

Online approach to near time-optimal task-space trajectory planning

Antun Skuric¹, Nicolas Torres Alberto¹, Lucas Joseph¹, Vincent Padois¹ and David Daney¹

Abstract—Conforming to safety standards often limits collaborative robots’ performance and size, restricting their applications despite their capabilities. Planning their motions in human environments involves a trade-off between optimal trajectory planning and quick adaptation to dynamic, unstructured spaces. Traditional trajectory planning methods either use simplified robot models and sacrifice robot’s abilities for computational efficiency, or exploit robots’ abilities fully but have high computational complexity and rely on substantial pre-computation. This paper introduces an approach for trajectory planning that exploits robot’s full motion abilities while planning on-the-fly. In each step of the trajectory execution, it evaluates robot’s movement ability using polytope algebra and calculates a time-optimal Trapezoidal Acceleration Profile (TAP) on the remaining trajectory. The method is shown to be near time-optimal (around 5% slower trajectories) by benchmarking it against the state-of-the-art time-optimal method TOPP-RA. The method allows reaching higher velocities (able to plan up to 100% of the robot’s kinematic limits) while at the same time lowering the tracking error (under 4mm) than traditional Cartesian Space planning methods. A mock-up experiment demonstrates its efficiency in collaborative waste sorting using a Franka Emika Panda robot.

I. INTRODUCTION

The field of collaborative robotics has seen an unprecedented growth in recent years with the development of safer and cheaper robots with promising potential applications in industry, research and even everyday life [1]. However, as the complexity of the environments in which these robots operate increases, planning for robot motions becomes a significant challenge. The robots need to dynamically adapt to changing environmental conditions and tasks, as well as the presence of humans. At the same time, in order to be safe, collaborative robots tend to be smaller and relatively more limited in performance compared to more traditional industrial robots [2]. Due to these limitations, it is becoming increasingly important to utilise their physical abilities fully in order to make their real-world applications viable.

When it comes to planning robot motions, the most common approach is to decouple path and trajectory planning [3]. First, the robot’s geometric path is found, accomplishing certain task and potentially avoiding the obstacles in the environment. Then the optimal sequence of movements along this path is calculated in order to optimise certain criteria, the most common ones used in the literature being minimum energy, minimum jerk and minimum execution time. Among these criteria, the one that aims to fully exploit the robot’s movement

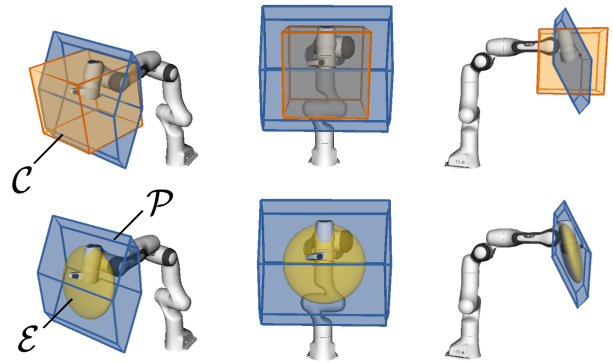


Fig. 1. End-effector linear velocity limits as computed using the fixed CS limits provided by the manufacturer $C = \{\dot{\mathbf{x}} \in \mathbb{R}^3 \mid |\dot{\mathbf{x}}| \leq \dot{\mathbf{x}}_{max}\}$ (orange) and using the JS limits and the current configuration of the robot, represented by a polytope $\mathcal{P} = \{\dot{\mathbf{x}} \in \mathbb{R}^3 \mid \dot{\mathbf{x}} = J(\mathbf{q})\dot{\mathbf{q}}, \|\dot{\mathbf{q}}\|_2 \leq \dot{\mathbf{q}}_{max}\}$ (blue) and a manipulability ellipsoid $\mathcal{E} = \{\dot{\mathbf{x}} \in \mathbb{R}^3 \mid \dot{\mathbf{x}} = J(\mathbf{q})\dot{\mathbf{q}}, \|\dot{\mathbf{q}}\|_2 \leq \dot{\mathbf{q}}_{max}\}$ (yellow). While ellipsoid \mathcal{E} represents a considerable under-estimation of the true robot’s velocity capacity \mathcal{P} , the fixed CS limits C are under-estimating in certain directions while significantly over-estimating in others. Both JS and CS limits are in Table I and the visualisation scale is: 1 m.s⁻¹/10 cm.. The figure shows multiple views (side, front and left) for one single JS configuration \mathbf{q}

capacity corresponds to the minimum time criteria, resulting in time-optimal trajectories [4].

Traditional time-optimal trajectory planning techniques, developed for industrial robots, find trajectories with the highest possible speeds, within the robot’s and task constraints, along predefined paths. These methods are often defined in the robot’s Joint Space (JS) calculating the necessary optimal motions of each one of the robot’s joints. Time-optimal algorithms, such as the ones proposed by Bobrow et al. [5] and more recently by Pham and Pham [6], assume full in advance knowledge about both the robot’s path and the environment. In collaborative scenarios, the environment is often dynamic and conditions may change rapidly, requiring quick changes in robot’s trajectory and task. Traditional time-optimal approaches have long execution times, due to their computational complexity, which, in many cases, makes their use unpractical.

On the other end of the spectrum are the task space planning methods. They often abstract the robot as an ideal generator of Cartesian Space (CS) movements (positions and orientations) without accounting for its true motion capacity. With this simplification, they have short computation times and can even adapt to the real-time changes in the robot’s path, environment, or the task itself. Such algorithms, as proposed by Macfarlane and Croft [7], Haschke et al. [8], Kröger and Wahl [9] or Svorny et al. [10], are often based

¹AUCTUS Team, Inria, Talence, France
firstname.lastname@inria.fr

This work is partly supported by BPI France through the LICHIE project in collaboration with Airbus.

on S-curve or trapezoidal profile trajectories [11], producing smooth trajectories while assuming the robot's CS limits to be constant [12]. Yet, even though the robot's JS actuator limits are constant, once transformed to CS they become dependent on the robot state and can therefore vary significantly during the trajectory execution [13]. Therefore, such approaches are not capable of fully exploiting the robot's capacity, resulting in trajectories that might underestimate (unnecessary slow movements) or sometimes even overestimate them (resulting in large tracking errors). Figure 1 shows an example of this effect on a Franka Emika Panda collaborative robot.

This paper proposed a new trajectory planning method that plans entirely in CS, while at the same time allows fully exploiting robot's state dependant movement abilities. The method is based on real-time replanning and at each time-step of the trajectory execution, the proposed approach 1) evaluates the robot's CS movement capacity limits, proposing an efficient polytope algebra approach inspired by [14, III.A] 2) uses these limits to re-plan the time-optimal trajectory on the remaining path, based on the computationally efficient Trapezoidal Acceleration Profile (TAP) [15]. By continuously evaluating the robot's CS movement capacity and re-planning the time-optimal trajectory, the approach is capable of adapting to the changes in the robot's movement capacity entirely eliminating the need for any pre-computation. Additionally, as the method plans in CS it remains independent of the underlying robot controller (even supporting Cartesian controllers), facilitating its deployment with broader range of robotic platforms.

To evaluate the near-time optimal performance of the proposed method, it is compared against the state-of-the-art offline method TOPP-RA [6], which plans time-optimal trajectories in JS, as well as against a traditional CS methods, based on TAP planning with constant CS limits. The experimental results demonstrate that the proposed approach produces trajectories that are, on average, only around 5% slower than the TOPP-RA's, while achieving higher velocities and lower tracking errors (under 4mm) compared to traditional CS planning methods. Furthermore, the paper showcases the practical applicability of the proposed approach through a mock-up experiment of collaborative waste sorting, where the method generates near time-optimal trajectories on-the-fly.

In this paper, the intuition about the robot's movement capacity aware trajectory planning in CS is given in Section II. The description of the method for the robot's CS movement capacity evaluation is described in Section III. Then, Section IV gives an overview of the CS TAP planning and the description of the real-time planning strategy. Section V discusses several potentially undesirable effects of the proposed method and proposes the approaches to overcome them. The experimental validation of the proposed method is addressed in Section VII and Section VIII. Finally, the limitations and perspectives of the proposed method are discussed in Section IX.

II. ABILITY-AWARE TRAJECTORY PLANNING: CHALLENGES AND PROPOSED APPROACH

Time-optimal trajectory planning consists of finding the robot's motions with the highest possible speeds along prede-

defined paths, while respecting the robot's inherent motion ability and potentially user-defined task constraints. The robot's motions are generated by its n joint actuators, each with their physical limitations in available torques, velocities, accelerations, etc. As opposed to the robot's actuator limits which are specified in the robot's Joint Space (JS), the desired path to follow is often defined in the task space. The task space is often the 3-dimensional space of Cartesian Space (CS) positions ($m = 3$) or 6-dimensional if orientations are specified as well ($m = 6$). Therefore, one of the key challenges to time-optimal planning is the quantification of the robot's motion ability limits (ex. velocity, acceleration, jerk) along the path.

In the context of this paper, the paths are considered to be CS point-to-point straight lines from one pose X_a to another X_b (for example expressed as an homogeneous transform matrix in $SE(3)$). CS straight line paths are commonly used as they represent the shortest paths in the CS between the two CS positions, where many robotics tasks can be represented as a series of straight line motions (ex. pick-and-place). Additionally, different more complex paths can be approximated using a number of straight line segments [16][17][18].

A. Problem statement

Traditional approaches to the time-optimal trajectory planning consist in transforming the task space path to the JS, where they finese methods are developed for the special cased the fastest movements of the robot's joints following the path and respecting the actuator limits [19]. The robot's JS actuator limits (position, velocity, acceleration, jerk, etc.), are provided by manufacturers in advance, often in the form of n independent min-max ranges, and can reasonably be considered constant in time. Therefore, these constant and independent limits allow for efficient time-optimal trajectory planning for any pre-known JS path.

The significant part of the computational cost of these methods comes from the transformation of the CS path to the JS which is typically pre-computed. Additionally, when it comes to redundant robots, for which the JS is higher dimensional than the CS ($n > m$), there are multiple JS paths (potentially infinite number) that accomplish the same CS path. In that case, finding the optimal JS path for any given CS path is a challenging problem on its own and one with a substantial computational cost [20][21]. Finally, as the JS path is pre-computed, robot's redundant degrees of freedom are fixed in-advance, even though they do not participate in the movement generation and are no longer available to execute other tasks.

A complementary approach to the traditional JS planning, consists in transforming the robot's JS actuator limits onto the CS path and planning the time-optimal trajectory directly in CS [13][22]. Such methods ideally result in CS trajectories which respect all the robot's JS actuator limits without explicitly choosing a JS path. The robot's actuator limits, transformed to CS form movement capacity polytopes [23][24][25], often approximated by manipulability ellipsoids [26] (see Figure 1). As illustrated on Figure 2, these polytopes are highly dependant on the robot's state and can change significantly

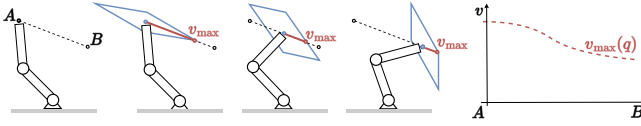


Fig. 2. Illustrations of the evolution of the CS linear velocity capacity while following a CS straight line path, from point A to B. The robot's CS velocity capacity polytope is shown in blue, while its capacity to generate velocity in the path direction is shown in red. The path is shown in black dashed lines. The graph on the right shows the evolution of the robot's velocity capacity in the path direction while executing the trajectory.

during the execution of the trajectory, presenting one of the main challenges of time-optimal trajectory planning in CS.

Most of the traditional CS planning methods rely on the constant set of robot's movement limits [7][8][10]. However, choosing any fixed set of CS limits leads to either overestimation of the robot's movement capacity or its underestimation, as shown on Figure 3. The overestimation can lead to planning for trajectories that are unfeasible for the robot, while underestimation leads to sub-optimal trajectories that do not fully exploit its movement potential. Planning for a true time-optimal trajectory would involve accounting for the robot's changing movement abilities along the trajectory. However, characterising the robot's state-dependant movement limits along the path in advance would require transforming the CS path to JS and result in high computational cost. It would also require making choices about the appropriate JS path and ultimately fixing the robot's redundant degrees of freedom, therefore losing most of the benefits over the planning in JS.

Rather than trying to account for the robot's state-dependant movement capacity in the offline planning stage, a set of online velocity scaling methods was proposed, such as Bianco and Ghilardelli [13]. These methods scale the robot's movement velocity on the path with respect to its changing movement ability, enhancing the feasibility of trajectories and maintain the accuracy of path following. However, these methods are developed for the special case of non-redundant robotic manipulators ($n = m$), where the transition from JS to CS and *vice-versa* can be done efficiently and has a finite number of solutions (except in singular configurations). Such methods are not directly applicable to the commonly used collaborative robots often being redundant manipulators [27].

B. Proposed approach

Partially inspired by the work of Bianco and Ghilardelli [13], this paper presents a new online method for CS trajectory planning capable of planing for both redundant and non-redundant manipulators.

At each step of the trajectory execution the proposed method first evaluates the instantaneous robot's movement ability in the path direction (physical ability to generate movement: velocity, acceleration and jerk), proposing an efficient polytope algebra based method. Then, these limits are considered constant and used to update the time-optimal trajectory on the remaining path using Trapezoidal Acceleration Profile (TAP) algorithm [15]. The combined computational efficiency of the new evaluation method (described in Section III) in combination with TAP planning (described in Section IV)

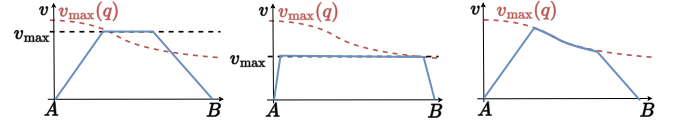


Fig. 3. Comparison of three trajectory planning strategies for CS, considering the robot's velocity capacity limits. Left and middle figures depict methods assuming a fixed velocity capacity, leading to overestimation (left) or significant underestimation (middle) of the robot's abilities. The right figure illustrates a time-optimal trajectory that fully exploits the robot's changing velocity capacity. Blue lines represent planned trajectories, while red dashed lines indicate the actual robot velocity capacity during execution.

allows for sub-millisecond execution, enabling it to quickly adapt to the changes in the robot's abilities while following the path. Figure 4 depicts the intuition of the proposed method.

III. EVALUATING THE ROBOT'S CARTESIAN SPACE MOVEMENT CAPACITY IN REAL-TIME

The robot's joint actuator limits are usually given by the manufacturer as ranges¹ of the feasible joint positions $\mathbf{q} \in \mathbb{R}^n$, velocity $\dot{\mathbf{q}} \in \mathbb{R}^n$, acceleration $\ddot{\mathbf{q}} \in \mathbb{R}^n$ and jerk $\dddot{\mathbf{q}} \in \mathbb{R}^n$

$$\begin{aligned} \ddot{\mathbf{q}} &\in [\ddot{\mathbf{q}}_{min}, \ddot{\mathbf{q}}_{max}], \quad \dot{\mathbf{q}} \in [\dot{\mathbf{q}}_{min}, \dot{\mathbf{q}}_{max}], \\ \mathbf{q} &\in [\mathbf{q}_{min}, \mathbf{q}_{max}] \end{aligned} \quad (1)$$

The mapping between the robot's n dimensional JS velocity $\dot{\mathbf{q}}$, acceleration $\ddot{\mathbf{q}}$ and jerk $\dddot{\mathbf{q}}$ and its m dimensional Cartesian Space (CS) equivalents $\dot{\mathbf{x}}, \ddot{\mathbf{x}}, \dddot{\mathbf{x}}$, for a certain fixed frame of interest (ex. end-effector frame), is nonlinear and dependant on the robot's state $\{\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}\}$

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{J}(\mathbf{q})\dot{\mathbf{q}} \\ \ddot{\mathbf{x}} &= \mathbf{J}(\mathbf{q})\ddot{\mathbf{q}} + \dot{\mathbf{J}}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} \\ \dddot{\mathbf{x}} &= \mathbf{J}(\mathbf{q})\dddot{\mathbf{q}} + 2\dot{\mathbf{J}}(\mathbf{q}, \dot{\mathbf{q}})\ddot{\mathbf{q}} + \ddot{\mathbf{J}}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})\dot{\mathbf{q}} \end{aligned} \quad (2)$$

Where $\mathbf{J}(\mathbf{q}) \in \mathbb{R}^{m \times n}$ is the Jacobian matrix and $\dot{\mathbf{J}}(\mathbf{q}, \dot{\mathbf{q}}) \in \mathbb{R}^{m \times n}$ and $\ddot{\mathbf{J}}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) \in \mathbb{R}^{m \times n}$ are its time derivatives.

JS kinematic limits (1) are expressed in a form of an interval for each of the robot's n joints (degrees of freedom), forming n dimensional hyperrectangles. For any given robot

¹The true Joint Space (JS) acceleration $\ddot{\mathbf{q}}$ and jerk $\dddot{\mathbf{q}}$ limits are a consequence of the constrained nature of the robot's actuator torques $\boldsymbol{\tau}$ and torque derivatives $\dot{\boldsymbol{\tau}}$ [28], and have a polytope form. In this work, these limits are considered to be given in a form of independent ranges (1). However, the proposed approach is not restricted by this assumption and can be extended to account for the polytope limits as well.

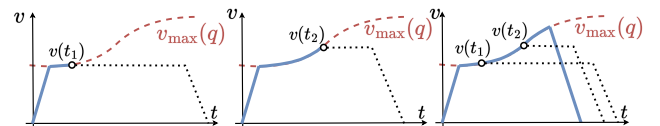


Fig. 4. Velocity curve, updated in the real time with instantaneous maximal values. At each time-step t_k during the trajectory execution (ex. t_1 and t_2) the TAP planning calculates the remaining trajectory considering that the maximal velocity \dot{v}_{max} is constant till the end of the movement (black dotted lines). As the maximal velocity \dot{v}_{max} increases during the trajectory execution (red dashed line), the final trajectory duration is considerably shorter than planned in moments t_1 and t_2 .

state $\{\mathbf{q}_k, \dot{\mathbf{q}}_k, \ddot{\mathbf{q}}_k\}$, CS kinematic limits can be calculated by projecting these JS n dimensional hyperrectangles (1) into the m dimensional CS using the expressions (2). The resulting CS limits will have the form of convex polytopes.

For a certain robot state $\{\mathbf{q}_k, \dot{\mathbf{q}}_k, \ddot{\mathbf{q}}_k\}$ the convex polytopes \mathcal{P}_v , \mathcal{P}_a and \mathcal{P}_j of achievable CS velocities $\dot{\mathbf{x}}$, accelerations $\ddot{\mathbf{x}}$ and jerks $\dddot{\mathbf{x}}$ can be expressed as

$$\mathcal{P}_v = \{\dot{\mathbf{x}} \in \mathbb{R}^m \mid \dot{\mathbf{x}} = J(\mathbf{q}_k)\dot{\mathbf{q}} + \mathbf{b}_v, \dot{\mathbf{q}} \in [\dot{\mathbf{q}}_{min}, \dot{\mathbf{q}}_{max}]\} \quad (3a)$$

$$\mathcal{P}_a = \{\ddot{\mathbf{x}} \in \mathbb{R}^m \mid \ddot{\mathbf{x}} = J(\mathbf{q}_k)\ddot{\mathbf{q}} + \mathbf{b}_a, \ddot{\mathbf{q}} \in [\ddot{\mathbf{q}}_{min}, \ddot{\mathbf{q}}_{max}]\} \quad (3b)$$

$$\mathcal{P}_j = \{\dddot{\mathbf{x}} \in \mathbb{R}^m \mid \dddot{\mathbf{x}} = J(\mathbf{q}_k)\dddot{\mathbf{q}} + \mathbf{b}_j, \dddot{\mathbf{q}} \in [\dddot{\mathbf{q}}_{min}, \dddot{\mathbf{q}}_{max}]\} \quad (3c)$$

where $\mathbf{b}_a, \mathbf{b}_j \in \mathbb{R}^m$ are the bias acceleration and jerk² produced by the effect of current joint velocity $\dot{\mathbf{q}}_k$ and acceleration $\ddot{\mathbf{q}}_k$, while $\mathbf{b}_v \in \mathbb{R}^m$ is a zero vector ($\mathbf{b}_v = \mathbf{0}$), added to unify the formulations.

$$\begin{aligned} \mathbf{b}_a &= \dot{J}(\mathbf{q}_k, \dot{\mathbf{q}}_k)\dot{\mathbf{q}}_k \\ \mathbf{b}_j &= 2\dot{J}(\mathbf{q}_k, \dot{\mathbf{q}}_k)\ddot{\mathbf{q}}_k + \ddot{J}(\mathbf{q}_k, \dot{\mathbf{q}}_k, \ddot{\mathbf{q}}_k)\dot{\mathbf{q}}_k \end{aligned} \quad (4)$$

Finally, the achievable CS velocity, acceleration and jerk, given the current robot state $\{\mathbf{q}_k, \dot{\mathbf{q}}_k, \ddot{\mathbf{q}}_k\}$ can be expressed as

$$\dot{\mathbf{x}} \in \mathcal{P}_v(\mathbf{q}_k), \quad \ddot{\mathbf{x}} \in \mathcal{P}_a(\mathbf{q}_k, \dot{\mathbf{q}}_k), \quad \dddot{\mathbf{x}} \in \mathcal{P}_j(\mathbf{q}_k, \dot{\mathbf{q}}_k, \ddot{\mathbf{q}}_k) \quad (5)$$

As point-to-point CS paths are effectively one dimensional, instead of using the complete polytope definitions (3a - 3c), a more efficient solution is to use the polytopes to find the robot's movement capacity in the path direction. The calculated movement capacity can be expressed in the form or min-max ranges (intervals) for each one of the CS variables. The following section proposes an efficient approach to finding the projection of the CS polytopes of the robot's movement capacity in the path direction.

A. Finding movement capacity in the path direction

Characterising the robot's movement (velocity, acceleration and jerk) capacity, along the straight line path, requires finding the maximal (and minimal) values of the robot's CS movement variables in the path direction that are still within their respective polytopes (5). In other words, geometrically, this corresponds to finding the intersection of the polytope with the line corresponding to the path direction. This section proposes an efficient Linear Program (LP) based approach for characterising these intersections, inspired by [14, Sec. III.A], enabling to evaluate robot's movement capacity in each step of trajectory execution.

For any point-to-point CS path, an m dimensional CS unit vector \mathbf{c} , pointing in the direction of the path can be found. Then any CS vector $\mathbf{y} \in \mathbb{R}^m$ can be projected on the path direction using the scalar product

$$s = \mathbf{c}^T \mathbf{y} \quad (6)$$

where $s \in \mathbb{R}$ is a scalar. Analogously, the projection of the vector \mathbf{y} on the path's complementary (orthogonal) space can be expressed using [29, p. 431]

$$\mathbf{y}_\perp = (I_{m \times m} - \mathbf{c}\mathbf{c}^T)\mathbf{y} \quad (7)$$

²The term $\ddot{J}\dot{\mathbf{q}}_k$ in \mathbf{b}_j can often be neglected due to its minimal impact on the bias term ($J\dot{\mathbf{q}}_k \approx 0$). This simplification is beneficial for computational efficiency or when obtaining the second derivative \ddot{J} is challenging.

where $\mathbf{y}_\perp \in \mathbb{R}^m$ is a m dimensional vector containing the components of the vector \mathbf{y} orthogonal to the path. Mathematically, the matrix $N = (I_{m \times m} - \mathbf{c}\mathbf{c}^T)$ corresponds to the projector to the null-space $\mathcal{Ker}(\mathbf{c}^T)$ of the vector \mathbf{c} . A simple check can then be devised to verify if a certain CS vector \mathbf{y} is in the path direction, by verifying that it does not have components in the path null-space, or in other words $N\mathbf{y} = \mathbf{0}$.

With the known path direction \mathbf{c} and the path normal space projector N , the maximal value of the CS variable $\mathbf{y} \in \mathbb{R}^m$, in the path direction \mathbf{c} , within its range expressed as a convex polytope \mathcal{P}_y , can be found by solving the LP [30] [14, III.A]

$$\begin{aligned} \max_{\mathbf{y}} \quad & \mathbf{c}^T \mathbf{y} \\ \text{s.t.} \quad & N\mathbf{y} = \mathbf{0}, \\ & \mathbf{y} \in \mathcal{P}_y \end{aligned} \quad (8)$$

whereas the minimum can be found by minimising it.

By substituting the generic CS variable \mathbf{y} with CS velocity $\dot{\mathbf{x}}$, acceleration $\ddot{\mathbf{x}}$ and jerk $\dddot{\mathbf{x}}$ and its polytope \mathcal{P}_y with their respective polytopical limits (5), LP formulation (8) can be directly used to calculate the limits of the velocity \dot{s} , acceleration \ddot{s} and jerk \ddot{s} in the trajectory direction \mathbf{c} .

Cartesian velocity example: When searching for maximal CS velocity \dot{s}_{max} , assuming known robot configuration \mathbf{q}_k and trajectory direction vector \mathbf{c} , the CS velocity along the trajectory \dot{s} can be calculated as a projection of the velocity $\dot{\mathbf{x}}$

$$\dot{s} = \mathbf{c}^T \dot{\mathbf{x}} = \mathbf{c}^T J(\mathbf{q}_k)\dot{\mathbf{q}} + \mathbf{c}^T \mathbf{b}_v \quad (9)$$

Integrating this relationship into the LP formulation (8) yields

$$\begin{aligned} \dot{s}_{max} &= \max_{\dot{\mathbf{q}}} \mathbf{c}^T J(\mathbf{q}_k)\dot{\mathbf{q}} + \mathbf{c}^T \mathbf{b}_v \\ \text{s.t.} \quad & NJ(\mathbf{q}_k)\dot{\mathbf{q}} = -N\mathbf{b}_v, \\ & \dot{\mathbf{q}} \in [\dot{\mathbf{q}}_{min}, \dot{\mathbf{q}}_{max}] \end{aligned} \quad (10)$$

The equivalent LP expressions for finding the maximal acceleration and jerk are obtained by substituting $\dot{\mathbf{q}}$ and \mathbf{b}_v with $\ddot{\mathbf{q}}$, \mathbf{b}_a and $\ddot{\mathbf{q}}$, \mathbf{b}_j . The minimal values are found by minimising the same problem. Using the proposed procedure, for any path direction \mathbf{c} and a given robot state $\{\mathbf{q}_k, \dot{\mathbf{q}}_k, \ddot{\mathbf{q}}_k\}$, the robot's movement capacities (velocity \dot{s} , acceleration \ddot{s} and jerk \ddot{s}) in the path direction can be expressed as

$$\begin{aligned} \dot{s} &\in [\dot{s}_{min}(\mathbf{q}_k), \dot{s}_{max}(\mathbf{q}_k)] \\ \ddot{s} &\in [\ddot{s}_{min}(\mathbf{q}_k, \dot{\mathbf{q}}_k), \ddot{s}_{max}(\mathbf{q}_k, \dot{\mathbf{q}}_k)] \\ \ddot{s} &\in [\ddot{s}_{min}(\mathbf{q}_k, \dot{\mathbf{q}}_k, \ddot{\mathbf{q}}_k), \ddot{s}_{max}(\mathbf{q}_k, \dot{\mathbf{q}}_k, \ddot{\mathbf{q}}_k)] \end{aligned} \quad (11)$$

Therefore, the robot's instantaneous movement capacity in the path direction can be obtained by solving a sequence of 6 LP problems per path direction³ (ex. translation and rotation), where the computational efficiency of LP solvers allows for real-time execution [31].

The proposed method recalculates the set of limits (11) in each time-step and uses Trapezoidal Acceleration Profile (TAP) to find the trajectory that exploits them fully on the

³In practice, as the velocity bias $\mathbf{b}_v = \mathbf{0}$ and the JS velocity limits are often symmetric $\dot{\mathbf{q}}_{max} = -\dot{\mathbf{q}}_{min}$, the path velocity limits will be symmetric as well $\dot{s}_{max} = -\dot{s}_{min}$. In this case, the number of LPs is reduced to 5 per direction.

remaining path. Evaluating the limits (11) in each time-step allows for assuming each variable to be independent from one another, as any interdependence will be visible in their real-time changes. Additionally, the use of a TAP planning strategy guarantees that these limits are used in a compatible way [32] (ex. when \dot{s}_{max} is reached on the planned profile, accelerating is no longer possible $\ddot{s}_{max}=0$).

B. Considering task induced Cartesian space limits

In many cases, when planning for CS movement, the task requires limiting maximal CS velocity, acceleration and jerk. For example, when executing trajectories in the human vicinity, it is common practice to limit the robot's CS velocity $\dot{\mathbf{x}}$ to minimise the potential impact forces due to the robot's kinetic energy [2][33]. On the other hand, when designing trajectories of robotic manipulation of liquids, in order to prevent its sloshing or spilling, acceleration $\ddot{\mathbf{x}}$ continuity needs to be ensured [34], which in term corresponds to introducing CS jerk $\ddot{\mathbf{x}}$ limits.

Therefore, if an additional task specific limit of a CS variable \mathbf{y} (velocity, acceleration or jerk) is required that can be expressed as an interval $\mathbf{y} \in [\mathbf{y}_{min}, \mathbf{y}_{max}]$ or more generally in a form of polytope \mathcal{C} defined by a set of inequality constraints

$$\mathcal{C} = \{\mathbf{y} \mid A\mathbf{y} \leq \mathbf{b}\} \quad (12)$$

Then the LP problem (8) can be extended to account for the polytope (12)

$$\begin{aligned} \max_{\mathbf{y}} \quad & c^T \mathbf{y} \\ \text{s.t.} \quad & N\mathbf{y} = \mathbf{0}, \\ & A\mathbf{y} \leq \mathbf{b}, \\ & \mathbf{y} \in \mathcal{P}_y \end{aligned} \quad (13)$$

C. Scaling the robot's Cartesian space limits

In the industry it is a common practice to use only a part of the robot's motion capabilities when planing for its trajectories. This is partially due to the fact that more dynamic robot movements require using large part of the robot's actuation capacity, and leave less margin to account for potential tracking error which can lead to impaired tracking performance and potentially even raise different safety concerns. Most of the standard teach-pendant allow the operator to set the desired speed of the robot's movements by specifying the ratio of the robot's limits used [35][36].

When the robot's CS kinematic limits are assumed constant, the simplest form of scaling can be done by multiplying with scalar factors $\alpha_v, \alpha_a, \alpha_j \in [0, 1]$.

$$\begin{aligned} \dot{\mathbf{x}} &\in [\alpha_v \dot{\mathbf{x}}_{min}, \alpha_v \dot{\mathbf{x}}_{max}], \quad \ddot{\mathbf{x}} \in [\alpha_a \ddot{\mathbf{x}}_{min}, \alpha_a \ddot{\mathbf{x}}_{max}], \\ \ddot{\mathbf{x}} &\in [\alpha_j \ddot{\mathbf{x}}_{min}, \alpha_j \ddot{\mathbf{x}}_{max}] \end{aligned} \quad (14)$$

allowing the robots velocity $\dot{\mathbf{x}}$, acceleration $\ddot{\mathbf{x}}$ and jerk $\ddot{\mathbf{x}}$ not to exceed certain percentage of the specified limits.

However, if the robot's CS kinematic capacity is not considered constant, but a result of the robot's current state $\{\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}\}$,

and its JS kinematic limits (1), then the scaling using the scalars $\alpha_v, \alpha_a, \alpha_j \in [0, 1]$ can be done in the JS

$$\begin{aligned} \dot{\mathbf{q}} &\in [\alpha_v \dot{\mathbf{q}}_{min}, \alpha_v \dot{\mathbf{q}}_{max}], \quad \ddot{\mathbf{q}} \in [\alpha_a \ddot{\mathbf{q}}_{min}, \alpha_a \ddot{\mathbf{q}}_{max}], \\ \ddot{\mathbf{q}} &\in [\alpha_j \ddot{\mathbf{q}}_{min}, \alpha_j \ddot{\mathbf{q}}_{max}] \end{aligned} \quad (15)$$

These new modulated JS limits indirectly scale the robot's CS movement capacity by modifying the size of the polytopes limits (3a - 3c).

Both scaling strategies are compatible with the proposed approach and can be leveraged to achieve appropriate robot's CS motion in different scenarios.

IV. EVOLVING CAPACITY AWARE CARTESIAN SPACE TRAJECTORY PLANNING

In order to fully exploit the robot's changing movement capacity while executing the trajectory, this work proposes a real-time re-planning strategy. The proposed trajectory planning method leverages the computational efficiency of Trapezoidal Acceleration Profile (TAP) or S-curve velocity profiles [12, Chapter 9.2.2.2][37][15], a classical approach to finding the Cartesian Space (CS) time-optimal or minimum-time trajectories [4]. In each step of the trajectory execution, the proposed method evaluates the robot's movement capacity in the path direction, using the efficient method described in Section III-A, and recalculates the new time-optimal trajectory using TAP planning on the remaining path.

Section IV-A brings a brief introduction to the basics of the TAP planning, while Section IV-B provides more details about the real-time re-planning approach.

A. Trapezoidal acceleration profile basics

CS point-to-point straight-line paths connect two CS poses X_a and X_b (for example expressed as homogeneous matrices in $SE(3)$) of the desired robot's frame (ex. end-effector frame) [12, Chapter 9.2.1]. These paths can be expressed as $\mathcal{P}(s)$, where $s \in [0, d]$ is a scalar position on the path with a length d [38][39].

When it comes to finding the time-optimal evolution of the robot's CS pose $X(t)$ following the straight line path between X_a and X_b , one of the most common approaches in the literature are the TAP planning techniques [12, Chapter 9.2.2.2]. TAP trajectory planning finds the time-optimal evolution of the path position variable $s(t)$ respecting the limits on all its derivatives

$$\dot{s} \in [\dot{s}_{min}, \dot{s}_{max}], \quad \ddot{s} \in [\ddot{s}_{min}, \ddot{s}_{max}], \quad \ddot{\ddot{s}} \in [\ddot{\ddot{s}}_{min}, \ddot{\ddot{s}}_{max}] \quad (16)$$

as well as the starting and end conditions

$$\dot{s}(0) = \dot{s}_0, \quad \dot{s}(T) = \dot{s}_T, \quad \ddot{s}(0) = \ddot{s}_0, \quad \ddot{s}(T) = \ddot{s}_T \quad (17)$$

T is the trajectory duration, \dot{s}_0 and \ddot{s}_0 represent the velocity and acceleration at the beginning of the path $\mathcal{P}(s)$, while \dot{s}_T and \ddot{s}_T represent their final values at the end of the trajectory.

When planing for the geometric paths $\mathcal{P}(s)$ in CS, from the CS pose X_a to X_b , it is common practice to separate the translation $\mathcal{P}_t(s_t)$ and orientation $\mathcal{P}_r(s_r)$ component of

the path $\mathcal{P}(s)$ [19]. The translation component of the path can then be expressed as

$$\mathcal{P}_t(s_t) = s_t \mathbf{u}_t, \quad s_t \in [0, \|X_b^t - X_a^t\|_2] \quad (18)$$

where X_a^t and X_b^t are the translation parts of the poses X_a and X_b and \mathbf{u}_t is the unit vector pointing from X_a^t to X_b^t . For the orientation, the common approach is to use the axis-angle representation of the rotation, and specify the difference in orientation between X_a and X_b as an angle θ around the rotation axis \mathbf{u}_r . Then the geometric path corresponding to the orientation $\mathcal{P}_o(s_r)$ can be written as

$$\mathcal{P}_r(s_r) = s_r \mathbf{u}_r, \quad s_r \in [0, \theta] \quad (19)$$

$\mathcal{P}_r(s_r)$ and $\mathcal{P}_t(s_t)$ represent a relative change in the orientation and translation over the course of the trajectory from the initial pose X_a . The desired CS pose $X(t)$ can be calculated using an homogeneous transformation matrix H , constructed from \mathcal{P}_r and \mathcal{P}_t

$$X(t) = X_a H(\mathcal{P}_r(s_r(t)), \mathcal{P}_t(s_t(t))) \quad (20)$$

and optimal CS velocity and acceleration can be found

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} \dot{s}_t(t) \mathbf{u}_t \\ \dot{s}_r(t) \mathbf{u}_r \end{bmatrix}, \quad \ddot{\mathbf{x}}(t) = \begin{bmatrix} \ddot{s}_t(t) \mathbf{u}_t \\ \ddot{s}_r(t) \mathbf{u}_r \end{bmatrix} \quad (21)$$

Conversely, for any given set of CS variables $\{\dot{\mathbf{x}}, \ddot{\mathbf{x}}\}$, their projection to the scalar path variables $\{\dot{s}_i, \ddot{s}_i\}$

$$\dot{s}_i = \mathbf{c}_i^T \dot{\mathbf{x}}, \quad \ddot{s}_i = \mathbf{c}_i^T \ddot{\mathbf{x}}, \quad i = \{r, t\} \quad (22)$$

can be expressed through the translation path $\mathcal{P}_t(s_t)$ and the orientation path $\mathcal{P}_r(s_r)$ direction vectors

$$\mathbf{c}_t = [\mathbf{u}_t^T, \mathbf{0}_{3 \times 1}]^T, \quad \mathbf{c}_r = [\mathbf{0}_{3 \times 1}, \mathbf{u}_r^T]^T \quad (23)$$

Once both translation and orientation TAP trajectories are found resulting in optimal $s_r(t)$ and $s_t(t)$, to synchronise the two movements, their time duration is matched, making both trajectories last the same time T , the time taken by the longer of the two trajectories $T = \max\{T_r, T_t\}$.

The inherent challenge of TAP planning for the robot's motion in the CS is that its movement capacity limits (16) are robot's state dependent, and over the course of the trajectory the robot's movement capacity can change significantly. Therefore, no set of fixed CS limits as defined in (16) will be able to fully exploit the robot's movement capabilities.

B. Updating CS movement limits on-the-fly

To address the issue of constantly changing movement limits (16) during the robot trajectory execution, a real-time re-planning approach is proposed. In each step of trajectory execution the proposed approach first evaluates the jerk $\ddot{\mathbf{s}}$, acceleration $\dot{\mathbf{s}}$ and velocity \mathbf{s} limits, using the proposed method in Section III-A. Then the updated limits used with TAP algorithm to calculate the time-optimal profile of the remaining trajectory. Figure 4 illustrates the intuition behind the proposed approach, Algorithm 1 shows its pseudo-code and Figure 5 shows its integration in the overall robot control scheme.

As shown in Algorithm 1, in each time step k , corresponding to the moment in time t_k and the robot's state $\{\mathbf{q}_k, \dot{\mathbf{q}}_k, \ddot{\mathbf{q}}_k\}$,

Algorithm 1 One step of the proposed approach

Require: Path lengths d, θ and directions $\mathbf{c}_r, \mathbf{c}_t$ (18,19,23)

Require: Current robot state $(\mathbf{q}_k, \dot{\mathbf{q}}_k, \ddot{\mathbf{q}}_k)$, $(X_k, \dot{\mathbf{x}}_k, \ddot{\mathbf{x}}_k)$

- 1: $s_{t,k}, s_{r,k} \leftarrow X_k$ (18 - 19)
- 2: $\dot{s}_{t,k}, \ddot{s}_{t,k}, \dot{s}_{r,k}, \ddot{s}_{r,k} \leftarrow \dot{\mathbf{x}}_k, \ddot{\mathbf{x}}_k$ (22)
- 3: $(\dot{s}_{t,min}, \dot{s}_{t,max}, \dots) \leftarrow \text{UpdateLimits}(\mathbf{c}_t, \mathbf{q}_k, \dot{\mathbf{q}}_k, \ddot{\mathbf{q}}_k)$ (11)
- 4: $(\dot{s}_{r,min}, \dot{s}_{r,max}, \dots) \leftarrow \text{UpdateLimits}(\mathbf{c}_r, \mathbf{q}_k, \dot{\mathbf{q}}_k, \ddot{\mathbf{q}}_k)$ (11)
- 5: $d_k, \theta_k \leftarrow \text{Update remaining path length}$ (18 - 19)
- 6: $(s_{t,0}, s_{r,0}, \dots) \leftarrow \text{Update initial conditions}$ (24)
- 7: $(s_t, s_r, \dot{s}_t, \dot{s}_r, \dots) \leftarrow \text{TAP}(d_k, \theta_k, s_{t,0}, s_{r,0}, \dot{s}_{t,0}, \dot{s}_{r,0}, \dots, \dot{s}_{t,min}, \dot{s}_{r,min}, \ddot{s}_{t,min}, \ddot{s}_{r,min}, \dots)$ (20)
- 8: $X_d \leftarrow s_{t,k+1}, s_{r,k+1}$ (20)
- 9: $\dot{\mathbf{x}}_d, \ddot{\mathbf{x}}_d \leftarrow \dot{s}_{t,k+1}, \dot{s}_{r,k+1}, \ddot{s}_{t,k+1}, \ddot{s}_{r,k+1}$ (21)
- 10: **return** $X_d, \dot{\mathbf{x}}_d, \ddot{\mathbf{x}}_d$

the instantaneous limits on the path variables $s_r(t)$ and $s_t(t)$ are calculated using the method described in Section III-A. The new updated limits have a form of min-max ranges for each one of the path variables (11). The translation limits are calculated using the direction vector \mathbf{c}_t and the rotation limits are calculated using \mathbf{c}_r , defined in (23).

Given the robot's coordinate on the path $s_t(t_k), s_r(t_k)$, the remaining length of the geometric path to the target position are calculated as $d_k = s_{t,T} - s_t(t_k)$ and the remaining angle of rotation $\theta_k = s_{r,T} - s_r(t_k)$. Then, the initial conditions are updated for the new planning execution

$$\begin{aligned} s_t \in [0, d_k], \quad \dot{s}_{t,0} = \dot{s}_t(t_k), \quad \ddot{s}_{t,0} = \ddot{s}_t(t_k), \\ s_r \in [0, \theta_k], \quad \dot{s}_{r,0} = \dot{s}_r(t_k), \quad \ddot{s}_{r,0} = \ddot{s}_r(t_k) \end{aligned} \quad (24)$$

Finally, the updated TAP trajectory is calculated for the translation $s_t(t)$ and the orientation $s_r(t)$, resulting in an optimal robot trajectory given the updated robot's limits (11) and the initial conditions (24). Equations (20 - 21) are then used to calculate the robot's desired CS pose X_d , velocity $\dot{\mathbf{x}}_d$ and acceleration $\ddot{\mathbf{x}}_d$, based on the optimal path variables $s_{t,k+1}, s_{r,k+1}, \dots, \dot{s}_{t,k+1}, \dot{s}_{r,k+1}$ in the next time step $k+1$.

The assumption of constant limits (11) for the duration of the remaining trajectory is a simplification that does not hold true in practice, as the robot's movement capacity can change significantly over time. However, this issue is addressed through real-time re-planning, which constantly updates the plan using the latest information on the robot's current capacity. The TAP planning algorithm is well-suited

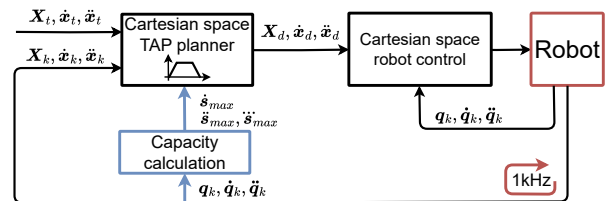


Fig. 5. A schematic overview of the integration of the proposed method into the real-time robot control framework. The elements in blue present the extension of a standard CS based TAP planning.

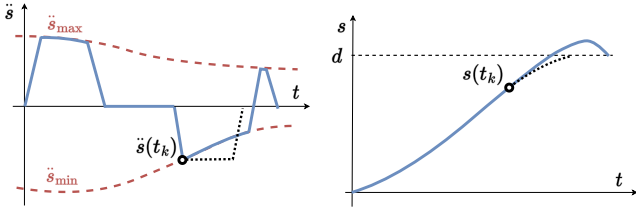


Fig. 6. The effect of diminishing braking capacity towards the end of the trajectory resulting with an overshoot.

for this approach, as it is highly efficient in computing optimal trajectories, allowing for real-time execution.

V. COMPENSATING FOR REAL-TIME RE-PLANNING NEGATIVE EFFECTS

The proposed approach, in each planning iteration, considers constant robot's capabilities until the end of the trajectory. Based on this prediction, the Trapezoidal Acceleration Profile (TAP) planning decides the phase of the trajectory: when to stop accelerating and when to start braking. In some cases, especially when the robot's capacity changes significantly towards the end of the trajectory, TAP planning can produce chattering effect and an overshoot due to switching between the modes of braking and accelerating.

Section V-A describes the overshoot effect due to the robot's decreasing braking capacity towards the end of the trajectory, while Section V-B describes the chattering effect produced by the robot's increasing braking capacity towards the end of the trajectory. The sections propose heuristics for account for these effects and numerically validate the choice of their parameters. Finally, Section V-C then brings a short discussion about the diminishing movement abilities when passing close to Joint Space (JS) singularities.

A. Decreasing braking capacity: Overshoot effect

The overshoot effect comes from the assumption of the constant robot's braking capacity till the end of the trajectory. If the robot's braking capacity suddenly decreases during the braking phase of the trajectory, as it is the case in the trajectory time-step t_k on Figure 6, TAP planning overestimates the robot's braking ability and starts braking too late. In that case the robot is not able to come to the complete stop before it reaches the target position, resulting in the overshoot behavior.

This is an inherent challenge of real-time planning and cannot be solved without adding a degree of prediction of the robot's future capabilities. One such approach is to predict the robot's minimal braking capacity for the remaining Cartesian Space (CS) path $\mathcal{P}(s(t_i))$ until the end of the trajectory $t_i \in [t_k, T]$ and use this value for TAP planning.

$$\ddot{s}_{min} = \max \{ \ddot{s}_{min}(t_i) \} \quad t_i \in [t_k, T] \quad (25)$$

This approach in many cases results in more conservative trajectories as the robot's braking capacity is underestimated, but at the same time it guarantees the overshoot removal. As the robot's braking capacity depends on its joint state $\{q, \dot{q}\}$, finding the minimal braking capacity (25) requires

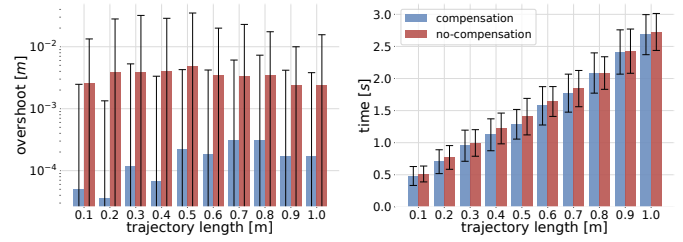


Fig. 7. Plots showing the comparison of the average overshoot (left) and average execution time (right) of the proposed method with and without overshoot compensation. For each length ranging from $d = 0.1$ to $1m$, means and variances are calculated over 100 random trajectories.

either knowing the exact robot's JS path until the end of the trajectory, which is by definition not known, or finding all the possible joint configurations the robot can be in on the remaining path, which is very long and not practical for most real-time applications.

In this work a sampling-based approximation of (25) is proposed, where the approximation is done by calculating the minimal braking capacity between the current one $\ddot{s}_{k,min}$ (corresponding to the current position X_k in the time step t_k and state $\{q, \dot{q}\}$) and the braking capacity at the end of the trajectory $\ddot{s}_{T,min}$.

$$\ddot{s}_{min} = \max \{ \ddot{s}_{k,min}, \ddot{s}_{T,min} \} \quad (26)$$

The JS configuration q_T , at the end of the trajectory, is calculated by evaluating the Inverse Kinematics (IK) solution for the pose X_T and taking the closest one to the initial configuration q_0 . Joint velocity \dot{q}_T and acceleration \ddot{q}_T are considered to be $\dot{q}_T, \ddot{q}_T = \mathbf{0}$, as the robot will come to a stop at the target X_T .

This method (26) offers the added advantage of efficient implementation, which requires evaluation of the final braking capacity of the robot $\ddot{s}_{T,min}$ only once per execution of the trajectory. However, this approach presents a relatively coarse approximation of (25) and especially for longer trajectories, where it is hard to predict the robot's final JS configuration, this approach can lead to significant under-estimation of robot's braking capacity and significantly increase the trajectory time.

Experimental validation of compensation parameters: Figure 7 presents a comparison between the proposed method's performance with and without overshoot compensation. To evaluate the methods, 1000 random translation trajectories (random fixed orientation) were generated in the robot's workspace, ranging in length from $d = 10cm$ to $d = 1m$. The experiments were conducted in simulation using a Franka Emika Panda robot, the implementation details are described in Section VI. The overshoot and execution time were recorded for each trajectory. The results indicate that the proposed method with overshoot compensation significantly reduces the expected overshoot compared to the method without compensation, going from $3mm$ average overshoot to $0.1mm$. Moreover, the compensation strategy does not seem to have a negative impact on the trajectory execution time. On the contrary, as shown in the Figure 7, the experimental results

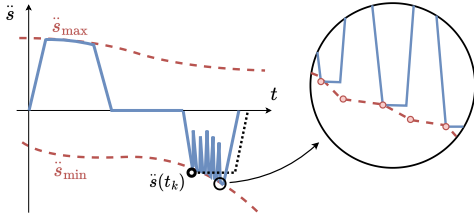


Fig. 8. Due to significantly increasing braking capacity towards the end of the trajectory, the robot starts braking too early. As soon as the new higher braking capacity is updated, the TAP planning stops braking and accelerates again. As the braking capacity continues increasing, this effect continues and sometimes results with the chattering effect, due to the rapid switching between the modes of deceleration and acceleration.

suggest that having the overshoot penalises the execution time more than having a more conservative strategy that starts breaking earlier with reduced but constant intensity. However, depending on the path taken, especially if the target X_T is close to a singularity, where the braking capacity $\ddot{s}_{T,min}$ would be very small, the proposed compensation approach could lead to the under-estimation of robot's braking capacity and significantly increase the trajectory time.

B. Increasing braking capacity: Chattering effect

When the robot's braking capacity increases along the trajectory, scenario shown in Figure 8, the proposed method can result in chattering behaviour. Due to considering the robot's capacity constant, the robot's braking capacity is underestimated and the decision of the TAP planning to start braking, at the time step t_k , comes too soon. In the next step t_{k+1} the robot's braking capacity increases which makes TAP planning take the decision to stop braking. This sequence of switching modes between braking and accelerating can produce chattering effect of the planned acceleration profile and consequently jerky trajectories.

The chattering is a common consequence of fast mode switching and is well studied in the sliding mode control systems [40]. When it comes to chattering in real-time trajectory planning, there are two main approaches proposed in the literature. 1) Garelli et al. [41] proposed introducing low-pass filtering of the high variations of the acceleration, having a direct impact of the jerk variations and smoothing the trajectory. 2) Instead of filtering, Bianco et al. [22] proposed introducing more restrictive jerk limits which directly reduce the magnitude of the acceleration chattering.

In this paper, an alternative approach is proposed, consisting in down-sampling the TAP planning. Instead of planning the TAP trajectory in each time step t_k , the trajectory is planned with the time step Δt_p , while linearly interpolating the trajectory between the planning steps $t \in [t_k, t_k + \Delta t_p]$.

$$s(t) = s_k + \frac{s_{k+1} - s_k}{\Delta t_p} (t - t_k), \quad t \in [t_k, t_k + \Delta t_p] \quad (27)$$

where s_k is path position in the current step t_k and s_{k+1} is its values in the next planning step $t_{k+1} = t_k + \Delta t_p$ calculated by the TAP planning. The same linear interpolation can be applied to the velocity \dot{s} and acceleration \ddot{s} .

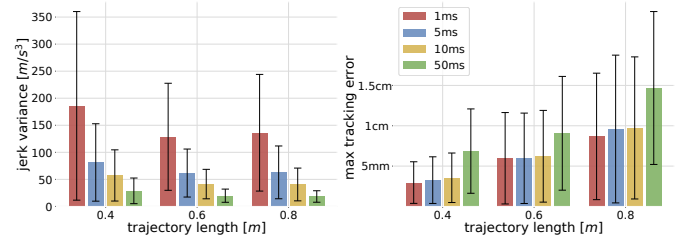


Fig. 9. Plots showing the comparison of the jerk variance (left) and tracking error (right) of the proposed method with and without oscillation compensation, for trajectory lengths $d = 0.4, 0.6$ and $0.8m$. Means and variances are calculated over 100 random trajectories. Four down-sampling times are compared $\Delta t_p = 1, 5, 10$ and $50ms$.

Between the TAP planning steps, the robot's movement capacity is considered constant and in that way the high-frequency chattering induced by rapid changes in the braking capacity are filtered. However, when implementing this compensation strategy, a choice of the planning step Δt_p requires a trade-off to be made between the filtering the high-frequency oscillations and the allowable level of tracking error. The longer the time between the planning Δt_p , the more the effects of the coupling are reduced. On the other hand, as the robot's movement capacity is considered constant between the planning steps Δt_p , the planned trajectory can potentially overestimate the robot's capacity in this period and lead to the increase of the tracking error due to its inability to follow the planned trajectory.

Experimental validation of compensation parameters: In order to find the optimal down-sampling time Δt_p an empirical study is conducted for 100 random trajectories with lengths $d = 40cm, 60cm$ and $80cm$ in the robots workspace. The experiments are conducted in simulation using a Panda robot, the implementation details are described in Section VI. Four down sampling times are compared $\Delta t_p = 1ms, 5ms, 10ms$ and $50ms$. For each executed trajectory the maximal deviation from the path and the CS jerk variance is evaluated.

Figure 9 illustrates an inverse proportionality between the down sampling time and the jerk variance. Additionally, it reveals that as the planning time step Δt_p increases, there is a corresponding increase in the deviation from the desired path. Therefore, the choice of the appropriate planning time step Δt_p implies making a trade-off between the two. In the case of this work, the planning step $\Delta t_p = 10ms$ is chosen, resulting in a significant decrease in the jerk variance while having relatively low impact on the tracking error.

C. Diminishing movement ability: Singularity effect

When the CS path passes near or through a singularity (where $\text{rank}(J) < m$), the robot's velocity (\dot{s}) capacity can be significantly reduced and may even drop to zero, as shown in Figure 10 (left). While the limits of the higher order derivatives, acceleration (\ddot{s}) and jerk (\dddot{s}), can even go beyond zero and change signs (see right plot).

If the robot's velocity capacity decreases to zero and it is unable to continue moving along the trajectory, the proposed method will fail to complete the path, causing the robot

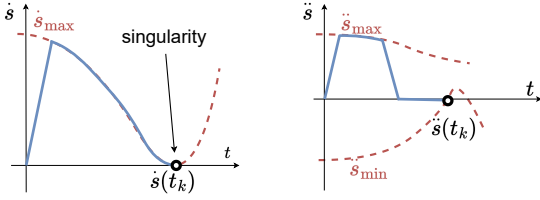


Fig. 10. The effect of diminishing velocity \dot{s} and acceleration \ddot{s} capacity in the path direction due to the effect of singularity.

to stop. Similarly, if the acceleration (\ddot{s}) or jerk (\dddot{s}) drop beyond zero, the robot locally loses the ability to accelerate or decelerate and TAP planning might fail as it can no longer reach the end of the trajectory. In practice TAP planning requires upper and lower limits to have positive and negative signs respectively ($\dot{s}_{max}, \ddot{s}_{max}, \dddot{s}_{max} \geq 0$ and $\dot{s}_{min}, \ddot{s}_{min}, \dddot{s}_{min} \leq 0$). To account for this, in this paper all the positive and negative limits calculated by (11) are clipped to zero: $x_{max} = \max\{0, x_{max}\}$ and $x_{min} = \min\{0, x_{min}\}$. This measure ensures that TAP planning will never fail, but at the same time can cause a deviation from the planned path and increase the tracking error. This issue is a direct limitation of TAP algorithm and potentially better approach would be to replace TAP planning by a more flexible dynamical filter such as in the work of [42].

The confound effects of passing close to singular configurations, where the robot might experience sudden and abrupt changes in its movement abilities, might result in robot's current path state $\{\dot{s}_k, \ddot{s}_k\}$ violating the local movement limits in the path direction (ex. $\dot{s}_k > \dot{s}_{max}$ or $\ddot{s}_k < \ddot{s}_{min}$). In such situations, TAP planning will produce the minimal-time trajectory that brings the path state $\{\dot{s}_k, \ddot{s}_k\}$ back within the limits, but this might result in a deviation from the path and increase the tracking error.

Therefore, when it comes to the tracking performance of the proposed method, as most of the CS planning methods [43], avoiding singular JS configurations is a key challenge. As the proposed method plans in CS and considers not having access to the low-level JS robot control, it relies on the low-level robot controller for singularity avoidance [44][45].

VI. EXPERIMENTAL SETUP

All the experiments, both in simulation and in real world, are conducted on a Franka Emika Panda robot. The robot's kinematic limits in Joint Space (JS) and the Cartesian Space (CS) are obtained from Franka Emika's official datasheet [36]. These values are publicly available⁴ and are listed in Table I.

A. Robot control architecture

The overall schematic diagram of the proposed planning method in combination within the robot control paradigm is shown on Figure 5. Given the robot's current state $\{q_k, \dot{q}_k, \ddot{q}_k\}$ in step k , the approach first determines the robot's current CS movement capacity in the path direction using the approach

⁴Full set of parameters is available, as of November 2025, at: https://frankarobotics.github.io/docs/control_parameters

TABLE I
FRANKA EMIKA PANDA ROBOT KINEMATIC LIMITS⁴. THE LOWER LIMITS ARE SYMMETRIC TO THE UPPER ONES.

Joint Space Kinematic Limits							
Limits	q_0	q_1	q_2	q_3	q_4	q_5	q_6
\dot{q}_{max} [rad/s]	2.17	2.17	2.17	2.17	2.61	2.61	2.61
\ddot{q}_{max} [rad/s ²]	15	7.5	10	12.5	15	20	20
\dddot{q}_{max} [krad/s ³]	7.5	3.75	5.0	6.25	7.5	10.0	10.0

Cartesian Space Kinematic Limits		
Limits	Translation	Orientation
\dot{x}_{max}	1.7 m/s	2.5 rad/s
\ddot{x}_{max}	13 m/s ²	25 rad/s ²
\dddot{x}_{max}	6500 m/s ³	12500 rad/s ³

described in Section III-A, resulting in velocity \dot{s}_{max} , acceleration \ddot{s}_{max} and jerk \dddot{s}_{max} limits. Using the updated limits, the robot's current CS state $X_k, \dot{x}_k, \ddot{x}_k$ and the target state $X_t, \dot{x}_t, \ddot{x}_t$, the proposed method calculates the new optimal trajectory using Trapezoidal Acceleration Profile (TAP) planning, as described in Section IV-B. Once the optimal TAP trajectory is found, desired states $X_d, \dot{x}_d, \ddot{x}_d$ are sent to the inverse velocity kinematics layer of the control architecture.

In this work, the robot control strategy for real-time CS trajectory following is formulated as a Quadratic Program (QP) and solved in each control loop.

$$\ddot{q}_{opt} = \arg \min_{\ddot{q}} \underbrace{\|\ddot{x}^* - J\ddot{q} - \dot{J}\dot{q}_k\|^2}_{\text{trajectory tracking}} + \underbrace{\omega_r \|\ddot{q}_r - \ddot{q}\|^2}_{\text{regularisation}} \quad (28)$$

s.t. $\ddot{q} \in [\ddot{q}_{lb}(q_k, \dot{q}_k, \ddot{q}_k), \ddot{q}_{ub}(q_k, \dot{q}_k, \ddot{q}_k)]$

where the $q_k, \dot{q}_k \in \mathbb{R}^n$ is the robot's current state, $\ddot{q}_k \in \mathbb{R}^n$ is the robot's current acceleration⁵, $J(q_k) \in \mathbb{R}^{m \times n}$ and $\dot{J}(q_k, \dot{q}_k) \in \mathbb{R}^{m \times n}$ are the robot's state dependent jacobian matrix and its time derivative. The bounds of each of joint accelerations $\ddot{q}_{i,ub}, \ddot{q}_{i,lb}$ are calculated in a way to guarantee that the joint jerk \dddot{q} , acceleration \ddot{q} , velocity \dot{q} and position q in the horizon δt respect their limits.

$$\ddot{q}_{i,ub} = \min \left\{ \underbrace{\ddot{q}_{i,k} + \underbrace{\ddot{q}_{i,max} \delta t}_{\text{jerk}}}_{\text{velocity}}, \underbrace{\ddot{q}_{i,max}}_{\text{acceleration}}, \dots \right. \\ \left. \underbrace{\frac{1}{\delta t} (\dot{q}_{i,max} - \dot{q}_{i,k})}_{\text{velocity}}, \underbrace{\frac{2}{\delta t^2} (q_{i,max} - q_{i,k} - \dot{q}_{i,k} \delta t)}_{\text{position}} \right\} \quad (29)$$

where the horizon δt has to be chosen long enough to ensure constraints compatibility without leading to conservative behaviour [28]. Equation (29) shows the upper bound expression, the lower bound calculation is equivalent, and is obtained by substituting *min* for *max* and maximising instead of minimising. The horizon δt chosen for the experiments is 15ms.

The optimisation problem (28) consists in two tasks: trajectory tracking task and a secondary (regularisation) task.

⁵As the joint acceleration measurements are quite challenging and often relatively noisy (often based on the numerical derivation of joint velocities), throughout this paper the robot's current JS acceleration \ddot{q}_k is estimated to be the equal to the control vector \ddot{q}_{opt} calculated by the QP formulation (28) in the previous time-step ($k - 1$).

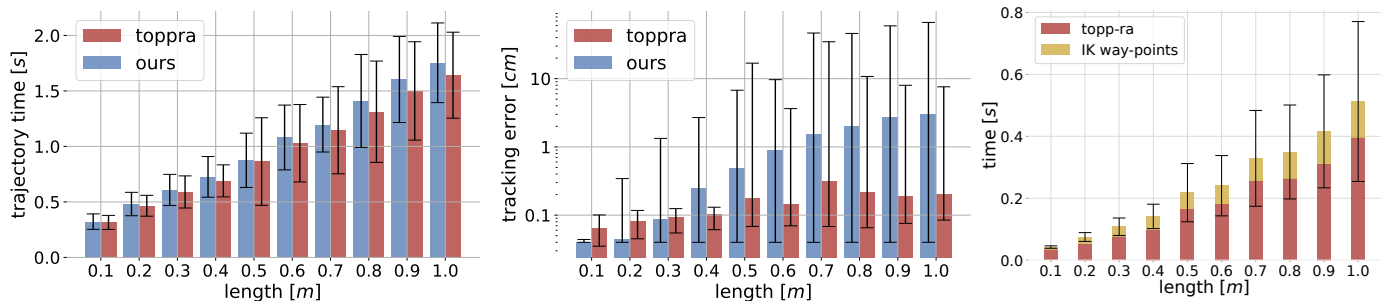


Fig. 11. Left plot shows the comparison of the trajectory execution time of the proposed method with TOPP-RA, for trajectory lengths ranging from $d = 0.1$ to $1m$, while the center plot shows the trajectory tracking error of the both methods. The graph on the right shows the computation time of the TOPP-RA algorithm on the same experiment as a comparison to the proposed method that runs in real-time. Means and variances are calculated over 100 random trajectories.

Trajectory tracking task: The trajectory tracking task is accomplished using the control law formulated as a PD controller with a feed-forward term through the desired Cartesian acceleration \ddot{x}^*

$$\begin{aligned} \ddot{x}^* &= K_p e + K_d(\dot{x}_d - \dot{x}_k) + \ddot{x}_d \\ e &= \text{Ad}(X_k) \log(X_k^{-1} X_d) \end{aligned} \quad (30)$$

$X_d, \dot{x}_d, \ddot{x}_d$ are the desired CS pose, velocity and acceleration in the next step, while X_k and \dot{x}_k are the measured CS pose and velocity in current step k . $K_p, K_d \in \mathbb{R}^m$ are diagonal matrices containing the proportional and derivative gains. Vector e is the CS pose error expressed in the world frame and the $\text{Ad}(X_k) \in \mathbb{R}^{6 \times 6}$ is the adjoint matrix [46, Section II.F]. In the experiments, the PD controller gains used are

$$\begin{aligned} K_p &= \text{diag}([170, 170, 170, 100, 100, 100]) \\ K_d &= \text{diag}([50, 50, 50, 30, 30, 30]) \end{aligned}$$

Regularisation task: The robot's redundant degrees of freedom are used to dampen the movement in the trajectory null-space and keep the robot away from its joint limits. It is expressed through the joint acceleration \ddot{q}_r

$$\ddot{q}_r = k_{rp}(\mathbf{q}_r - \mathbf{q}) - k_{rd}\dot{\mathbf{q}}, \quad \mathbf{q}_r = 0.5(\mathbf{q}_{max} + \mathbf{q}_{min}) \quad (31)$$

where k_{rp} and k_{rd} are scalar gains and \mathbf{q}_r is the initial robot pose at the center of all the joint ranges. The secondary task gains used are $k_{rp} = 5s^{-2}$ and $k_{rd} = 2\sqrt{k_{rp}}s^{-1}$, while secondary task weight is $\omega_r = 1e^{-5}$.

The robot is controlled using the joint velocity commands which are calculated using an Euler backward numerical integration

$$\dot{\mathbf{q}}_{k+1}^* = \dot{\mathbf{q}}_k^* + \ddot{\mathbf{q}}_{opt} \Delta t \quad (32)$$

where $\Delta t = 1ms$ is the sampling time at which both the control loop and the proposed trajectory planning approach are executed.

Singularity avoidance: manipulability-maximisation: As discussed in Section V-C, the proposed trajectory planning method is relatively vulnerable to the effects of kinematic singularities. It does not have control over the JS path taken by the robot and delegates their avoiding to the underlying robot control. QP based controllers are particularly suitable for this task, as they can easily incorporate an additional regularisation task accomplishing this. Avoiding singularities is in practice

often accomplished by maximising the manipulability index [47] using approaches such as [48][45]. As this feature not strictly necessary for the implementation of the method, and many of simpler industrial controllers might not implement it, it is omitted in this work.

B. Software implementation details

Both simulation and real-world experiments use the open-source library `pinocchio` [49] for rigid body simulation, `ruckig` library [15] for TAP planning and the Linear Program (LP) solver used is GLPK [50]. All the simulations and experiments are run on a computer with a processor: Intel i7-8650U @ 1.90GHz .

The simulation experiments used for the comparative studies are implemented in Python using open-source implementations of TOPP-RA [6] while `robotics-toolbox` library [51] (version *v1.1.0*) was used for calculating the Inverse Kinematics (IK) (C++ implementation of the *Levenberg-Marquardt* solver). The code used for the simulation experiments is open-source and publicly available⁶.

For the mock-up experiment, the real-time execution of the proposed method is implemented in C++ and integrated using Robot Operating System (ROS).

VII. COMPARATIVE STUDY

A simulation based comparative study is conducted to evaluate the performance of the proposed method against well-known time-optimal planning algorithms. First, the proposed algorithm is compared to a state-of-the-art time-optimal Joint Space (JS) planning method called TOPP-RA [6]. Then, the proposed algorithm is compared against the standard reactive planning method based on Trapezoidal Acceleration Profile (TAP) planning with fixed Cartesian Space (CS) limits provided by the manufacturer.

A. Comparison: Time-optimal planning in Joint space

To evaluate the performance of the trajectories found and executed by the proposed online CS planning approach, it is compared against a state-of-the-art time-optimal JS planning

⁶Git: https://github.com/auctus-team/capacity_aware_trajectory_planning

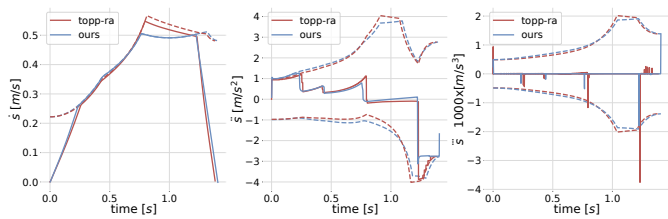


Fig. 12. The plots show the comparison of the path velocity, acceleration and jerk, generated by the proposed approach (blue) and TOPP-RA (red) for a single 50cm straight line path. The dashed lines show the robot’s movement limits in the trajectory direction while executing the paths (red toppra, blue ours). In both cases the limits are calculated using the method described in Section III-A.

algorithm TOPP-RA [6]. As opposed to the proposed method, which re-plans on-the-fly, TOPP-RA is an offline algorithm that requires pre-planning for the time-optimal JS trajectories. The goal of this analysis is to compare the generated trajectory profiles of TOPP-RA to the proposed approach as well as their execution times, in order to assess the time-optimality of the proposed real-time planning approach.

TOPP-RA’s drawback however, is that it cannot deal with CS trajectories directly and it requires an additional step of finding the JS path corresponding to the CS path. For TOPP-RA, the full JS path is determined by sampling the CS path $\mathcal{P}(s)$ into the set of way-points $X_{w,i}$ and finding the inverse kinematic solution $q_{w,i}$ for each one of them. To ensure the JS path continuity, the inverse kinematics solution $q_{w,n+1}$ at the way-point $X_{w,n+1}$ is taken as the one closest to the joint configuration $q_{w,n}$ in the previous way-point $X_{w,n}$. The distance between the way-points used in these experiments is 5cm.

Both approaches are tested using 100% of the robot’s JS velocity, acceleration and jerk capacity $\alpha_v = \alpha_a = \alpha_j = 1.0$. Both algorithms are ran on a set of 1000 random straight line trajectories (with random fixed orientation), ranging in length from $d = 10cm$ to 1m. All the trajectories start and end by the robot being fully stopped ($\ddot{s}_T = \dot{s}_T = \ddot{s}_0 = \dot{s}_0 = 0$).

Results: The comparison of the planned velocity, acceleration and jerk profiles is shown in Figure 12. It can be seen that both methods find similar time profiles for all the path variables, showcasing the effectiveness of the proposed approach. Moreover, the figure further confirms the accuracy of the proposed method’s limit calculation, as even though TOPP-RA plans entirely in JS, using JS limits (1), its generated trajectory respects the path variable velocity \dot{s} and acceleration \ddot{s} limits (red dashed lines) calculated by the proposed method in Section III. Additionally, TOPP-RA does not account for the jerk limits, which can be seen on the jerk plots in Figure 12.

The influence of the overshoot compensation strategy can be observed in the middle plot on Figure 12. The proposed approach starts braking at the same time as TOPP-RA and does not use the full braking capacity, stopping slightly later. The resulting trajectory does not have an overshoot, even though the robot’s braking capacity decreases significantly towards the end of the trajectory. This demonstrates the effectiveness of the proposed strategy in reducing overshoot while not significantly increasing the execution time.

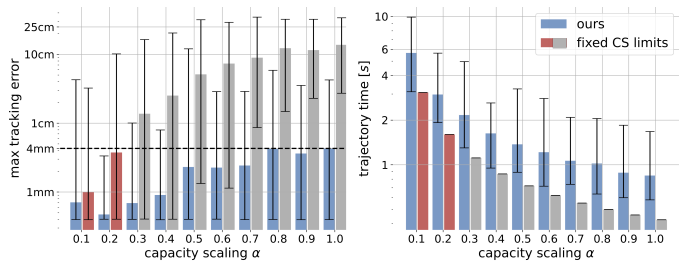


Fig. 13. Result of the benchmarking experiment comparing the proposed method (blue) to the fixed CS planning approach (red and gray). The graph on the left shows comparison of the max tracking error with respect to the ratio of the capacity α used while the graph in the right shows the trajectory execution time comparison. The gray colored bins show the fixed CS planning experiments which show more tracking error than the proposed method (average max error higher than 4mm). Means and variances are calculated on 100 random robot trajectories.

The trajectory execution time comparison is presented in Figure 11 (left and center). The figure shows that the proposed method has a comparable execution time for all tested trajectory lengths. The average trajectory execution time of the proposed method ranges from of 2% longer 7% than TOPP-RA time. This result confirms the near time-optimal nature of the proposed method.

Figure 11 (right) shows the TOPP-RA algorithm’s computation time in the same experiment. The results highlight the considerable computation times of the TOPP-RA algorithm, ranging from 50ms for short to over 0.5 second for longer trajectories (roughly half a second per meter of the trajectory). A significant part of the computation time can be attributed to the inherent complexity of the algorithm and the requirement to transform the CS path to the set of JS way-points. Therefore the choice of the Inverse Kinematics (IK) solver has a direct impact on the efficiency of TOPP-RA planning. This result further emphasises the potential of the proposed approach. It produces comparable time-optimal trajectories to TOPP-RA without the need for time-consuming pre-computation.

It is worth noting that since the proposed approach plans entirely in CS, its resulting JS path can differ considerably from the pre-computed TOPP-RA path. As shown on Figure 11, in some cases the proposed approach might have shorter trajectory times, by following a more favourable JS path than TOPP-RA’s one. The opposite effect is possible as well, the proposed approach taking a less favourable JS path. The proposed method delegates determining the JS path to the lower-level controller, while it plans the near-time-optimal trajectory accounting for the changes in robot’s movement abilities along any chosen JS path.

B. Comparison: Time-optimal planning in Cartesian space

The proposed adaptive approach is further compared to a standard TAP trajectory planning using fixed CS kinematic limits which is often used for online re-planning of reactive trajectories. The CS limits for the Franka Emika panda robot are taken from the standard datasheet [36], as given in Table I.

The approaches are compared over 100 straight line trajectories in the robot’s workspace (with random fixed orientation), with a fixed length of $d = 50cm$. All the trajectories start and

end by the robot being fully stopped ($\ddot{s}_T = \dot{s}_T = \ddot{s}_0 = \dot{s}_0 = 0$). The performance of the two approaches is compared for different scaling levels α , starting at 10% ($\alpha=0.1$) of the robot's capacity and going to 100% ($\alpha=1.0$). The scaling strategy is chosen to be equal for velocity, acceleration and jerk $\alpha = \alpha_v = \alpha_a = \alpha_j$.

Results: Figure 13 presents the results of a comparative study between the proposed method and the TAP planning method with fixed CS limits. Both methods show a linear relationship between the tracking error and the level of capacity used (α), however the proposed approach has significantly lower mean errors and variances for the same values α . This can be attributed, in part, to the overestimated CS limits (Table I) provided by the manufacturer for the Panda robot, see Figure 1. Due to this overestimation, the planned TAP trajectories, depending on the path direction in the robot's workspace, can become infeasible and induce large tracking errors. This is particularly apparent when using a larger percentage of the robot's movement capacity. For example, Figure 13 (on the left) shows that even for 50% of specified robot's CS limits ($\alpha=0.5$), the average tracking error is around 6cm with maximal values above 25cm on this 50cm trajectory. On the other hand, the proposed method keeps the tracking error under 4mm all the way up to 100% robot's limits.

Figure 13 further shows that any value above 20% of the manufacturer's CS limits (i.e., $\alpha = 0.2$) results in higher tracking error than the proposed method (> 4 mm). While for $\alpha = 0.2$, the fixed CS planning has an execution time of 1.6 seconds, the proposed method has a lower execution time for all $\alpha \geq 0.4$. These results confirm that when it comes to planning for highly dynamic trajectories, taking in consideration the robot's true movement capacity results in faster and more precise movements.

To illustrate the comparison between both methods, a short video is publicly available⁷. It highlights the effect of increasing the value of α on the trajectory time and, more importantly, on the tracking error.

VIII. MOCK-UP EXPERIMENT: COLLABORATIVE WASTE SORTING

Waste recycling is an important factor in reducing humanity's environmental impact. One of its crucial parts is the waste sorting procedure, where the recyclable materials are extracted from the regular waste and prepared for the recycling process. Robotics and computer vision technologies have a great potential to improve the waste processing efficiency and increase its volume [52]. However, waste sorting is a highly dynamic and unstructured environment, presenting many challenges to potentially robotised solutions. To be viable, waste sorting robots need to be able to operate at high speeds, where they use pick-and-place or pick-and-toss [53] techniques to sort the waste of different material groups. Therefore, one of the key challenges for building such systems is producing efficient and reactive movement generation techniques.

Several robotic solutions have been proposed in the literature to address waste sorting tasks [54], such as the *Zen-*

*robotics Fast picker*⁸ or *SELMA*⁹. However, these solutions are based on expensive industrial robots and are only viable for large scale recycling facilities.

In this work, we propose a mock-up interactive (collaborative) scenario for waste sorting that leverages the proposed real-time trajectory planning method in order to create fast and adaptable robot movement. In the experiment, a human operator is introducing different waste items at the sorting workstation, where the robot is placed. The operator introduces the waste items at any time and in any order, as well as modify their position and orientation on the table. The operator can do the same with the sorting buckets (the bins in which the sorted items are placed). The robot's job is to pick all the waste items and place them in the appropriate sorting buckets as fast as possible.

Waste items (two cans and two cartons), as well as the buckets, are tracked in real-time using a motion capture system *OptiTrack*. In order to efficiently and robustly sort the waste, object manipulation procedure is divided in 6 phases.

- 1) Position the gripper 5cm above the object with the appropriate orientation
- 2) Lower the gripper to the object
- 3) Close the gripper
- 4) Rise the object 10 cm
- 5) Transport the object to the appropriate bucket
- 6) Release the object

For this experiment, 60% of the robot's capacity is used ($\alpha=0.6$) in order to keep the tracking error under 1cm, as shown on Figure 13. The experiment is available in a form of accompanying video¹⁰.

Figure 14 shows the time evolution of one run of the experiment. The plot shows the evolution of the position s_t and orientation s_r in time as well as their respective velocities \dot{s}_t, \dot{s}_r . The grey areas show the moments in time where the gripper is closing or opening in order to grasp or let go of an object. Several images depicting various moments of the experiment are shown below the plot.

The experiment starts with the human operator placing the sorting bins (metal and paper) in the robot's workspace. Then, the operator introduces the objects (sometimes more than one at a time) on the table. The robot plans and executes the necessary trajectories in real-time, in order to place the objects in the appropriate sorting bin. Objects' and bins' positions and orientations are not known in advance and can change in real-time.

The velocity \dot{s}_t, \dot{s}_r show in Figure 14 demonstrate that the proposed approach is able to follow the robot's changing capacity and produce motions with the maximum reachable velocities. It can also be seen that, for the different trajectories executed, either the translation velocity \dot{s}_t or the rotation velocity \dot{s}_r is maximised. They are not both exploited simultaneously because the translation and orientation is synchronised in order for the robot to reach both target position and orientation at the same time.

⁸Zenrobotics Fast picker: <https://zenrobotics.com/de/>

⁹SELMA: <https://www.opteknik.se>

¹⁰Video: <https://youtu.be/BHGipnKNOFA>

⁷Video: <https://youtu.be/KBo0ZHih3I>

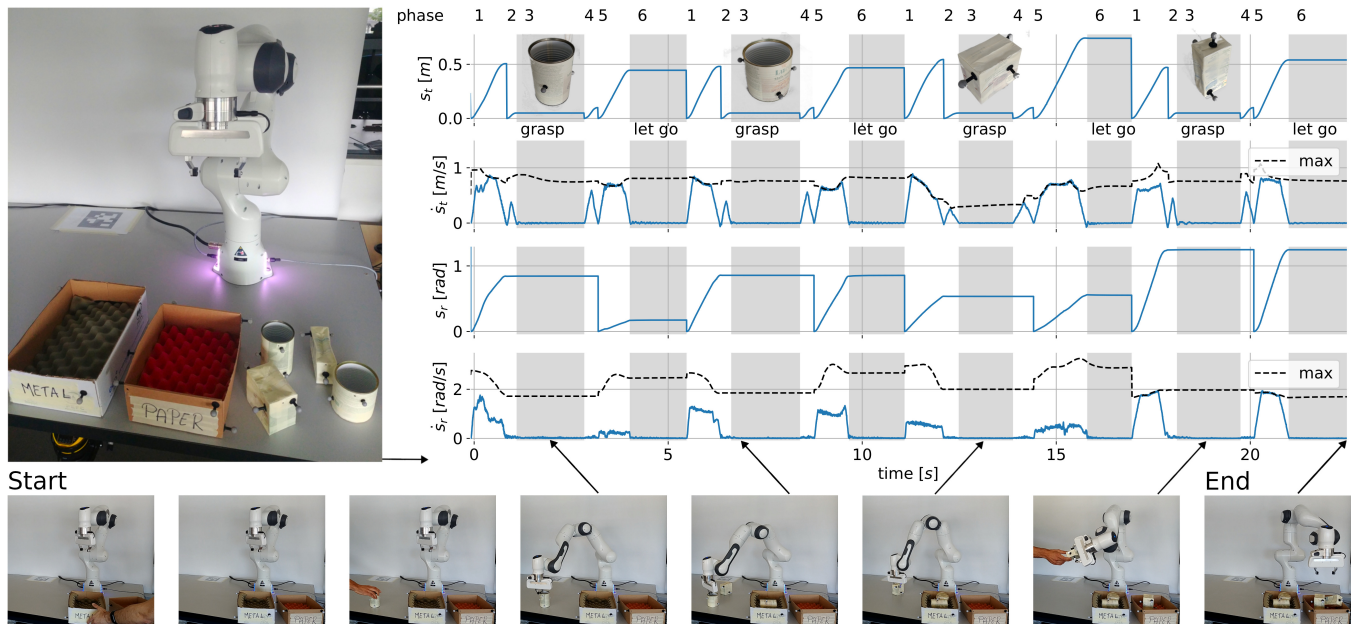


Fig. 14. Figure on the left shows the experimental setup of the mock-up waste sorting experiment. The robot used in the experiments is a Franka Emika Panda robot. The experiment uses six objects: two cans, two cartons and two sorting bins, all tracked in real time using motion capture system Optitrack. The images on the bottom show several moments of the experiment, while the plot (up right) shows the time evolution of the real-time executed trajectories. The plot shows the position s_t and orientation s_r path evolution in time as well as translation \dot{s}_t and rotation \dot{s}_r velocity on the path and their calculated maximal values (dotted lines). In the experiment, the operator first brings the sorting bins to the robot workstation, then introduces the waste objects to the robot with unknown positions and orientations. Robot plans and executes in real-time the trajectories necessary in order to place, as fast as possible, the objects into the appropriate sorting bin.

The adaptability of the proposed approach is further demonstrated as the final waste item is handed over to the robot, where the robot grasps the object from the operator's hand.

IX. DISCUSSION

While the results in this paper seem promising and make a case for using the proposed approach in real world applications, this section focuses on its main limitations and draws a more detailed parallel with Model Predictive Control (MPC).

A. A parallel with the MPC

In each step of the trajectory execution, the proposed approach efficiently calculates the robot's instantaneous Cartesian Space (CS) movement capacity, using the method described in Section III-A. Then, the robot's CS capacity is considered constant until the end of the trajectory and used to recalculate the time-optimal trajectory for the remaining path using Trapezoidal Acceleration Profile (TAP) planning. As the approach re-plans in each trajectory execution step, the assumption of constant limits is reasonable, since only the first step of each planned trajectory is ever executed (for which the calculated limits are valid).

The proposed approach can be seen as a simplified special case of the MPC approach, where the robot's model is entirely integrated within the robot's movement capacity limits, which are state dependent and calculated in each time-step. Then the real-time TAP re-planning makes the prediction of the robot's behaviour until the end of the trajectory and provides the optimal action to be made in order to execute the desired

path in minimum-time. While lacking the flexibility of MPC, such as considering additional optimality criteria (apart from minimum-time) or variable prediction horizon length (the proposed method always plans until the end of the trajectory), the proposed approach has a significantly higher computational efficiency and allows for real-time execution.

Recent advances in this domain are increasingly closing the gap to the real-time (sub-millisecond) operation. For example, Kleff et al. [55], showed to be able to implement a non-linear MPC at 1kHz but with 30ms planning step, Eckhoff et al. [56] showed the potential of a linear MPC running it with 25ms time-steps.

B. Limitations and ways of improvement

One limitation of the proposed approach is its assumption that the CS path can be represented using only one variable $s(t)$ (or two $s_t(t)$, $s_r(t)$ in the case of both translation and orientation). This assumption enables transforming the polytope-based capacity metrics (5) to the interval ranges of the path variables (11). As shown in Section III-A, these ranges can be efficiently calculated in real-time and used with a standard trajectory planning algorithm such as TAP. One drawback of this assumption is that the proposed method plans for the robot's motion only in the path direction, while the underlying robot controller is delegated with making sure that there are no other movements generated in other directions in space (orthogonal to the path). This is perfectly reasonable for straight line trajectories. However, if the application requires this method to be used with paths that are not straight lines, or

in dynamical environments with moving or changing targets, the appropriate robot controller has to be chosen, one able to deal with the attenuation of the orthogonal movements and respecting the robot's limits.¹¹ Another way to overcome this drawback would be to use a trajectory planning algorithms able to integrate polytope representations of path constraints (5) and avoid making the straight line assumption all-together. Therefore, a promising future direction is adapting the planning method for the family of planning techniques based on Quadratic Program (QP) optimisation, such as the MPC approach, which allow for integrating polytope-shaped limits, since they can be expressed in a form of linear constraints.

A different limitation of the proposed approach, that could be resolved using the same set of tools, is the assumption that the translation and orientation paths are independent. In reality, the limits on velocity, acceleration and jerk of both paths are not independent, as both the translational and rotational movements are generated by the same robot actuators with (1). The true limits on the path variables s_t and s_r would have the form of a polytope. A way to avoid this issue would be to formulate the path using Lie algebra [57] or Geometric algebra [58], where the rotation and translation could be represented in a single pose path variable. However, in that case, further considerations are required for the robot to move in straight lines in CS. In order to minimise the coupling effect between the rotation and translation, in the context of this work, translation limits are calculated while considering constant rotational velocity, while rotation limits are calculated considering constant translation velocity.

Another limitation of the proposed approach is the assumption of constant robot's capacities for every CS planning iteration. As described in Section V, this can lead to some negative effects such as overshoots and oscillations. In order to overcome this issue, instead of considering only the instantaneous robot's capacity for each trajectory generation execution, it would be necessary to predict the robot's movement capacity along the remaining trajectory. This is a challenging research topic whose results might be applied not just in TAP planning techniques but also in optimal control methods such as MPC.

Finally, the robot's Joint Space (JS) limits (1) are considered constant in time, which is generally not the case. These limits will depend on the robot's actuation limits τ and different dynamical and gravitational effects acting on the robot, as well of the robot's joint state $\{q, \dot{q}\}$. For highly dynamical robot's movements, the available actuation capacity of the robot might reduce some of the limits (1). Therefore, the integration of the robot's actuation limits τ could make the proposed approach more robust and is a promising direction for future research.

X. CONCLUSION

This paper addresses one of the key challenges when it comes to trajectory planning in Cartesian Space (CS): accounting for the robot's changing movement ability (physical

ability to generate movement: velocity, acceleration, etc.) while executing the trajectory. The paper proposes an efficient trajectory planning method that evaluates its abilities in real-time and updates the planned trajectory at each step of trajectory execution. The proposed method creates near time-optimal CS trajectories on-the-fly, not requiring any in advance pre-computation. Additionally, the method is independent of the underlying robot control algorithm, making it suitable for wide range of common robotic systems.

To evaluate the performance of the trajectories calculated by the proposed method, it is compared against a state-of-the-art time-optimal Joint Space (JS) planning method called TOPP-RA [6]. Even though TOPP-RA requires substantial pre-computation and the proposed method plans on-the-fly, the results show that the methods have comparable trajectory execution times: the proposed method having about 5% longer trajectory times on average. Furthermore, the proposed method is compared against a standard CS time-optimal Trapezoidal Acceleration Profile (TAP) planning approach, considering constant CS limits given by the manufacturer. The results show that the proposed method better exploits the robot's movement capacity (capable of planning up with up to 100% of robot's limits), while at the same time achieving a lower tracking error (under 5mm). The comparative study is described in Section VII.

To illustrate the practical application, a mock-up experiment for human-robot collaboration in waste sorting is conducted. The method dynamically plans near time-optimal robot motions in response to the introduction of waste items, showcasing its adaptability and responsiveness in collaborative scenarios, further highlighting its potential in real-world human-robot collaboration.

REFERENCES

- [1] A. Ajoudani, A. M. Zanchettin, S. Ivaldi, A. Albu-Schäffer, K. Kosuge, and O. Khatib, "Progress and prospects of the human-robot collaboration," *Autonomous Robots*, vol. 42, no. 5, pp. 957–975, 2018.
- [2] N. Mansfeld, B. Djellab, J. R. Veuthey, F. Beck, C. Ott, and S. Haddadin, "Improving the performance of biomechanically safe velocity control for redundant robots through reflected mass minimization," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5390–5397, 2017.
- [3] G. Pardo-Castellote and R. Cannon, "Proximate time-optimal algorithm for on-line path parameterization and modification," in *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 2, pp. 1539–1546 vol.2, 1996.
- [4] A. Gasparetto, P. Boscariol, A. Lanzutti, and R. Vidoni, "Trajectory planning in robotics," *Mathematics in Computer Science*, vol. 6, pp. 269–279, Sep 2012.
- [5] J. E. Bobrow, S. Dubowsky, and J. S. Gibson, "Time-optimal control of robotic manipulators along specified paths," *The international journal of robotics research*, vol. 4, no. 3, pp. 3–17, 1985.
- [6] H. Pham and Q.-C. Pham, "A new approach to time-optimal path parameterization based on reachability analysis," *IEEE Transactions on Robotics*, vol. 34, no. 3, pp. 645–659, 2018.
- [7] S. Macfarlane and E. Croft, "Jerk-bounded manipulator trajectory planning: design for real-time applications," *IEEE Transactions on Robotics and Automation*, vol. 19, no. 1, pp. 42–52, 2003.
- [8] R. Haschke, E. Weitnauer, and H. Ritter, "On-line planning of time-optimal, jerk-limited trajectories," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3248–3253, IEEE, 2008.
- [9] T. Kröger and F. M. Wahl, "Online trajectory generation: Basic concepts for instantaneous reactions to unforeseen events," *IEEE Transactions on Robotics*, vol. 26, no. 1, pp. 94–111, 2010.

¹¹For example, the constrained optimisation based controller proposed in Section VI-A is capable of ensuring the adherence to the robot limits (29), while it's ability to cancel the movement in the directions orthogonal to the path depends on PD gains chosen.

IEEE Transactions on Robotics (T-RO) paper, presented at ICRA 2026, Vienna, Austria. Cite as T-RO paper.

- [10] P. Svarny, M. Hamad, A. Kurdas, M. Hoffmann, S. Abdolshah, and S. Haddadin, "Functional mode switching for safe and efficient human-robot interaction," in *2022 IEEE-RAS 21st International Conference on Humanoid Robots (Humanoids)*, pp. 888–894, 2022.
- [11] Y. Fang, J. Hu, W. Liu, Q. Shao, J. Qi, and Y. Peng, "Smooth and time-optimal s-curve trajectory planning for automated robots and machines," *Mechanism and Machine Theory*, vol. 137, pp. 127–153, 2019.
- [12] K. M. Lynch and F. C. Park, *Modern Robotics: Mechanics, Planning, and Control*. USA: Cambridge University Press, 1st ed., 2017.
- [13] C. Guarino Lo Bianco and F. Ghilardelli, "Real-time planner in the operational space for the automatic handling of kinematic constraints," *IEEE Transactions on Automation Science and Engineering*, vol. 11, no. 3, pp. 730–739, 2014.
- [14] A. Skuric, V. Padois, N. Rezzoug, and D. Daney, "On-line feasible wrench polytope evaluation based on human musculoskeletal models: An iterative convex hull method," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 5206–5213, 2022.
- [15] L. Berscheid and T. Kröger, "Jerk-limited real-time trajectory generation with arbitrary target states," *Robotics: Science and Systems XVII*, 2021.
- [16] L. Wenyin and D. Dori, "Incremental arc segmentation algorithm and its evaluation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 4, pp. 424–431, 1998.
- [17] L. Pérez-Gaspar, F. Trujillo-Romero, S. Caballero-Morales, and F. Ramírez-Leyva, "Curve fitting using polygonal approximation for a robotic writing task," in *2015 International Conference on Electronics, Communications and Computers (CONIELECOMP)*, pp. 184–189, 2015.
- [18] J.-C. Perez and E. Vidal, "Optimum polygonal approximation of digitized curves," *Pattern Recognition Letters*, vol. 15, no. 8, pp. 743–750, 1994.
- [19] J. Bobrow, S. Dubowsky, and J. Gibson, "Time-optimal control of robotic manipulators along specified paths," *The International Journal of Robotics Research*, vol. 4, no. 3, pp. 3–17, 1985.
- [20] P. Chiacchio, "Exploiting redundancy in minimum-time path following robot control," in *1990 American Control Conference*, pp. 2313–2318, 1990.
- [21] K. Al Khudir and A. De Luca, "Faster motion on cartesian paths exploiting robot redundancy at the acceleration level," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3553–3560, 2018.
- [22] C. Guarino Lo Bianco and F. Ghilardelli, "A scaling algorithm for the generation of jerk-limited trajectories in the operational space," *Robotics and Computer-Integrated Manufacturing*, vol. 44, pp. 284–295, 2017.
- [23] J. Lee, "A study on the manipulability measures for robot manipulators," in *Proceedings of the IEEE/RJSJ International Conference on Intelligent Robot and Systems. Innovative Robotics for Real-World Applications. IROS '97*, vol. 3, pp. 1458–1465 vol.3, 1997.
- [24] P. Chiacchio, "A new dynamic manipulability ellipsoid for redundant manipulators," *Robotica*, vol. 18, no. 4, p. 381–387, 2000.
- [25] A. Skuric, V. Padois, and D. Daney, "On-line force capability evaluation based on efficient polytope vertex search," in *IEEE International Conference on Robotics and Automation*, (Xi'an, China), May 2021.
- [26] T. Yoshikawa, "Manipulability of robotic mechanisms," *The International Journal of Robotics Research*, vol. 4, no. 2, pp. 3–9, 1985.
- [27] F. Fahimi, *Redundant Manipulators*, pp. 1–36. Boston, MA: Springer US, 2009.
- [28] A. D. 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.
- [29] C. D. Meyer, *Matrix Analysis and Applied Linear Algebra*. SIAM, 2001.
- [30] S. Vajda and S. I. Gass, "Linear programming. methods and applications," *OR*, vol. 15, no. 4, p. 412, 1964.
- [31] J. L. Gearhart, K. L. Adair, J. D. Durfee, K. A. Jones, N. Martin, and R. J. Detry, "Comparison of open-source linear programming solvers," 10 2013.
- [32] S. Rubrecht, V. Padois, P. Bidaud, and M. de Broissia, "Constraints compliant control: Constraints compatibility and the displaced configuration approach," in *2010 IEEE/RJSJ International Conference on Intelligent Robots and Systems*, pp. 677–684, 2010.
- [33] L. Joseph, J. K. Pickard, V. Padois, and D. Daney, "Online velocity constraint adaptation for safe and efficient human-robot workspace sharing," in *2020 IEEE/RJSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 11045–11051, 2020.
- [34] L. Moriello, L. Biagiotti, C. Melchiorri, and A. Paoli, "Manipulating liquids with robots: A sloshing-free solution," *Control Engineering Practice*, vol. 78, pp. 129–141, 2018.
- [35] Universal Robots, *Universal Robots e-Series User Manual*, 2022.
- [36] Franka Emika, "Panda robot datasheet - franka control interface," 2022.
- [37] K. D. Nguyen, T.-C. Ng, and I.-M. Chen, "On algorithms for planning s-curve motion profiles," *International Journal of Advanced Robotic Systems*, vol. 5, no. 1, p. 11, 2008.
- [38] D. Constantinescu and E. A. Croft, "Smooth and time-optimal trajectory planning for industrial manipulators along specified paths," *Journal of Robotic Systems*, vol. 17, no. 5, pp. 233–249, 2000.
- [39] F. Pfeiffer and R. Johanni, "A concept for manipulator trajectory planning," *IEEE Journal on Robotics and Automation*, vol. 3, no. 2, pp. 115–123, 1987.
- [40] V. Utkin and H. Lee, "Chattering problem in sliding mode control systems," *IFAC Proceedings Volumes*, vol. 39, no. 5, p. 1, 2006. 2nd IFAC Conference on Analysis and Design of Hybrid Systems.
- [41] F. Garelli, L. Gracia, A. Sala, and P. Albertos, "Sliding mode speed auto-regulation technique for robotic tracking," *Robotics and Autonomous Systems*, vol. 59, no. 7, pp. 519–529, 2011.
- [42] C. Guarino Lo Bianco and F. Ghilardelli, "A discrete-time filter for the generation of signals with asymmetric and variable bounds on velocity, acceleration, and jerk," *IEEE Transactions on Industrial Electronics*, vol. 61, no. 8, pp. 4115–4125, 2014.
- [43] C. Guarino Lo Bianco and F. Ghilardelli, "Techniques to preserve the stability of a trajectory scaling algorithm," in *2013 IEEE International Conference on Robotics and Automation*, pp. 870–876, 2013.
- [44] B. Siciliano, "Kinematic control of redundant robot manipulators: A tutorial," *Journal of Intelligent and Robotic Systems*, vol. 3, pp. 201–212, Sep 1990.
- [45] J. Haviland and P. Corke, "A purely-reactive manipulability-maximising motion controller," 2020.
- [46] J. Solà, J. Deray, and D. Atchuthan, "A micro lie theory for state estimation in robotics," 2021.
- [47] J. P. Merlet, "Jacobian, manipulability, condition number, and accuracy of parallel robots," *Journal of Mechanical Design*, vol. 128, pp. 199–206, 06 2005.
- [48] K. Dufour and W. Suleiman, "On maximizing manipulability index while solving a kinematics task," *Journal of Intelligent & Robotic Systems*, vol. 100, no. 1, pp. 3–13, 2020.
- [49] J. Carpentier, N. Mansard, F. Valenza, J. Mirabel, G. Saurel, and R. Budhiraja, "Pinocchio - Efficient and versatile Rigid Body Dynamics algorithms," July 2021.
- [50] A. O. Makhorin, "Glpk - gnu linear programming kit, version 4.0." Online: <http://www.gnu.org/software/glpk/>, 2022.
- [51] P. Corke and J. Haviland, "Not your grandmother's toolbox—the robotics toolbox reinvented for python," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 11357–11363, IEEE, 2021.
- [52] M. Koskinopoulou, F. Raptopoulos, G. Papadopoulos, N. Mavrakis, and M. Maniadakis, "Robotic waste sorting technology: Toward a vision-based categorization system for the industrial robotic separation of recyclable waste," *IEEE Robotics & Automation Magazine*, vol. 28, no. 2, pp. 50–60, 2021.
- [53] G. Hassan, M. Gouttefarde, A. Chemori, P.-E. Hervé, M. E. Rafei, C. Francis, and D. Sallé, "Time-optimal pick-and-throw s-curve trajectories for fast parallel robots," *IEEE/ASME Transactions on Mechatronics*, vol. 27, no. 6, pp. 4707–4717, 2022.
- [54] R. Sarc, A. Curtis, L. Kandlbauer, K. Khodier, K. Lorber, and R. Pomberger, "Digitalisation and intelligent robotics in value chain of circular economy oriented waste management – a review," *Waste Management*, vol. 95, pp. 476–492, 2019.
- [55] S. Kleff, A. Meduri, R. Budhiraja, N. Mansard, and L. Righetti, "High-frequency nonlinear model predictive control of a manipulator," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7330–7336, 2021.
- [56] M. Eckhoff, R. J. Kirschner, E. Kern, S. Abdolshah, and S. Haddadin, "An mpc framework for planning safe & trustworthy robot motions," in *2022 International Conference on Robotics and Automation (ICRA)*, pp. 4737–4742, 2022.
- [57] N. Torres Alberto, A. Skuric, L. Joseph, V. Padois, and D. Daney, "Model Predictive Control for robots adapting their task space motion online." working paper or preprint, Apr. 2023.
- [58] T. Löw and S. Calinon, "Geometric algebra for optimal control with applications in manipulation tasks," *IEEE Transactions on Robotics*, vol. 39, no. 5, pp. 3586–3600, 2023.

IEEE Transactions on Robotics (T-RO) paper, presented at ICRA 2026, Vienna, Austria. Cite as T-RO paper.



Antun Skuric is a robotics researcher, currently with Pollen Robotics and Hugging Face. He holds a PhD in human-robot interaction from the University of Bordeaux. During his doctoral program he was part of the Auctus team at INRIA Bordeaux. His research focuses on algorithmic and formal methods for estimating physical abilities of humans and robots in real-time. The research has double aim. On the one hand, exploiting more optimally the physical abilities of robots. On the other hand, by for example evaluating physical abilities of humans

based on their biomechanical models, adapting their tasks and levels of robotic assistance to their needs. More recently his research focus has shifted toward exploring the role of robotics in sustainable development and assessing its different impacts.



Nicolas Torres Alberto holds a Ph.D. in Robotics (University of Bordeaux, France) and an M.Sc. in Computer Vision and Robotics (Telecom Physique Strasbourg). His work focuses on software engineering for critical systems and R&D, spanning robotics, automotive, and aerospace. His doctoral research, conducted in a collaborative project between the Auctus team at Inria (Bordeaux) and Stellantis, focused on using predictive control approaches to ensure safe human-robot collaboration in dynamic industrial contexts. The core contribution was a

novel constrained optimization approach, integrating Lie algebra for real-time trajectory optimization within a Model Predictive Control (MPC) framework. This technique, available as an open-source motion generation and control library, achieved highly reactive robot manipulator behavior and safe replanning, with guaranteed satisfaction of all operational constraints. He maintains a strong research interest in synthesizing hybrid control strategies that integrate deterministic, real-time control logic with machine learning methods to enhance robotic autonomy and robustness.



Lucas Joseph Lucas Joseph is a Research Engineer with Inria, Centre Bordeaux – Sud-Ouest, where he is a member of the Auctus team. His work focuses on real-time control and motion generation for collaborative robotic systems. He develops software frameworks for constrained-optimization control and model predictive control, with contributions spanning high-frequency SE(3) trajectory replanning, on-line adaptation in dynamic human–robot environments, and shared-autonomy strategies. His research interests include human–robot interaction, predictive control, and integrated control–planning architectures for safe collaboration.

Vincent Padois Vincent Padois is a Senior Research Scientist at Inria within the Auctus team (Talence, France), a position he has held since November 2020. From 2007 to October 2020, he was a tenured Associate Professor of Robotics at the Institute of Intelligent Systems and Robotics (ISIR), Sorbonne Université, Paris, France. Prior to joining ISIR, he was a postdoctoral researcher from 2006 to 2007 at the Stanford Artificial Intelligence Laboratory, working in the group of Prof. O. Khatib. He received his Ph.D. in 2005 from INP Toulouse under the supervision of Pr. P. Chiron and J.-Y. Fourquet at the Laboratoire Génie de Production (LGP), Tarbes, France. He also holds an M.Sc. in Systems and Automatic Control from INP Toulouse and an M.Eng. from the École Nationale d'Ingénieurs de Tarbes, both obtained in 2001. His research focuses on the design of control architectures enabling advanced robotic behaviours in constrained and dynamic environments, with a particular emphasis on robotic assistance for humans at work.



David Daney David Daney is a Senior Researcher at Inria. He received his Ph.D. in Robotics from the University of Nice Sophia Antipolis in 2000. From 2000 to 2002, he held visiting research positions at McGill University, Université Laval, and Rutgers University. In 2003, he joined the Inria research center in Sophia Antipolis as a permanent researcher. His research interests include the modeling, identification, and design of robotic manipulators. He has made substantial contributions to the development and dissemination of interval analysis in robotics. In

2013, he joined the Inria Centre at the University of Bordeaux, where his research expanded to encompass collaborative robotics, human motion analysis, and the design of robot behaviors for physical and cognitive human–robot interaction. He is the author of over 150 peer-reviewed scientific publications in these areas. Since 2017, he has been leading the AUCTUS research team, which focuses on the design, control, and evaluation of interactive robotic systems. He is also actively involved in the scientific coordination of national and European research initiatives and contributes to research governance within Inria.