 74–85, 1986.

[16] A. Björck, *Numerical methods for least squares problems*. Philadelphia, Pa: Society for Industrial and Applied Mathematics (SIAM 3600 Market Street Floor 6 Philadelphia PA 19104), 1996.

[17] A. Del Prete, "Joint position and velocity bounds in discrete-time acceleration/torque control of robot manipulators," *IEEE Robotics and Automation Letters*, vol. 3, no. 1, pp. 281–288, 2018.