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

Antun Skuric, Nicolas Torres Alberto, Lucas Joseph, Vincent Padois, David Daney

▶ **To cite this version:**

Antun Skuric, Nicolas Torres Alberto, Lucas Joseph, Vincent Padois, David Daney. Online approach to near time-optimal task-space trajectory planning. 2024. hal-04576076

HAL Id: hal-04576076

https://inria.hal.science/hal-04576076

Preprint submitted on 15 May 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

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

Antun Skuric[1], Nicolas Torres Alberto[1], Lucas Joseph[1], Vincent Padois[1] and David Daney[1]

***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 real-time responsiveness to dynamic, unstructured spaces. Traditional reactive trajectory planning methods use simplified robot models, while methods exploiting robots' abilities have high computational complexity or lack reactivity. This paper introduces an approach for real-time trajectory planning that exploits the robot's full motion abilities. 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 approach is compared to state-of-the-art methods, showing comparable execution time with better movement capacity utilization and lower tracking error. 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 a 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 capacity corresponds to the minimum time criteria, resulting in time-optimal trajectories [4].

[1]AUCTUS Team, Inria, Talence, France
firstname.lastename@inria.fr

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

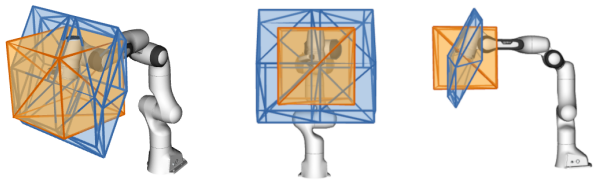

Fig. 1. End-effector linear velocity limits as computed using the fixed Cartesian limits provided by the manufacturer (orange) and using an estimation based on the joint space velocity limits and the current configuration of the robot (blue) (scale: 1 m.s^{-1}/ 10 cm). The true robot's velocity capacity are in some directions higher and in the others significantly lower than the manufacturer's limits, depending on the configuration.

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 real-time changes in the trajectory and the task. To account for these changes the trajectory needs to be adapted in real-time, however time-optimal approaches have long execution times due to their computational complexity, making such implementations unpractical.

On the other end of the spectrum, the real-time capable trajectory planning approaches often abstract the robot as an ideal generator of movements without accounting for its true motion capacity. These methods are commonly characterised within the task space of the robot, encompassing both the positions and orientations of the robot's coordinate system (Cartesian Space (CS)). With this simplification, they have short computation times and can quickly adapt to any changes in the robot's path, environment or the task itself. Such algorithms, as proposed by Macfarlane and Croft [7], Haschke et al. [8] or Svarny et al. [9], are often based on S-curve or trapezoidal profile trajectories [10], producing smooth trajectories that respect a set of fixed velocity, acceleration, jerk and sometimes even snap (jerk derivative) limits enforced by the task itself as well as by the robot's movement capacities. To render the problem solvable in a reactive way, unlike the time-optimal approaches mentioned earlier, these limits are assumed

to be constant [11]. Yet, they are in fact dependent on the robot state and can therefore vary significantly during the trajectory execution. Therefore such approaches are not capable of fully exploiting the robot's capacity, resulting in trajectories that might underestimate or sometimes even overestimate them. Figure 1 shows an example of this effect on a Franka Emika Panda collaborative robot.

To bridge this gap, several approaches have been proposed recently, aiming to fully exploit the robot's movement capacity while allowing for reactive adaptations online. These methods often decouple the time-optimality and reactivity problems. For instance, Palleschi et al. [12] proposed a decoupling method that plans for time-optimal robot trajectories offline, while the reactivity is ensured using an online re-planning approach in order to guarantee human safety. However, such method lacks the flexibility to accommodate real-time changes in the robot's task, which is often necessary in dynamic environments. A different decoupling approach by Springer et al. [13], proposes to use Dynamic Motion Primitives (DMP) to allow for real-time task changes. Their method calculates the DMP models of the time-optimal point-to-point trajectories in advance on a large dataset. Then, these DMP-based trajectories are used in runtime to allow for the real-time changes in the task. Nonetheless, the DMP trajectories only approximate time-optimal solutions and might not accurately reflect the true motion capabilities of the robot, leading to potential under or overestimation.

The family of approaches based on optimal control methods like Model Predictive Control (MPC) [14] offers a promising avenue for generating reactive and time-optimal trajectories in real-time without the need for decoupling. Instead of calculating the entire time-optimal trajectory in advance, MPC approaches determine the optimal action to be executed by the robot one time step at a time, given its current state, the target state, and the predictions of its future states obtained using its model. As these approaches re-evaluate the optimal robot's action at each time step, they are capable of exploiting the robot's changing movement capabilities as well as quickly adapting to the changes in the task while reacting to changes in the environment [15][16]. However, as robot models are highly non-linear, the implementation of these methods remains relatively complex and challenging, often relying on non-linear optimisation techniques [17][18] or on different degrees of model simplifications [19].

In the case of robotic manipulators, MPC algorithms often lack the computational efficiency and robustness for real-time operation. However, inspired by them, this paper proposes an efficient trajectory planning method designed to generate near time-optimal and responsive trajectories based on real-time re-planning. At each time-step of the trajectory execution, the proposed approach 1) evaluates the robot's CS movement capacity limits using efficient tools from polytope algebra 2) uses these limits to re-plan the time-optimal trajectory on the remaining path, based on the computationally efficient Trapezoidal Acceleration Profile (TAP) [11]. 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 without requiring any pre-computation. Additionally, the real-time re-planning enables reacting to potential changes in the robot's trajectory induced by its environment, making it well-suited for dynamic contexts, such as human-robot collaboration.

To evaluate the performance of the proposed method, it is compared against the state-of-the-art offline method TOPP-RA [6], which plans in JS, as well as against a standard reactive approach, based on TAP planning in CS. Simulation based analysis, on a Franka Emika Panda robot, shows comparable or even shorter trajectory execution times compared to TOPP-RA, while at the same time producing faster and more precise trajectories than the reactive TAP approach. Furthermore, the paper showcases the reactivity of the proposed approach through a mock-up experiment of collaborative waste sorting. The method generates near time-optimal and adaptable trajectories for the robot, highlighting its practical applicability.

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. Problem statement: Ability aware trajectory planning

Time-optimal trajectory planning consists in finding the robot's motions with the highest possible speeds along pre-defined paths, while respecting the robot's inherent motion ability and potentially user-defined task constraints. 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.

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($m = 6$). 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 [20][21].

Traditional approaches to the time-optimal trajectory planning consist in transforming the path to the JS, where they find the fastest movements of the robot's joints following

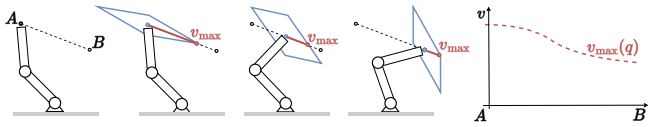

Fig. 2. Illustrations of the evolution of the CS linear velocity capacity while following a CS straight line path, from point A to B. The robot's CS velocity capacity polytope is shown in blue, while its capacity to generate velocity in the path direction is shown in red. The path is shown in black dashed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

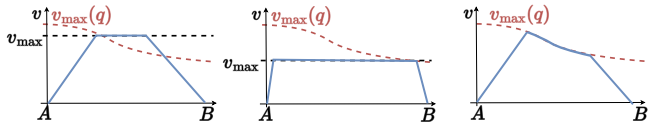

Fig. 3. Comparison of three trajectory planning strategies for CS, considering the robot's velocity capacity limits. Left and middle figures depict methods assuming a fixed velocity capacity, leading to overestimation (left) or significant underestimation (middle) of the robot's abilities. The right 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.

the path and respecting the actuator limits [22]. 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 main limitation of this approach appears when the JS is higher dimensional than the CS ($n > m$), making the mapping between the CS and JS not unique. In practice this means that there are multiple JS paths that accomplish the same CS path. Each one of these JS paths has different properties which makes finding the optimal one a challenging problem on its own and one with a substantial computational cost. Additionally, such methods require choosing the behaviour of the robot's redundant degrees of freedom in-advance, even though they do not participate in the movement generation.

The method proposed in this paper opts for a complementary approach, consisting in transforming the robot's JS actuator limits to the CS and planning the time-optimal trajectory in CS directly. Such method results in the CS trajectory which respects 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]. The robot's movement polytopes are highly dependant on its state and can change significantly during the execution of the trajectory. Therefore, the robot's state-dependent CS movement capacity presents one of the main challenges of time-optimal trajectory planning in CS. Figure 2 illustrates the robot's state-dependent CS velocity capacity while executing the CS trajectory.

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, as the robot's movement capacity depends on its joint states, evaluating its capacities along the path in advance would require transforming the CS path to JS. Such approach would lose most of the benefits of CS trajectory planning: resulting in high computational cost, requiring making choices about the appropriate JS path and fixing the robot's redundant degrees of freedom.

Rather than trying to account for the robot's changing movement capacity in the offline planning stage, this paper proposes a method that adapts to the changes on-the-fly. Section III introduces the real-time capable method for evaluating the robot's movement capacity limits in the path direction, leveraging efficient polytope algebra tools [26]. Section IV then describes the proposed re-planning approach, where at each time-step, the updated robot's movement capacity limits are used to recalculate the time-optimal trajectory on the remaining path, based on the efficient Trapezoidal Acceleration Profile (TAP) planning method [27].

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 ranges[1] of the feasible joint positions $\boldsymbol{q} \in \mathbb{R}^n$, velocity $\dot{\boldsymbol{q}} \in \mathbb{R}^n$, acceleration $\ddot{\boldsymbol{q}} \in \mathbb{R}^n$ and jerk $\dddot{\boldsymbol{q}} \in \mathbb{R}^n$

$$\begin{aligned} \dddot{\boldsymbol{q}} \in [\dddot{\boldsymbol{q}}_{min}, \dddot{\boldsymbol{q}}_{max}],\ \ddot{\boldsymbol{q}} \in [\ddot{\boldsymbol{q}}_{min}, \ddot{\boldsymbol{q}}_{max}], \\ \dot{\boldsymbol{q}} \in [\dot{\boldsymbol{q}}_{min}, \dot{\boldsymbol{q}}_{max}],\ \boldsymbol{q} \in [\boldsymbol{q}_{min}, \boldsymbol{q}_{max}] \end{aligned} \tag{1}$$

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

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

Where $J(\boldsymbol{q}) \in \mathbb{R}^{m \times n}$ is the Jacobian matrix and $\dot{J}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \in \mathbb{R}^{m \times n}$ and $\ddot{J}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}) \in \mathbb{R}^{m \times n}$ 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 state $\{\boldsymbol{q}_k, \dot{\boldsymbol{q}}_k, \ddot{\boldsymbol{q}}_k\}$, 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.

[1]The true Joint Space (JS) acceleration $\ddot{\boldsymbol{q}}$ and jerk $\dddot{\boldsymbol{q}}$ 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.

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

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

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

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

where $\boldsymbol{b}_a, \boldsymbol{b}_j \in \mathbb{R}^m$ are the bias acceleration and jerk[2] produced by the effect of current joint velocity $\dot{\boldsymbol{q}}_k$ and acceleration $\ddot{\boldsymbol{q}}_k$, while $\boldsymbol{b}_v \in \mathbb{R}^m$ is a zero vector ($\boldsymbol{b}_v = \boldsymbol{0}$), added to unify the formulations.

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

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

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

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, 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 $\boldsymbol{c}$, pointing in the direction of the path can be found. Then any CS vector $\boldsymbol{y} \in \mathbb{R}^m$ can be projected on the path direction using the scalar product

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

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

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

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

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

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

whereas the minimum can be found by minimising it.

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

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

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

Integrating this relationship into the LP formulation (8) yields

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

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

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

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

It is worth noting that the robot's movement capacity limits, as expressed in (11), assume that the robot's velocity, acceleration and jerk limits are independent and do not influence one another, which is not usually the case (ex. if one of the robot's joints reaches its maximal velocity, it will no longer be able to accelerate in that direction). However, these limits are used for Trapezoidal Acceleration Profile (TAP) planning which inherently accounts for this issue and finds the trajectory respecting all the set constraints at the same time.

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

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 $\dot{\boldsymbol{x}}$ to minimise the potential impact forces due to the robot's kinetic energy [2][32]. On the other hand, when designing trajectories of robotic manipulation of liquids, in order to prevent its sloshing or spilling, acceleration $\ddot{\boldsymbol{x}}$ continuity needs to be ensured [33], which in term corresponds to introducing CS jerk $\dddot{\boldsymbol{x}}$ limits.

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

$$\mathcal{C} = \{\boldsymbol{y} \mid A\boldsymbol{y} \leq \boldsymbol{b}\} \tag{12}$$

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

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

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 [34][35].

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

$$\begin{aligned} \dot{\boldsymbol{x}} \in [\alpha_v\dot{\boldsymbol{x}}_{min}, \alpha_v\dot{\boldsymbol{x}}_{max}],\ \ddot{\boldsymbol{x}} \in [\alpha_a\ddot{\boldsymbol{x}}_{min}, \alpha_a\ddot{\boldsymbol{x}}_{max}], \\ \dddot{\boldsymbol{x}} \in [\alpha_j\dddot{\boldsymbol{x}}_{min}, \alpha_j\dddot{\boldsymbol{x}}_{max}] \end{aligned} \tag{14}$$

allowing the robots velocity $\dot{\boldsymbol{x}}$, acceleration $\ddot{\boldsymbol{x}}$ and jerk $\dddot{\boldsymbol{x}}$ 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 $\{\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}\}$, 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

$$\begin{aligned} \dot{\boldsymbol{q}} \in [\alpha_v\dot{\boldsymbol{q}}_{min}, \alpha_v\dot{\boldsymbol{q}}_{max}],\ \ddot{\boldsymbol{q}} \in [\alpha_a\ddot{\boldsymbol{q}}_{min}, \alpha_a\ddot{\boldsymbol{q}}_{max}], \\ \dddot{\boldsymbol{q}} \in [\alpha_j\dddot{\boldsymbol{q}}_{min}, \alpha_j\dddot{\boldsymbol{q}}_{max}] \end{aligned} \tag{15}$$

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 [11, Chapter 9.2.2.2][36][27], 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) [11, 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 [37][38].

When is 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 [11, 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

$$\dot{s} \in [\dot{s}_{min}, \dot{s}_{max}],\ \ddot{s} \in [\ddot{s}_{min}, \ddot{s}_{max}],\ \dddot{s} \in [\dddot{s}_{min},\ \dddot{s}_{max}] \tag{16}$$

as well as the starting and end conditions

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

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 $\mathcal{P}(s)$[22]. The translation component of the path can then be expressed as

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

where X_a^t and X_b^t are the translation parts of the poses X_a and X_b and $\boldsymbol{u}_t$ 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 $\boldsymbol{u}_r$. Then the geometric path corresponding to the orientation $\mathcal{P}_o(s_r)$ can be written as

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

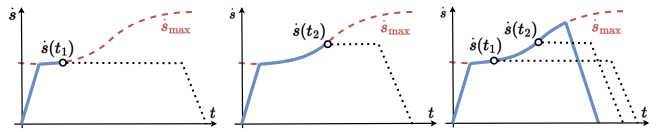

Fig. 4. Velocity curve, updated in the real time with instantaneous maximal values. At each time-step t_k during the trajectory execution (ex. t_1 and t_2) the TAP planning calculates the remaining trajectory considering that maximal velocity $\dot{s}_{max}$ constant till the end of the movement (black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

$\mathcal{P}_r(s_r)$ 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 $\mathcal{P}_t$

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

and optimal CS velocity and acceleration can be found

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

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

$$\dot{s}_i = \boldsymbol{c}_i^T \dot{\boldsymbol{x}}, \quad \ddot{s}_i = \boldsymbol{c}_i^T \ddot{\boldsymbol{x}}, \qquad i = \{r, t\} \tag{22}$$

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

$$\boldsymbol{c}_t = [\boldsymbol{u}_t^T, \boldsymbol{0}_{3\times1}]^T, \qquad \boldsymbol{c}_r = [\boldsymbol{0}_{3\times1}, \boldsymbol{u}_r^T]^T \tag{23}$$

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 $\dddot{s}$, acceleration $\ddot{s}$ and velocity $\dot{s}$ 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, while Algorithm 1 shows its pseudo-code.

As shown in Algorithm 1, in each time step k, corresponding to the moment in time t_k and the robot's state $\{\boldsymbol{q}_k, \dot{\boldsymbol{q}}_k, \ddot{\boldsymbol{q}}_k\}$, the instantaneous limits on the path variables $s_r(t)$ and $s_t(t)$ are calculated using the method described in Section III-A.

Algorithm 1 One step of the proposed approach

Require: Path lengths d, θ and directions $\boldsymbol{c}_r$, $\boldsymbol{c}_t$ (18,19,23)
Require: Current robot state $(\boldsymbol{q}_k, \dot{\boldsymbol{q}}_k, \ddot{\boldsymbol{q}}_k)$, $(X_k, \dot{\boldsymbol{x}}_k, \ddot{\boldsymbol{x}}_k)$
$s_{t,k}, s_{r,k} \leftarrow X_k$ (18 - 19)
$\dot{s}_{t,k}, \ddot{s}_{t,k}, \dot{s}_{r,k}, \ddot{s}_{r,k} \leftarrow \dot{\boldsymbol{x}}_k, \ddot{\boldsymbol{x}}_k$ (22)
$(\dot{s}_{t,min}, \dot{s}_{t,max}, \ldots) \leftarrow$ UpdateLimits$(\boldsymbol{c}_t, \boldsymbol{q}_k, \dot{\boldsymbol{q}}_k, \ddot{\boldsymbol{q}}_k)$ (11)
$(\dot{s}_{r,min}, \dot{s}_{r,max}, \ldots) \leftarrow$ UpdateLimits$(\boldsymbol{c}_r, \boldsymbol{q}_k, \dot{\boldsymbol{q}}_k, \ddot{\boldsymbol{q}}_k)$ (11)
d_k, $\theta_k \leftarrow$ Update remaining path length (18 - 19)
$(s_{t,0}, s_{r,0}, \ldots) \leftarrow$ Update initial conditions (24)
$(s_t, s_r, \dot{s}_t, \dot{s}_r, \ldots) \leftarrow$ TAP$(d_k, \theta_k, s_{t,0}, s_{r,0}, \dot{s}_{t,0}, \dot{s}_{r,0}, \ldots, \dot{s}_{t,min}, \dot{s}_{r,min}, \ddot{s}_{t,min}, \ddot{s}_{r,min}, \ldots)$
$X_d \leftarrow s_{t,k+1}, s_{r,k+1}$ (20)
$\dot{\boldsymbol{x}}_d, \ddot{\boldsymbol{x}}_d \leftarrow \dot{s}_{t,k+1}, \dot{s}_{r,k+1}, \ddot{s}_{t,k+1}, \ddot{s}_{r,k+1}$ (21)
return $X_d, \dot{\boldsymbol{x}}_d, \ddot{\boldsymbol{x}}_d$

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 $\boldsymbol{c}_t$ and the rotation limits are calculated using $\boldsymbol{c}_r$, 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

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

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 $\dot{\boldsymbol{x}}_d$ and acceleration $\ddot{\boldsymbol{x}}_d$, based on the optimal path variables $s_{t,k+1}, s_{r,k+1}, \ldots, \ddot{s}_{t,k+1}, \ddot{s}_{r,k+1}$ 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 for this approach, as it is highly efficient in computing optimal trajectories, allowing for real-time execution.

V. Compensating for Real-Time TAP 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 oscillations and an overshoot due to the switching between the modes of braking and accelerating. Section V-A describes the overshoot effect due to the robot's decreasing braking capacity

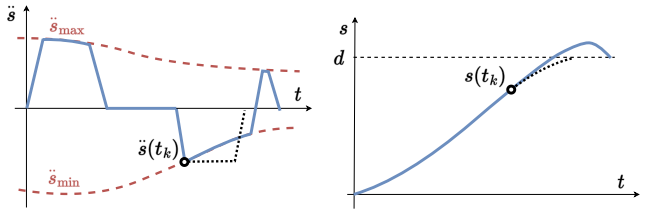

Fig. 5. Figure shows the effect of diminishing braking capacity towards the end of the trajectory, where the end result is an overshoot.

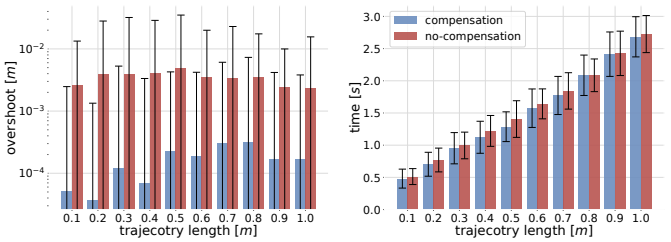

Fig. 6. Plots showing the comparison of the average overshoot (left) and average execution time (right) of the proposed method with and without overshoot compensation. For each length ranging from $d = 0.1$ to $1m$, means and variances are calculated over 100 random trajectories.

towards the end of the trajectory, while Section V-B describes the oscillation 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.

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 5, 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.

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

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 $\{\boldsymbol{q}, \dot{\boldsymbol{q}}\}$, finding the minimal braking capacity (25) requires either knowing the exact robot's Joint Space (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 based on the sampling the remaining CS path (from the current pose X_k to the final pose X_b) into N poses $X_i \in SE(3)$. The corresponding configuration $\boldsymbol{q}_i$ is computed through inverse kinematics, choosing the solution closest to the current configuration $\boldsymbol{q}_k$. Finally, the prediction of (25) can be found as the minimal braking capacity among all the sampled ones $\ddot{s}_{i,min}$

$$\ddot{s}_{min} = \max\{\ddot{s}_{0,min},\ \ldots,\ \ddot{s}_{N,min}\} \tag{26}$$

Experimental validation of compensation parameters: In the experiments the remaining trajectory is sampled with $N = 2$ points, corresponding to the current position X_k in the time step t_k and the final position at the end of the trajectory X_T. Joint velocity $\dot{\boldsymbol{q}}_T$ and acceleration $\ddot{\boldsymbol{q}}_T$ are considered to be $\dot{\boldsymbol{q}}_T, \ddot{\boldsymbol{q}}_T = \boldsymbol{0}$, as the robot will come to a stop at the target X_T. Finally the braking capacity used for TAP planning in the step t_k is the minimum of the two

$$\ddot{s}_{min} = \max\{\ddot{s}_{k,min},\ \ddot{s}_{T,min}\} \tag{27}$$

This method (27) offers the added advantage of efficient implementation, requiring evaluation of the robot's final braking capacity $\ddot{s}_{T,min}$ only once per trajectory execution.

Figure 6 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 seem to have a negative impact on the trajectory execution time. On the contrary, as shown in the Figure 6, the experimental results 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.

B. Increasing braking capacity: Oscillations effect

When the robot's braking capacity increases along the trajectory, scenario shown on Figure 7, the proposed method can result in oscillatory 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. The repetitions of this sequence create the oscillations of the planed acceleration profile and produces jerky trajectories.

The effect of these oscillations is greatly amplified as the robot's braking capacity $\ddot{s}_{min}$, in some cases, is inversely proportional to the robot's current joint velocity $\dot{\boldsymbol{q}}_k$ through the bias term $\boldsymbol{b}_a = \dot{J}(\boldsymbol{q}_k, \dot{\boldsymbol{q}}_k)\dot{\boldsymbol{q}}_k$. In such cases, decreasing

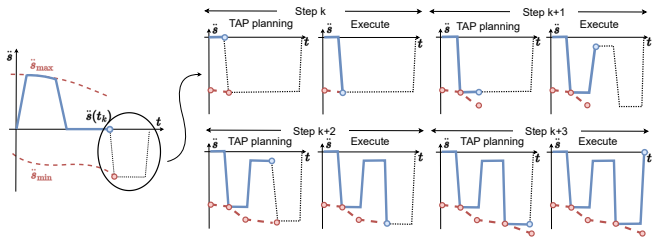

Fig. 7. The figure shows the braking stage of the TAP planning trajectory with the effect of increasing braking capacity towards the end of the trajectory. The executed acceleration $\ddot{s}$ is shown in blue, the planned one is shown in black dotted lines and the maximal braking limit is shown in red dashed lines. Due to, in some cases, inverse proportional coupling between the robot's velocity $\dot{\boldsymbol{q}}$ and the braking capacity $\ddot{s}_{min}$, the more the robot brakes, the more its velocity $\dot{\boldsymbol{q}}$ decreases and the more the braking capacity increases. This effect produces an oscillatory behaviour, as shown in the sequence of figures. The robot starts braking in time step k. Because it was braking, its braking capacity increased in step $k+1$. With the increased braking capacity, TAP planning decides to decrease braking in step $k+1$ as it can brake stronger later.

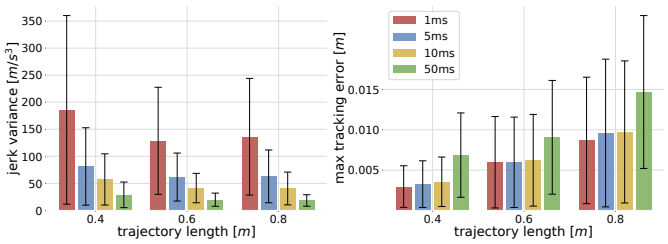

Fig. 8. Plots showing the comparison of the jerk variance (left) and tracking error (right) of the proposed method with and without oscillation compensation, for trajectory lengths $d = 0.4$, 0.6 and $0.8m$. Means and variances are calculated over 100 random trajectories. Four down-sampling times are compared $\Delta t_p = 1, 5, 10$ and 50ms.

joint velocity $\dot{\boldsymbol{q}}_k$ decreases the bias term $\boldsymbol{b}_a$ and increases the braking capacity $\ddot{s}_{min}$. This effect creates a closed loop behaviour, where the more the robot brakes, the more its braking capacity increases, thus inducing the described switching behaviour. Figure 7 illustrates few time steps of such oscillatory behaviour.

In order to smooth the acceleration profile and reduce the effect of coupling introduced by the bias $\boldsymbol{b}_a$, a simple strategy is proposed which consists 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 $t \in [t_k, t_k + \Delta t_p]$.

$$s(t) = s_k + \frac{s_{k+1} - s_k}{\Delta t_p}(t - t_k), \quad t \in [t_k, t_k + \Delta t_p] \tag{28}$$

The same linear interpolation can be applied to the velocity $\dot{s}$ and acceleration $\ddot{s}$

$$\dot{s}(t) = \dot{s}_k + \frac{\dot{s}_{k+1} - \dot{s}_k}{\Delta t_p}(t - t_k), \;\; \ddot{s}(t) = \ddot{s}_k + \frac{\ddot{s}_{k+1} - \ddot{s}_k}{\Delta t_p}(t - t_k)$$

where s_k,$\dot{s}_k$ are $\ddot{s}_k$ are path position, velocity and acceleration in the current step t_k and s_{k+1},$\dot{s}_{k+1}$ are $\ddot{s}_{k+1}$ are the their optimal values in the next planning step $t_{k+1} = t_k + \Delta t_p$ calculated by the TAP planning.

Between the TAP planning steps, the robot's movement capacity is considered constant and in that way the high-frequency oscillations induced by the bias term $\boldsymbol{b}_a$ are filtered. Furthermore, the longer the time between the planning Δt_p, the more the effects of the coupling are reduced. On the other hand, the robot's movement capacity is considered constant between the planning steps Δt_p. Since the robot's movement capacity can decrease between planning steps, the planned trajectory can potentially overestimate the robot's capacity in this period. This overestimation can therefore increase the robot's tracking error due to its inability to follow the planned trajectory. Therefore, when implementing this oscillation 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.

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 Franka Emika 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 8 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.

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 [35]. These values are publicly available[3] and are listed in Table I.

TABLE I
Franka Emika Panda robot kinematic limits[3]. 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
$\dot{\boldsymbol{q}}_{max}$ $[rad/s]$	2.17	2.17	2.17	2.17	2.61	2.61	2.61
$\ddot{\boldsymbol{q}}_{max}$ $[rad/s^2]$	15	7.5	10	12.5	15	20	20
$\dddot{\boldsymbol{q}}_{max}$ $[krad/s^3]$	7.5	3.75	5.0	6.25	7.5	10.0	10.0

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

[3]Full datasheet available at: https://frankaemika.github.io/docs/control_parameters.html

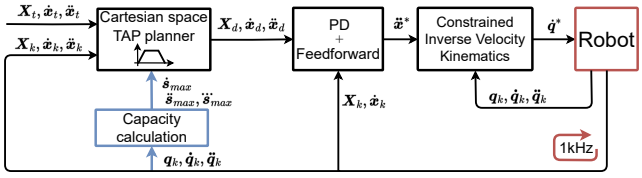

Fig. 9. Proposed method schematic overview in the context of the robot control, elements in blue present the extension of a standard CS based TAP planning. Both the control loop and trajectory planning run at 1kHz.

A. Robot control architecture

The schematic diagram of the proposed approach within the robot control paradigm is shown on Figure 9. Given the robot's current state $\{\boldsymbol{q}_k, \dot{\boldsymbol{q}}_k, \ddot{\boldsymbol{q}}_k\}$ in step k, the approach first determines the robot's current CS movement capacity in the path direction using the approach 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,$\dot{\boldsymbol{x}}_k$,$\ddot{\boldsymbol{x}}_k$ and the target state X_t,$\dot{\boldsymbol{x}}_t$,$\ddot{\boldsymbol{x}}_t$, 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,$\dot{\boldsymbol{x}}_d$,$\ddot{\boldsymbol{x}}_d$ 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.

$$\ddot{\boldsymbol{q}}_{opt} = \arg\min_{\ddot{\boldsymbol{q}}} \overbrace{||\ddot{\boldsymbol{x}}^* - J\ddot{\boldsymbol{q}} - \dot{J}\dot{\boldsymbol{q}}_k||^2}^{\text{trajectory tracking}} + \overbrace{\omega_r||\ddot{\boldsymbol{q}}_r - \ddot{\boldsymbol{q}}||^2}^{\text{regularisation}} \quad (29)$$
$$\text{s.t.} \quad \ddot{\boldsymbol{q}} \in [\ddot{\boldsymbol{q}}_{lb}(\boldsymbol{q}_k, \dot{\boldsymbol{q}}_k, \ddot{\boldsymbol{q}}_k),\ \ddot{\boldsymbol{q}}_{ub}(\boldsymbol{q}_k, \dot{\boldsymbol{q}}_k, \ddot{\boldsymbol{q}}_k)]$$

where the $\boldsymbol{q}_k, \dot{\boldsymbol{q}}_k \in \mathbb{R}^n$ is the robot's current state, $\ddot{\boldsymbol{q}}_k \in \mathbb{R}^n$ is the robot's current acceleration, $J(\boldsymbol{q}_k) \in \mathbb{R}^{m\times n}$ and $\dot{J}(\boldsymbol{q}_k, \dot{\boldsymbol{q}}_k) \in \mathbb{R}^{m\times n}$ 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 $\dddot{\boldsymbol{q}}$, acceleration $\ddot{\boldsymbol{q}}$, velocity $\dot{\boldsymbol{q}}$ and position $\boldsymbol{q}$ in the horizon δt respect their limits.

$$\ddot{q}_{i,ub} = \min\Big\{ \underbrace{\ddot{q}_{i,k} + \dddot{q}_{i,max}\delta t}_{\text{jerk}},\ \underbrace{\ddot{q}_{i,max}}_{\text{acceleration}},\ \dots$$
$$\underbrace{\frac{1}{\delta t}(\dot{q}_{i,max} - \dot{q}_{i,k})}_{\text{velocity}},\ \underbrace{\frac{2}{\delta t^2}(q_{i,max} - q_{i,k} - \dot{q}_{i,k}\delta t)}_{\text{position}} \Big\} \quad (30)$$

where the horizon δt has to be chosen long enough to ensure constraints compatibility without leading to conservative behaviour [28]. Equation (30) 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 (29) consists in two tasks: trajectory tracking task and a secondary (regularisation) task.

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 $\ddot{\boldsymbol{x}}^*$

$$\ddot{\boldsymbol{x}}^* = K_p\boldsymbol{e} + K_d(\dot{\boldsymbol{x}}_d - \dot{\boldsymbol{x}}_k) + \ddot{\boldsymbol{x}}_d$$
$$\boldsymbol{e} = \text{Ad}(X_k)\log(X_k^{-1}X_d) \quad (31)$$

X_d,$\dot{\boldsymbol{x}}_d$,$\ddot{\boldsymbol{x}}_d$ are the desired CS pose, velocity and acceleration in the next step, while X_k and $\dot{\boldsymbol{x}}_k$ 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 $\boldsymbol{e}$ is the CS pose error expressed in the world frame. In the experiments, the PD controller gains used are

$$K_p = \text{diag}\big([170, 170, 170, 100, 100, 100]\big)$$
$$K_d = \text{diag}\big([50, 50, 50, 30, 30, 30]\big)$$

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 $\ddot{\boldsymbol{q}}_r$

$$\ddot{\boldsymbol{q}}_r = k_{rp}(\boldsymbol{q}_r - \boldsymbol{q}) - k_{rd}\dot{\boldsymbol{q}}, \quad \boldsymbol{q}_r = 0.5(\boldsymbol{q}_{max} + \boldsymbol{q}_{min}) \quad (32)$$

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

$$\dot{\boldsymbol{q}}^*_{k+1} = \dot{\boldsymbol{q}}^*_k + \ddot{\boldsymbol{q}}_{opt}\Delta t \quad (33)$$

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

B. Software implementation details

Both simulation and real-world experiments use the open-source library `pinocchio` [39] for rigid body simulation, `ruckig` library [27] for TAP planning, and the Linear Program (LP) solver used is GLPK [40].

The simulation experiments used for the comparative studies are implemented in Python using open-source implementations of TOPP-RA [6] and `robotics-toolbox` library [41] for calculating the inverse kinematics. The code used for the simulation experiments is open-source and piblicly available[4].

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

[4]Gitlab: https://gitlab.inria.fr/auctus-team/people/antunskuric/papers/catp

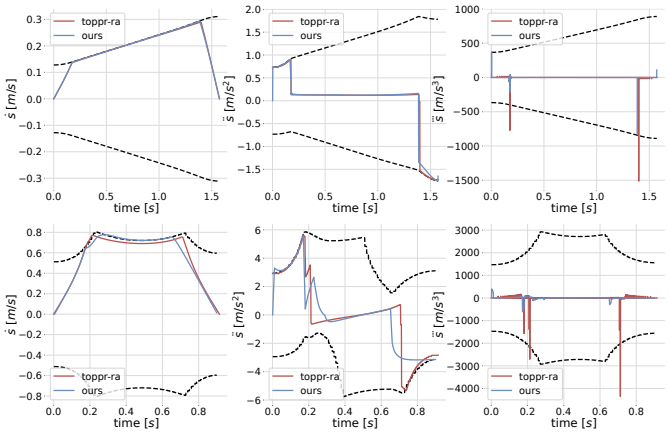

Fig. 10. The plots show the comparison of the path velocity, acceleration and jerk, generated by the proposed approach (blue) and TOPP-RA(red), for two different trajectories (up and down). The dashed lines show the robot's movement limits in the trajectory direction calculated using the proposed method, described in Section III-A.

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 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 asses 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 $\boldsymbol{q}_{w,i}$ for each one of them. To ensure the JS path continuity, the inverse kinematics solution $\boldsymbol{q}_{w,n+1}$ at the way-point $X_{w,n+1}$ is taken as the one closest to the joint configuration $\boldsymbol{q}_{w,n}$ 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 50% of the robot's JS velocity, acceleration and jerk capacity $\alpha_v = \alpha_a = \alpha_j == 0.5$. Both algorithms are ran on a set of 1000 random straight line trajectories (with random fixed orientation), ranging in length from $d = 10cm$ to $1m$.

Results: The comparison of the planned velocity, acceleration and jerk profiles for two trajectories are shown on Figure 10. 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,

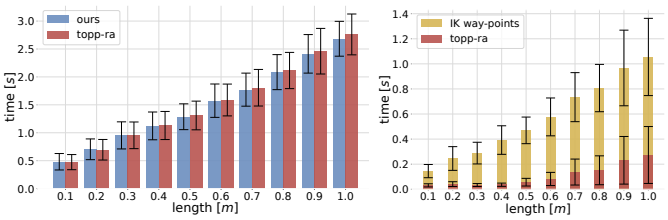

Fig. 11. Plot on the left shows the comparison of the trajectory execution time of the proposed method with TOPP-RA, for trajectory lengths ranging from $d = 0.1$ to $1m$. The right graph shows the computation time of the TOPP-RA algorithm on the same experiment as a comparison to the proposed method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

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 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 10.

Additionally, the influence of the overshoot compensation strategy can be observed in the second (bottom) trajectory. The proposed approach starts braking earlier and does not use the full braking capacity of the robot. 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.

The trajectory execution time comparison is presented in Figure 11 (left). The figure shows that the proposed method has a comparable execution time for all tested trajectory lengths. Interestingly, for lengths over $40cm$, our approach is even faster than TOPP-RA. This can be attributed to the fact that our method takes a different JS path than TOPP-RA, which in some cases might be more optimal than the one calculated in advance. This effect is more present for longer trajectories, where taking the JS path the closest to the initial joint configuration might not be the most optimal criteria.

Finally, 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 algorihtm, ranging from 100ms for short to over 1 second for longer trajectories. The long computation time can be attributed in part the inherent complexity of the algorithm and the requirement to transform the CS path to the set of JS way-points. 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.

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 [35], as given in Table I.

The approaches are compared over 100 straight line trajectories in the robot's workspace (with random fixed orientation),

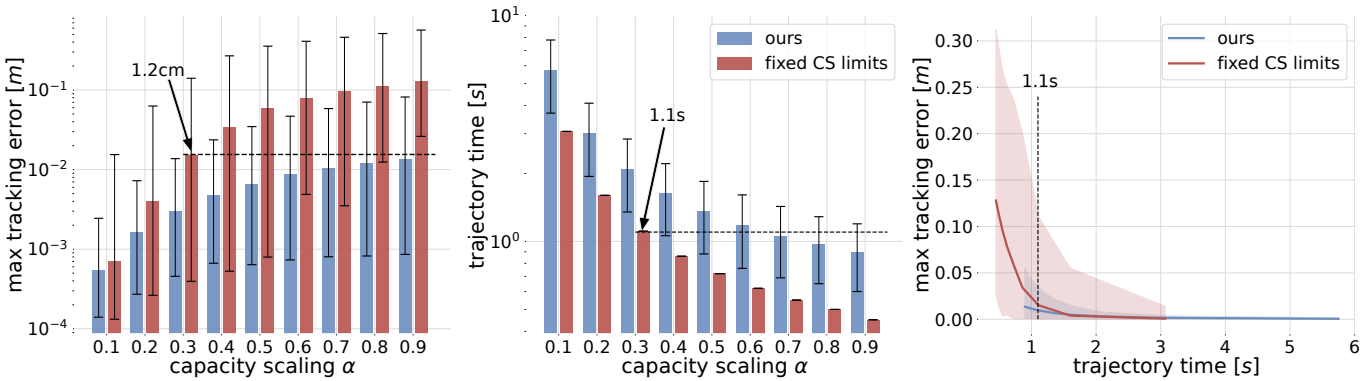

Fig. 12. Result of the benchmarking experiment comparing the fixed CS planning approach (red) to the proposed method (blue). The graph on the left shows comparison of the max tracking error with respect to the ratio of the capacity α used while the graph in the middle shows the trajectory execution time comparison. The graph on the right unites the two graphs on the left in order to show the relation of the trajectory execution time against the max tracking error. Means and variances are calculated on 100 random robot trajectories.

with a fixed length of $d = 50cm$. The performance of the two approaches is compared for different scaling levels α, starting at 10% (α=0.1) of the robot's capacity and going to 90% (α=0.9). The scaling strategy is chosen to be equal for velocity, acceleration and jerk $\alpha = \alpha_v = \alpha_a = \alpha_j$.

Results: Figure 12 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. Figure 12 (on the left) shows that the average tracking error for $\alpha \geq 0.7$ is larger than $10cm$ (while the length of the trajectory is $50cm$).

Figure 12 further shows that using 30% of the manufacturer's CS limits (i.e., α=0.3) results in a tracking error of around 1.2cm, which is higher than any of the mean errors achieved by the proposed method for all tested values of α. While for α=0.3, the fixed CS planning has an execution time of 1.1 seconds, the proposed method has a lower execution time for all $\alpha \geq 0.7$. 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 available[5]. It highlights the effect of increasing the value of α on the overall motion velocity but, 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 [42]. 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 [43] 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 [44], such as the *Zenrobotics Fast picker*[6] or *SELMA*[7] . 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

[5]Video: https://youtu.be/KBo0ZHihi3I

[6]Zenrobotics Fast picker: https://zenrobotics.com/de/

[7]SELMA: https://www.opteknik.se

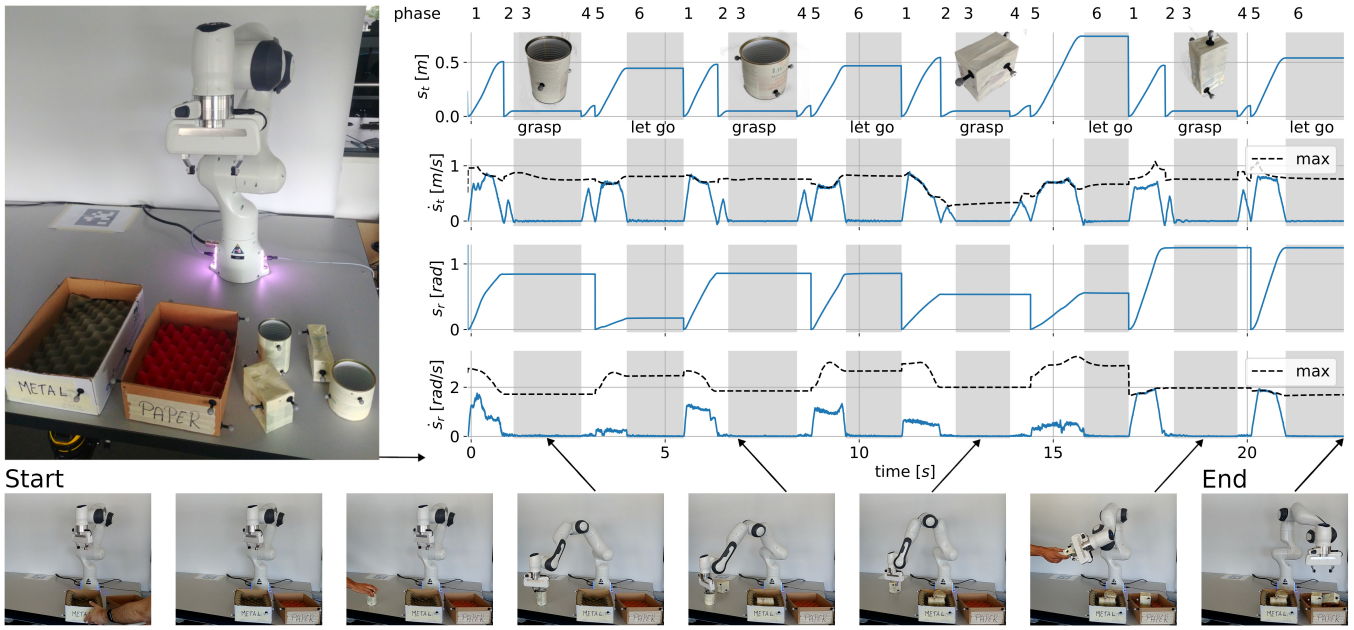

Fig. 13. Figure on the left shows the experimental setup of the mock-up waste sorting experiment. The robot used in the experiments is a Franka Emika Panda robot. The experiment uses six objects: two cans, two cartons and two sorting bins, all tracked in real time using motion capture system Optitrack. 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.

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 (α=0.6) in order to keep the tracking error under 1cm, as shown on Figure 12. The experiment is available in a form of accompanying video[8].

Figure 13 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 then 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 13 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.

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.

[8]Video: https://youtu.be/BHGipnKNOfA

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.

B. Limitations and ways of improvement

One limitation of the proposed approach is its assumption that the CS path can be represented using 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. The consequence of such assumption is that the planned trajectory can guarantee respecting the calculated limits only in the path direction, which is reasonable for straight line trajectories. However, if the path is not a straight line, or if it is a straight line but the target position changes during the trajectory execution, the proposed approach will not be able to guarantee respecting the robot's limits in the directions orthogonal to the path. In order to overcome this drawback, trajectory planning algorithms able to integrate polytope representations of path constraints (5) are required. 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 constrains.

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 [15] or Geometric algebra [45], 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 $\boldsymbol{\tau}$ and different dynamical and gravitational effects acting on the robot, as well of the robot's joint state $\{\boldsymbol{q}, \dot{\boldsymbol{q}}\}$. 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 $\boldsymbol{\tau}$ could make the proposed approach more robust and is a promising direction for future research.

X. Conclusion

This paper aims to address one of the key challenges when it comes to trajectory planning in Cartesian Space (CS): accounting for the robot's changing movement capacity while executing the trajectory. To tackle this challenge, the paper proposes an efficient trajectory planning method, capable of adapting to the robot's changing capacities, by evaluating them in real-time and updating the planned trajectory to account for their changes. This paper showcases the potential of using a real-time evaluation of the robot's movement capacity (physical ability to generate movement: velocity, acceleration, etc.) for creating near time-optimal and reactive trajectory planning strategies in CS.

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]. The results of the comparison show that the proposed method has a comparable trajectory execution time to TOPP-RA even though it is planning in real-time, as opposed to the in-advance optimisation done by TOPP-RA. 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 and while also achieving a lower tracking error. 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. This experiment further demonstrates the proposed method's capability to efficiently utilise the robot's movement capacity, 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] 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.

[10] 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.

[11] K. M. Lynch and F. C. Park, *Modern Robotics: Mechanics, Planning, and Control*. USA: Cambridge University Press, 1st ed., 2017.

[12] A. Palleschi, M. Hamad, S. Abdolshah, M. Garabini, S. Haddadin, and L. Pallottino, "Fast and safe trajectory planning: Solving the cobot performance/safety trade-off in human-robot shared environments," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5445–5452, 2021.

[13] K. Springer, H. Gattringer, and C. Stöger, "A real-time nearly time-optimal point-to-point trajectory planning method using dynamic movement primitives," in *2014 23rd International Conference on Robotics in Alpe-Adria-Danube Region (RAAD)*, pp. 1–6, 2014.

[14] B. Kouvaritakis and M. Cannon, *Model Predictive Control*, ch. Chapter 2: MPC with No Model Uncertainty. Springer International Publishing, 1 ed., 2016.

[15] 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.

[16] 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.

[17] 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.

[18] M. Massaro, S. Lovato, M. Bottin, and G. Rosati, "An optimal control approach to the minimum-time trajectory planning of robotic manipulators," *Robotics*, vol. 12, no. 3, 2023.

[19] S. Zhang, A. M. Zanchettin, R. Villa, and S. Dai, "Real-time trajectory planning based on joint-decoupled optimization in human-robot interaction," *Mechanism and Machine Theory*, vol. 144, p. 103664, 2020.

[20] 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.

[21] 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.

[22] 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.

[23] J. Lee, "A study on the manipulability measures for robot manipulators," in *Proceedings of the IEEE/RSJ 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] 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.

[27] L. Berscheid and T. Kröger, "Jerk-limited real-time trajectory generation with arbitrary target states," *Robotics: Science and Systems XVII*, 2021.

[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] 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/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 11045–11051, 2020.

[33] 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.

[34] Universal Robots, *Universal Robots e-Series User Manual*, 2022.

[35] Franka Emika, "Panda robot datasheet - franka control interface." https://frankaemika.github.io/docs/control_parameters.html, 2022.

[36] 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.

[37] 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.

[38] 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.

[39] J. Carpentier, N. Mansard, F. Valenza, J. Mirabel, G. Saurel, and R. Budhiraja, "Pinocchio - Efficient and versatile Rigid Body Dynamics algorithms," July 2021.

[40] A. O. Makhorin, "Glpk - gnu linear programming kit, version 4.0." Online: http://www.gnu.org/software/glpk/, 2022.

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] 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.