Mechanism and Machine Theory 156 (2021) 104152

Contents lists available at ScienceDirect

Mechanism and Machine Theory

journal homepage: www.elsevier.com/locate/mechmachtheory

Research paper

A real-time trajectory planning method for enhanced path-tracking performance of serial manipulators

Marco Faroni[a,1,*], Manuel Beschi[b,1], Antonio Visioli[b], Nicola Pedrocchi[a]

[a] Istituto di Sistemi e Tecnologie Industriali Intelligenti per il Manifatturiero Avanzato, National Research Council of Italy, Milan, Italy
[b] Dipartimento di Ingegneria Meccanica e Industriale, University of Brescia, Brescia, Italy

A R T I C L E   I N F O

Article history:
Received 22 September 2020
Revised 16 October 2020
Accepted 16 October 2020

Keywords:
Motion planning
Trajectory scaling
Path following
Joint constraints
Robot manipulators

A B S T R A C T

Robotized assembly and manufacturing often require to modify the robot motion at run-time. When the primary constraint is to preserve the geometrical path as much as possible, it is convenient to scale the nominal trajectory in time to meet the robot constraints. Look-ahead techniques are computationally heavy, while non-look-ahead ones usually show poor performance in critical circumstances. This paper proposes a novel technique that can be embedded in non-look-ahead scaling algorithms to improve their performance. The proposed method takes into account the robot velocity, acceleration, and torque limits and modifies the velocity profile based on an approximated look-ahead criterion. To do this, it considers only the last point of a look-ahead window and, by linearizing the problem, it computes the maximum admissible robot velocity. The technique can be applied to existing trajectory scaling algorithms to confer look-ahead properties on them. Simulation and experimental results on a 6-degree-of-freedom manipulator show that the proposed method significantly reduces path-following errors.

1. Introduction

Recent advances in industrial robotics require robots to work in unstructured environments or to cooperate with other agents (*e.g.*, human operators). This has renewed the interest in motion planning algorithms that cope with dynamic situations. Motion planning of industrial manipulators is typically structured as follows:

Step 1: A collision-free path that connects the initial and the final (and possible intermediate) configurations is computed. The path has to be consistent with the joint range limits and the robot reachable workspace. The path can be defined manually in the case of point-by-point programming, but it is more often computed by means of path-planning algorithms. Among them, sampling-based path planners are the most widespread [1,2].

Step 2: A time parametrization is assigned to the path. The result of this step is a velocity profile along the path. Such profile should respect the robot limits (*e.g.*, joint maximum velocity, acceleration, and/or torque) and process requirements (*e.g.*, constant-speed traits in gluing and laser-cutting operations).

[*] Corresponding author.
E-mail addresses: marco.faroni@stiima.cnr.it (M. Faroni), manuel.beschi@unibs.it (M. Beschi), antonio.visioli@unibs.it (A. Visioli), nicola.pedrocchi@stiima.cnr.it (N. Pedrocchi).
[1] These authors contributed equally to the work.

https://doi.org/10.1016/j.mechmachtheory.2020.104152
0094-114X/© 2020 Elsevier Ltd. All rights reserved.

Step 3: An open-loop controller modifies the nominal trajectory based on real-time information from the process during the execution of the motion. This step may include real-time trajectory re-planning [3,4], safety speed modulation for human-robot collaboration [5,6] or speed coordination with conveyors and other robots.

The resulting motion is then tracked by a closed-loop controller. Note that this paradigm is not strict. For example, Steps 1 and 2 may merge if the path planner also takes into account the robot velocity and acceleration limits (these planners are usually referred to as kinodynamic [1]). Also, Step 3 may not be present in simple applications.

Steps 1 and 2 are computed before the execution of the motion. In static applications (*i.e.*, without cooperation with other agents and in structured environments), they may be performed offline by using, for example, optimization-based algorithms [7–9] or by shaping high-order polynomials [10–12]. In the presence of dynamic obstacles and goals, they need to be performed just before the robot movement and computational time is therefore a key issue. Step 3 runs during the execution of the motion; computing time requirements are therefore strict as the algorithm has a sampling period comparable with the robot controller's one.

As mentioned, the output of Step 2 is a velocity profile along the path that respects the robot limits. However, finding an optimal solution to the path-parametrization problem in Step 2 is time consuming and not compatible with many applications that do not allow for offline trajectory planning. For example, in many manipulation scenarios, grasping poses of the objects are not fixed, so they are calculated based on a vision system. Feasible trajectories are therefore planned on the fly, just before their execution, and planning time is a critical point. Similar issues arise in human-robot collaboration and multi-agent systems, where online re-planning is often required. A common approach is to use approximated methods in Steps 1 and 2 and to refine the motion online in Step 3. In this way, part of the computational burden can be moved to Step 3, reducing latency of the trajectory execution. In other words, Steps 1 and 2 compute an approximated trajectory that does not take into account kinodynamic constraints. The algorithms in Step 3 are then in charge of modifying the trajectory iteratively at runtime, to make it compliant with the robot limits at the current sampling period.

Another issue addressed in Step 3 is the fast reaction to the behavior of other agents (*e.g.*, human co-workers) and changes in the environment. Real-time path re-planning may be adopted, but the resulting planning problem is usually too complex to be solved with small latency. In this regard, anytime sampling-based motion planners allow for collision-free online path re-planning, but they are far from being implementable with sampling periods in the order of few milliseconds [2,13]. Another approach consists in the modification of the velocity profile along the same path. Preserving the nominal path leads to two advantages: i) the technological application is not jeopardized[2]; and ii) there is no need for re-computing a collision-free path, which was the computational bottleneck of the aforementioned approach. For these reasons, online speed reduction is also the preferred way to ensure safety in human-robot collaboration. For example, Speed and Separation Monitoring and Power Force Limitation modes, as defined in the safety standard ISO/TS 15066 [14], are implemented by slowing down the robot speed according to safety constraints from the standard. This has usually been implemented by means of danger zones where the robot speed is limited [15], but recent more advanced strategies tend to modulate the speed dynamically depending on the human-robot distance and the safety requirements defined by the standard (namely, the minimum protective distance and the maximum impact force) [5]. These advanced approaches might take advantage of algorithms capable of taking into account robot physical limitations during the deceleration phase.

1.1. Related works

In many cases, it is desirable to preserve the nominal geometrical path defined by the technological process the robot is performing. Real-time algorithms can modify the nominal velocity profile to preserve the path when the trajectory in execution is expected to exceed the robot physical limitation or to avoid collisions with humans and other robots. The methods devoted to this purpose are referred to as *online trajectory scaling methods*.

Different approaches have been proposed in the literature. Most of them approximate the planning problem by only taking into account the desired motion at the current time instant to reduce the computational effort as much as possible. For this reason, they are referred to as *local* or *non-look-ahead*. Examples can be found in [16], in which a scaling factor that multiplies the nominal velocity is calculated at each iteration of the task. Such scaling factor can slow down the trajectory execution in case it does not satisfy the joint velocity or acceleration limits. In [17], this principle is extended to redundant manipulators under joint kinematic limits. Another strategy is proposed in [18–20], where the scaling architecture is based on a nonlinear filter that modifies the longitudinal velocity profile based on the activation of the joint constraints. In case of infeasibility, path deformations arise and the filter is in charge of converging to the original task in minimum time. The main disadvantage of the *non-look-ahead* methods is that they only take into account information about the current state of the system. As a consequence, it is often impossible to prevent saturation of the robot joint controller. This results in path-following errors. Heuristic strategies were devised in [21] to partially address this problem in the methods described in [18]. Such rules need, however, to be tuned empirically and their effectiveness strongly depends on the trajectory to be performed.

[2] Consider, for example, a gluing or a cutting operation along a given path, where the dispensing of glue or the cutting speed/power automatically adapts to the speed of the robot; in this case it is possible to change the timing law to optimize the execution of the task.

Look-ahead strategies have been also proposed [22–24] to overcome this issue. They typically set up a constrained non-linear optimal control problem to minimize the path deviation along a predictive time horizon. The path-parametrization variable is used as an additional control variable to determine the optimized timing law. They give better results if compared with non-look-ahead techniques, but they require much higher computational effort to solve the optimal control problem online. In fact, their implementation can be troublesome on industrial controllers because of the limited computing resources and mathematical software tools.

Online trajectory scaling has also been adopted for safe human-robot collaboration. In [25], linear safety constraints in the robot velocity are designed to avoid human-robot collisions; in [26], a safety function is related to the scaling of the trajectory; in [27], a receding-horizon approach scales the trajectory based on the human-robot relative distance. In these works, trajectory scaling is a tool used to slow down the trajectory to a safe speed according to the human proximity. The trajectory scaling algorithms used in these works trace back to those described in the aforementioned approaches. In particular, [25,26] use non-look-ahead methods;[27] uses the look-ahead MPC-based approach described in [23].

1.2. Contribution

The paper proposes a novel technique that can be embedded in non-look-ahead scaling techniques to improve their performance. The proposed method takes into account the robot velocity, acceleration, and torque limits and modifies the velocity profile (output of Step 2) based on an approximated look-ahead criterion. As a result, the maximum admissible robot speed is given as new reference to a non-look-ahead trajectory scaling method. This confers predictive features on the existing scaling algorithm. The advantages of the proposed approach compared to existing techniques are that:

- The proposed technique can be integrated with existing scaling algorithms as it only requires to modify the nominal velocity profile before using it in any scaling methods.
- The additional computation burdens with respect to non-look-ahead algorithm are negligible (as only few simple calculations are needed) and, thus, way lower than existing optimization-based look-ahead methods [22–24].
- The performance in terms of path-following error and execution time are significantly better than non-look-ahead approaches (*e.g.*, [16–20]), as proven by simulation and experimental results.
- The method is not optimization-based, which is an advantage with respect to existing look-ahead methods [22–24] in terms of ease of implementation, especially on industrial controllers, often characterized by limited computational capabilities.
- Velocity, acceleration, and torque limits are taken into account at the same time (note that [22] does not include velocity and acceleration limits and [23] does not include torque limits).

Simulation and experimental tests are presented to prove the effectiveness of the method. Complex operational-space tasks are performed by a 6-degree-of-freedom Universal Robot UR10 robot. A comparison with look-ahead and non-look-ahead methods is also provided. A video also accompanies the paper to show the experimental tests and clarify the practical consequences.

The paper is organized as follows. Section 2 introduces preliminary notions on online trajectory scaling. In Section 3, the method to compute the modified velocity profile is described. Both joint- and operational-space trajectories are addressed. Simulation and experimental results on a 6-degree-of-freedom manipulator performing a complex task are analyzed in Sections 4 and 5 respectively. Finally, conclusions are drawn in Section 6.

2. Preliminaries

Consider an n-degree-of-freedom robot manipulator and denote with q, $\dot{q}$, $\ddot{q}$, and $\tau \in \mathbb{R}^n$ its joint configuration, velocity, acceleration, and torque vectors respectively. Physical limitations of the robot typically regard the joint velocities, accelerations and torques, which can be expressed as:

$$-\dot{q}_{\max} \leq \dot{q} \leq \dot{q}_{\max} \tag{1}$$

$$-\ddot{q}_{\max} \leq \ddot{q} \leq \ddot{q}_{\max} \tag{2}$$

$$-\tau_{\max} \leq \tau \leq \tau_{\max} \tag{3}$$

where $\dot{q}_{\max}$, $\ddot{q}_{\max}$, and $\tau_{\max} \in \mathbb{R}^n$ are the maximum joint velocity, acceleration, and torque vectors respectively. From a practical perspective, torque and velocity limits are typically physical constraints due to the limited motor effort and are usually given in the datasheet of the robot. Acceleration limits are usually software limits imposed to avoid abrupt motion and are typically set depending on the application/robot. Configuration limits are not considered in this work, as the path is assumed to be within the robot workspace, that is, the path generated by the path planner (Step 1 in Section 1) is reasonably contained in the configuration bounds (with a safety margin accounting for possible tracking errors).

The robot is required to perform a joint-space curve described by a vector function $\Gamma(\gamma)$ defined as follows:

$$\begin{aligned}\Gamma &: [0, \gamma_f] \to \mathbb{R}^n \\ \gamma &\mapsto q_d := \Gamma(\gamma)\end{aligned} \tag{4}$$

where $\gamma \in \mathbb{R}$ is the scalar variable chosen to parametrize the curve and γ_f is its final value. For example, if γ is the arc-length along the path, then γ_f is equal to the total length of the path. Now define the timing law $\gamma(t)$ as:

$$\begin{aligned}\gamma &: [0, t_f] \to [0, \gamma_f] \\ t &\mapsto \gamma(t)\end{aligned} \tag{5}$$

where t_f is the total traveling time and γ is non-decreasing in time to avoid inversions of the sense of motion.

Referring to Section 1, Γ is computed by a path planner at Step 1, and $\gamma(t)$ should be computed at Step 2. Online scaling algorithms take the nominal trajectory computed at Step 2 as input and modify the timing law iteratively in Step 3. The scaling algorithm has to solve the following Optimal Control Problem (OCP) at each time instant t_0:

$$\begin{aligned}&\text{minimize}_\gamma \int_{t_0}^{t_f} \left(\gamma(t) - \gamma^0(t)\right)^2 dt \\ &\text{subject to } -\dot{q}_{\max} \le \dot{q}(t) \le \dot{q}_{\max} \\ &\qquad -\ddot{q}_{\max} \le \ddot{q}(t) \le \ddot{q}_{\max} \\ &\qquad -\tau_{\max} \le \tau(t) \le \tau_{\max} \\ &\qquad \dot{\gamma}(t) \ge 0 \qquad\qquad \forall t \in [t_0, t_f]\end{aligned} \tag{6}$$

where γ^0 is the nominal timing law (*i.e.*, the output of Step 2) and inequalities between vectors are intended as component-wise. OCP (6) aims to compute a timing law as close as possible to the nominal one γ^0, but that is also compliant with the robot physical limitations. The requirement $\dot{\gamma}(t) \ge 0$ is necessary to discard solutions with motion direction opposite to the desired one.

Limited computing resources do not permit to solve (6) within a reasonable sampling period (*i.e.*, around 1–10 ms). Considering a shorter time horizon $[t_0, t_0 + h]$, $0 < h < t_f - t_0$ gives, however, satisfactory results, as proven in previous works [22,24]. We will refer to such approach as look-ahead. In fact, most existing methods only consider the system at time t_0 in (6), to reduce the computing time as much as possible. We will refer to this last approach as non-look-ahead, as it is equivalent to setting $h = 0$. The resulting simplified OCP is:

$$\begin{aligned}&\text{minimize}_r \left(r - \gamma^0(t_0)\right)^2 \\ &\text{subject to } -\dot{q}_{\max} \le \dot{q}(t_0) \le \dot{q}_{\max} \\ &\qquad -\ddot{q}_{\max} \le \ddot{q}(t_0) \le \ddot{q}_{\max} \\ &\qquad -\tau_{\max} \le \tau(t_0) \le \tau_{\max} \\ &\qquad \dot{r} \ge 0\end{aligned} \tag{7}$$

where r is the decision variable, corresponding to the timing law at the current time instant.

OCP (7) is a general formulation that applies to most of the existing non-look-ahead methods in the literature (with some minor modifications that, nonetheless, may significantly affect the performance). The main differences between existing methods are in the algorithm to solve the problem and in the constraints taken into consideration, as pointed out in Section 1. For example, [18,19] find the optimal value of r in closed form by re-writing the robot constraints in the scalar variable r; [17] approximates (7) with a quadratic program in the joint velocity vector and in the timing law scaling factor (this approach is also used in Sections 4 and 5 to assess the performance of our proposed approach).

Please note that, according to the tiered planning framework described in Section 1, OCPs (6) and (7) are implemented similarly to a discrete-time controller with sampling period T. That is, the OCP is solved at each sampling time and the solution is sent as reference timing law to the robot controller; then the procedure is repeated at the next sampling time.

3. Method

3.1. Working principle

Existing scaling methods aim to compute a timing law that is as close as possible to the reference γ^0. When the non-look-ahead approach (7) is adopted, there is a loss of information about the future trajectory that may lead to poor control choices. The result is a large path-following error because of saturation of the joint variables. In this paper, we introduce a Timing-law Adaptation Module (TAM) that modifies the reference γ^0 if the future trajectory is expected to be infeasible with respect to the physical limits of the robot. A block diagram of the proposed algorithm is represented in Fig. 1, where the joint-space case is illustrated. The bottom part of the figure represents a typical implementation of trajectory scaling algorithms. The nominal trajectory block computes the desired motion at each time instant. In practice, it can be

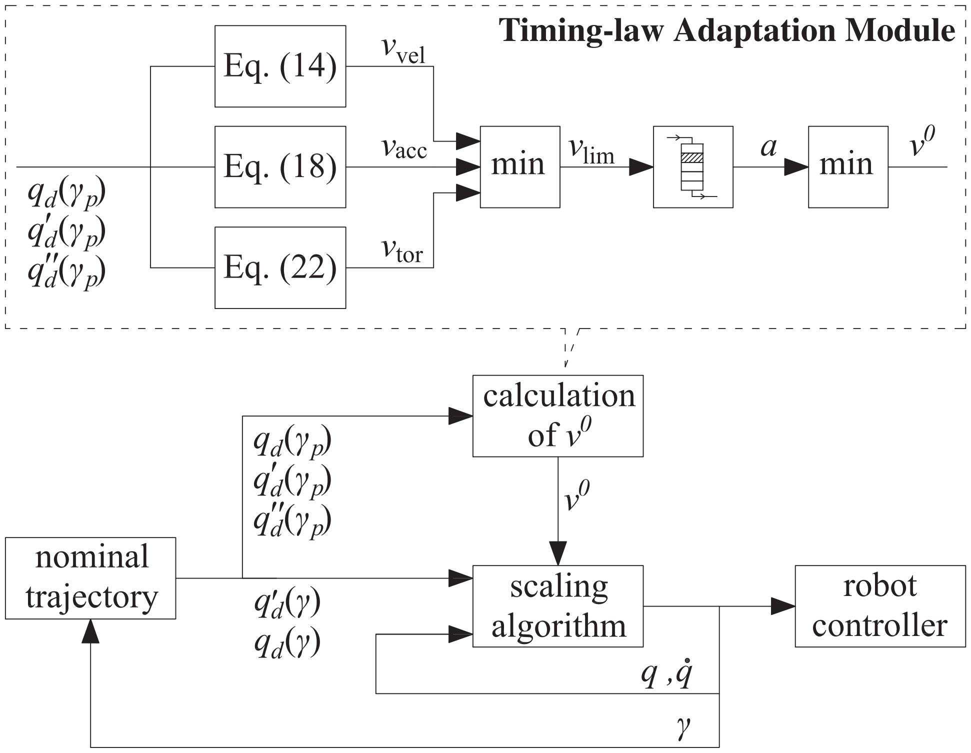


Fig. 1. Block scheme of the proposed scaling algorithm.

implemented as an explicit function or as an interpolator between waypoints; the output is however the desired motion at a certain value of the parameter γ. The desired motion is, in turn, received as input by the scaling algorithm, which can modify it, if it is expected to violate joint limits or user-defined constraints (*e.g.*, safety constraints in human-robot collaboration). The output of the scaling algorithm is the actual reference motion sent to the robot controller at each time instant. The scaling algorithm also updates the parameter γ that will be used to compute the new nominal motion at the next cycle. The Timing-law Adaptation Module (top part of Fig. 1) is added to this framework to compute a modified scaling reference ν^0 instead of $\gamma^0(t_0)$. To do so, joint configuration, velocities, accelerations, and torques expected at the end of a predictive window are considered to compute the maximum allowed scaling reference that would satisfy the robot physical limits (ν_{vel}, ν_{acc}, ν_{tor}). If the nominal scaling $\gamma^0(t_0)$ is compliant with them, then it is not modified; otherwise the minimum value ν_{lim} is chosen as new candidate reference. This candidate reference is stored in a circular buffer of length equal to the predictive window. This circular buffer is necessary to keep track of the maximum allowed values along the whole window (and not only at the end of it). As a matter of fact, considering only the end point of the window would discard possible violation of constraints that were analyzed at previous iterations. The minimum value in the buffer is therefore chosen as the actual scaling reference ν^0 and sent to the existing scaling algorithm. This procedure is repeated at each sampling time, with the look-ahead window (and the buffer) moving forward along the trajectory.

3.2. Timing-law adaptation module

Consider an n-degree-of-freedom manipulator performing the desired joint trajectory (4) and (5). The timing law $\gamma(t)$ can be adjusted online by acting on a decision variable ν s.t. $\dot{\gamma} = \nu$. Our goal is to choose the proper value of ν at each time instant, in such a way that the resulting motion along a look-ahead window of length h is compliant with the robot limits. To this purpose, consider a future time instant $t_p = t_0 + h$, $h \in \mathbb{R}^+$, where t_0 is the current time instant. At t_p corresponds $\gamma_p = \gamma(t_0) + \int_{t_0}^{t_p} \nu \, dt$. The motion at time t_p can be expressed as:

$$q(t_p) = q_d(\gamma_p) \tag{8}$$

$$\dot{q}(t_p) = q'_d(\gamma_p)\,\nu(t_p) \tag{9}$$

$$\ddot{q}(t_p) = q''_d(\gamma_p)\,\nu^2(t_p) + q'_d(\gamma_p)\,\dot{\nu}(t_p) \tag{10}$$

where $q'_d = dq_d/d\gamma$, $q''_d = dq'_d/d\gamma$ and $\dot{v}_p = \frac{dv}{dt}$. Variable γ_p actually depends on the future trend of v, which is unknown, but it is reasonable to use the first-order approximation $\gamma_p \approx \gamma(t_0) + hv(t_0)$. The proper value of v should be chosen in such a way that the resulting motion $q(t_p)$, $\dot{q}(t_p)$, $\ddot{q}(t_p)$ respects the robot constraints. We therefore analyze velocity, acceleration, and torque limits at time t_p to find the maximum allowed value to satisfy them.

Velocity limits on the n joints of the robot at time t_p can be imposed as:

$$-\dot{q}_{\max,i} \le \dot{q}_i(t_p) \le \dot{q}_{\max,i} \quad \forall i = 1,\dots,n, \tag{11}$$

which, using (9), becomes:

$$-\dot{q}_{\max,i} \le q'_{d,i}(\gamma_p)\, v \le \dot{q}_{\max,i} \quad \forall i = 1,\dots,n, \tag{12}$$

from which:

$$v \le \frac{\dot{q}_{\max,i}}{|q'_{d,i}(\gamma_p)|} \quad \forall i = 1,\dots,n. \tag{13}$$

Thus, the maximum value of v which permits to satisfy the velocity constraints of all joints is defined as:

$$v_{\text{vel}} := \min_{i=1,\dots,n} \frac{\dot{q}_{\max,i}}{|q'_{d,i}(\gamma_p)|} \tag{14}$$

(in case $q'_{d,i}(\gamma_p) = 0$, we impose $\dot{q}_{\max,i}/|q'_{d,i}(\gamma_p)| = +\infty$).

Similarly, joint acceleration limits at time t_p can be expressed as:

$$-\ddot{q}_{\max,i} \le \ddot{q}_i(t_p) \le \ddot{q}_{\max,i} \quad \forall i = 1,\dots,n. \tag{15}$$

By using (10), it results:

$$-\ddot{q}_{\max,i} \le q''_{d,i}(\gamma_p)\, v^2 + q'_{d,i}(\gamma_p)\, \dot{v} \le \ddot{q}_{\max,i} \quad \forall i = 1,\dots,n. \tag{16}$$

The size of the predictive window h is a tunable parameter of the method. Its value should be long enough for the system to reach the modified reference in a smooth manner. Thus, we assume that the term dependent on $\dot{v}$ is negligible compared with the one dependent on v^2 (this assumption will be verified empirically in Section 4.3.3). The acceleration bounds become:

$$-\ddot{q}_{\max,i} \le q''_{d,i}(\gamma_p)\, \dot{\gamma}_p^2(t_p) \le \ddot{q}_{\max,i} \quad \forall i = 1,\dots,n. \tag{17}$$

In this way, the maximum value of $\dot{\gamma}_p$ that respects the acceleration bound is defined as:

$$v_{\text{acc}} := \min_{i=1,\dots,n} \sqrt{\frac{\ddot{q}_{\max,i}}{|q''_{d,i}(\gamma_p)|}} \tag{18}$$

(if $q''_{d,i}(\gamma_p) = 0$, we impose $\ddot{q}_{\max,i}/|q''_{d,i}(\gamma_p)| = +\infty$).

Consider now the torque limits at time t_p:

$$-\tau_{\max,i} \le \tau_i(t_p) \le \tau_{\max,i} \quad \forall i = 1,\dots,n. \tag{19}$$

Torque limits are rewritten in the joint kinematic variables through the robot inverse dynamics function. We consider here the robot dynamic model in the well-known form:

$$\tau = H(q)\,\ddot{q} + C(q,\dot{q})\,\dot{q} + g(q) + f_v\,\dot{q} + f_s\,\text{sign}(\dot{q}) \tag{20}$$

where $H(q) \in \mathbb{R}^{n\times n}$ is the symmetric positive-definite inertia matrix, $C(q,\dot{q}) \in \mathbb{R}^{n\times n}$ accounts for centrifugal and Coriolis effects, $g(q) \in R^n$ represents gravity torques, and f_v, $f_s \in \mathbb{R}^n$ model viscous and Coulomb friction. Torque τ at time t_p can be re-written by using (8)–(10) in (20) as follows:

$$\begin{aligned}\tau(t_p) = {} & H\big(q_d(\gamma_p)\big)\left(q''_d(\gamma_p)\,v^2 + q'_d(\gamma_p)\dot{v}\right) + C\big(q_d(\gamma_p), q'_d(\gamma_p)\big)q'_d(\gamma_p)v^2 \\ & + g\big(q_d(\gamma_p)\big) + f_v\,q'_d(\gamma_p)v + f_s\,\text{sign}\big(q'_d(\gamma_p)\big).\end{aligned} \tag{21}$$

By using the same approximation adopted from (16) to (17), it is possible to drop the term $q'_d(\gamma_p)\dot{v}$ in (21). Then, the maximum value of v that satisfies (19) can be found by solving the resulting second-order equation, that is:

$$v_{\text{tor}} := \min_{i=1,\dots,n} \left\{ v \ge 0 \;\middle|\; \big(H_i\,q''_{d,i}(\gamma_p) + C_i\,q'_{d,i}(\gamma_p)\big)v^2 + \dots\dots + f_{v,i}\,q'_{d,i}(\gamma_p)\,v + g_i + f_{s,i}\,\text{sign}\big(q'_{d,i}(\gamma_p)\big) - \tau_{\text{lim},i} = 0 \right\}, \tag{22}$$

where H_i, C_i, g_i, $f_{v,i}$, and $f_{s,i}$ denote the ith row of the respective matrices/vectors (the dependencies on the joint variables have been omitted for the sake of clarity) and:

$$\tau_{\text{lim},i} = \begin{cases} -\tau_{\max,i} & \text{if } \tau_{d,i}(\gamma_p) < 0, \\ \tau_{\max,i} & \text{if } \tau_{d,i}(\gamma_p) > 0. \end{cases} \tag{23}$$

where $\tau_{d,i}(\gamma_p)$ is the nominal desired torque computed by using $v = \dot{\gamma}^0$ in (21).

Once the limit values ν_{vel}, ν_{acc}, and ν_{tor} have been computed, the overall reference value is given by:

$$\nu_{\text{lim}} = \min\left(\nu_{\text{vel}}, \nu_{\text{acc}}, \nu_{\text{tor}}, \dot{\gamma}^0(t_p)\right). \tag{24}$$

At this point, ν_{lim} is inserted in circular buffer a of length $l = \lceil h/T \rceil$ (rounded using ceiling function), where T is the sampling period at which TAM runs so that:

```
**for** $j = 1$ to $l - 1$ **do**
    $a(j) \leftarrow a(j+1)$
**end for**
$a(l) \leftarrow \nu_{\text{lim}}$
```

Note that the buffer is necessary to keep track of the values of ν_{lim} calculated at the previous l iterations. Without it, possible constraints violations along the predictive window would be discarded, as only the end point of the window would be taken into account. The minimum value of the buffer is the maximum admissible value of ν along the whole predictive window. By denoting the circular buffer with $a \in \mathbb{R}^l$, the reference value ν^0 at time t_0 is:

$$\nu^0(t_0) = \min_{i=1,\ldots,l} a(i). \tag{25}$$

TAM's output ν^0 determines the velocity profile of the timing law to be given as reference to (7). The reference timing law γ^0 in (7) is therefore computed as $\gamma^0(t_0) = \int_0^{t_0} \nu^0(t)dt$.

3.3. Operational-space trajectories

Section 3.2 naturally applies to joint-space trajectories, but it can be extended to operational-space tasks as well. To this purpose, consider a Cartesian curve $\Gamma_{cs}(\gamma)$ and denote with $f : \mathbb{R}^n \to \mathbb{R}^m$, $q \mapsto f(q)$ (with $1 \le m \le n$) the robot forward kinematics function. If f is invertible (*i.e.*, there are no kinematic singularities on the path and the robot is not redundant), the problem traces back to that described in Section 3.2 by defining an equivalent joint-space desired motion as follows:

$$q_d(\gamma_p) := f^{-1}\left(\Gamma_{cs}(\gamma_p)\right), \tag{26}$$

$$q_d'(\gamma_p) := J^{-1}\,\Gamma_{cs}'(\gamma_p), \tag{27}$$

$$q_d''(\gamma_p) := J^{-1}\left(\Gamma_{cs}''(\gamma_p) - \left[\frac{\partial J}{\partial q} q_d'(\gamma_p)\right] q_d'(\gamma_p)\right). \tag{28}$$

where $f^{-1} : \mathbb{R}^m \to \mathbb{R}^n$ denotes the inverse kinematics function of the manipulator and $J = \partial f / \partial q$ is its Jacobian. Then, the methodology derived from (8) on can be applied just alike.

4. Simulations

4.1. Setup

Simulations have been performed by using a 6-degree-of-freedom Universal Robots UR10 manipulator. The limits of the robot have been set to:

$$\dot{q}_{\text{max}} = (\,2,\ 2,\ 3,\ 3,\ 3,\ 3\,)\ \text{rad/s}, \tag{29}$$

$$\ddot{q}_{\text{max}} = (\,5,\ 5,\ 10,\ 10,\ 10,\ 10\,)\ \text{rad/s}^2, \tag{30}$$

$$\tau_{\text{max}} = (\,200,\ 200,\ 100,\ 50,\ 50,\ 50\,)\ \text{Nm}. \tag{31}$$

Simulations use a virtual model of the robot, that is, the planned motion is tracked with no errors. This allows the evaluation of the performance of the planning methods, as the tracking error introduced by the controller is not present. Path following errors are therefore due to constrained solutions of the OCP. The sampling period of the online planning algorithm is 1 ms. The dynamics parameters needed to include the torque limits (23) are publicly available on the website of the robot manufacturer and are discussed in details in [28].

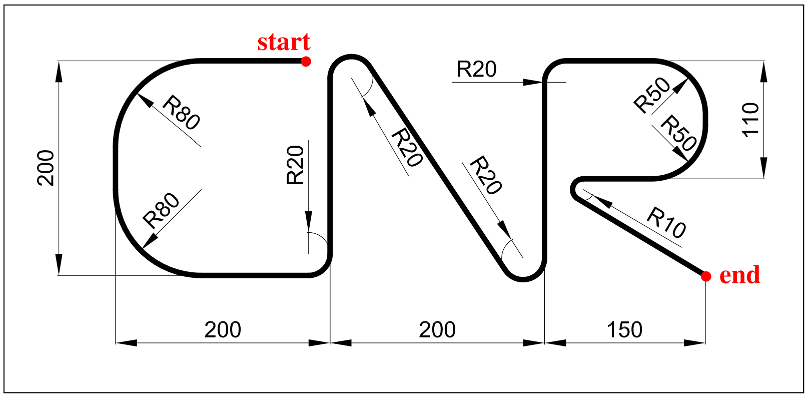


Fig. 2. Drawing of the simulation task (quotes are in millimeters).

4.2. Compared methods, tests, and settings

TAM can be applied to generic online trajectory scaling methods to improve their performance. In these tests, we use a Non-Look-Ahead method (NLA) based on [17], with acceleration and torque limits added as described in [29]. This NLA method approximates OCP (7) with a quadratic program, where the optimization variables are the joint acceleration vector u and the timing law speed v, such that:

$$\begin{aligned} &\text{minimize}_{u,v} \left\| J\,\dot{q} - x^0(t_0)\,v \right\|^2 + \lambda\big(\dot{\gamma}^0(t_0) - v\big)^2 \\ &\text{subject to} \quad -\dot{q}_{\max} \le \dot{q} \le \dot{q}_{\max} \\ &\qquad -\ddot{q}_{\max} \le \ddot{q} \le \ddot{q}_{\max} \\ &\qquad -\tau_{\max} \le \tau \le \tau_{\max} \\ &\qquad \tau = H\ddot{q} + C\dot{q} + g + f_v\,\dot{q} + f_s\,\mathrm{sign}(\dot{q}) \\ &\qquad \ddot{q} = u \end{aligned} \tag{32}$$

The cost function of (32) is an approximation of that of (7): the first term aims to minimize the difference between the computed Cartesian twist and the direction of the twist $x^0(t_0)$. In other words, it converts the path-following requirement into a soft-constraint in order to ensure the feasibility of the quadratic program. Note that the problem is formulated at velocity level. Because of this, if position errors occur (*e.g.*, due to numerical drift), OCP (32) is not able to recover them. To cope with this, the nominal twist is modified based on a proportional control law such that $x^0 = \Gamma'_{cs}(\gamma) + Ke$, where $e = \Gamma_{cs} - f(q)$ is the position Cartesian error, K is a proportional gain and f is the forward kinematics function of the robot (see [30] for details). The second term of the cost function minimizes the difference between the computed timing law v and the nominal one. OCP (32) also includes joint velocity, acceleration and torque bounds, just like (7). Torque bounds are considered through the inverse dynamics of the robot, H, C, g, f_v, f_s are the dynamics matrices as defined in (20) evaluated at $q(t_0)$ and $\dot{q}(t_0)$. Similarly, J is the Jacobian of the manipulator at $q(t_0)$. TAM is applied above NLA, in the sense that it modifies the nominal velocity profile to improve the feasibility chances. That is, when NLA is used, $\dot{\gamma}^0$ is the nominal velocity profile; when TAM is used, $\dot{\gamma}^0$ is the modified velocity reference obtained from (25). In all tests, we set $K = 100$ and $\lambda = 1 \cdot 10^{-3}$ in (32), and the look ahead window $h = 0.2$ s.

The look-ahead method [23] is also considered for comparison. This look ahead method solves OCP (6) with a finite horizon and approximates the resulting nonlinear program with a quadratic program in order to reduce computing time, allowing for online implementation with high sampling rates. We implement the look-ahead method with a predictive horizon of 0.2 s, in order to have a fair comparison with the tuning of TAM. Moreover, the predictive horizon is discretized into a staircase function of 10 steps during the solution of the OCP. Details on setting and implementation issues of the method are discussed in [23,24]. For this technique is based on Model Predictive Control, it will be referred to as MPC.

Position and orientation errors are evaluated. Position error is the Euclidean distance between the end-effector position and the closest point on the desired path. Orientation error is the absolute value of the angle between the end-effector reference frame and the desired reference frame. Another relevant parameter is the average slowdown introduced by the scaling method, computed as the ratio between the desired execution time and the time taken by the robot to perform the whole task (t_{real}/t_f).

The task is an example of manufacturing task, such as gluing or laser cutting, along complex shapes. The nominal path consists in the stylized profile of the word *CNR* in the *yz*-plain. The shape is a sequence of segments and circular arcs; quotes are shown in Fig. 2. During the task, the end-effector orientation should be orthogonal to the *yz*-plain. The nominal

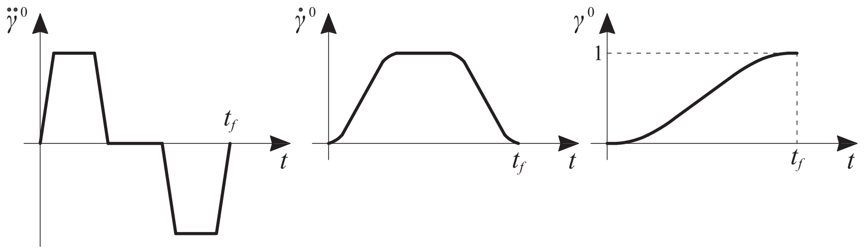


Fig. 3. Seven-trait-acceleration timing law along.

Table 1
Translation error for different nominal trajectory times t_f.

t_f	$e_{\max}$[m]			e_{mean}[m]		
[s]	TAM	NLA	MPC	TAM	NLA	MPC
5.0	$1.33{\cdot}10^{-4}$	$5.30{\cdot}10^{-3}$	$1.55{\cdot}10^{-4}$	$2.55{\cdot}10^{-5}$	$2.98{\cdot}10^{-4}$	$2.67{\cdot}10^{-5}$
4.0	$1.71{\cdot}10^{-4}$	$1.78{\cdot}10^{-2}$	$1.98{\cdot}10^{-4}$	$3.58{\cdot}10^{-5}$	$1.95{\cdot}10^{-3}$	$3.93{\cdot}10^{-5}$
3.0	$2.29{\cdot}10^{-4}$	$4.65{\cdot}10^{-2}$	$2.01{\cdot}10^{-4}$	$5.04{\cdot}10^{-5}$	$7.44{\cdot}10^{-3}$	$4.99{\cdot}10^{-5}$
2.0	$3.85{\cdot}10^{-4}$	$1.43{\cdot}10^{-1}$	$2.99{\cdot}10^{-4}$	$8.43{\cdot}10^{-5}$	$4.05{\cdot}10^{-2}$	$1.03{\cdot}10^{-4}$

Table 2
Orientation error for different nominal trajectory times t_f.

t_f	$e_{\max}$[rad]			e_{mean}[rad]		
[s]	TAM	NLA	MPC	TAM	NLA	MPC
5.0	$6.44{\cdot}10^{-5}$	$7.62{\cdot}10^{-3}$	$8.06{\cdot}10^{-5}$	$6.79{\cdot}10^{-7}$	$4.35{\cdot}10^{-4}$	$1.02{\cdot}10^{-6}$
4.0	$7.82{\cdot}10^{-5}$	$2.51{\cdot}10^{-2}$	$8.98{\cdot}10^{-5}$	$1.04{\cdot}10^{-6}$	$3.05{\cdot}10^{-3}$	$1.33{\cdot}10^{-6}$
3.0	$8.35{\cdot}10^{-5}$	$6.37{\cdot}10^{-2}$	$9.77{\cdot}10^{-5}$	$1.85{\cdot}10^{-6}$	$1.11{\cdot}10^{-2}$	$1.55{\cdot}10^{-6}$
2.0	$8.63{\cdot}10^{-5}$	$4.19{\cdot}10^{-1}$	$9.91{\cdot}10^{-5}$	$3.11{\cdot}10^{-6}$	$1.11{\cdot}10^{-1}$	$2.01{\cdot}10^{-6}$

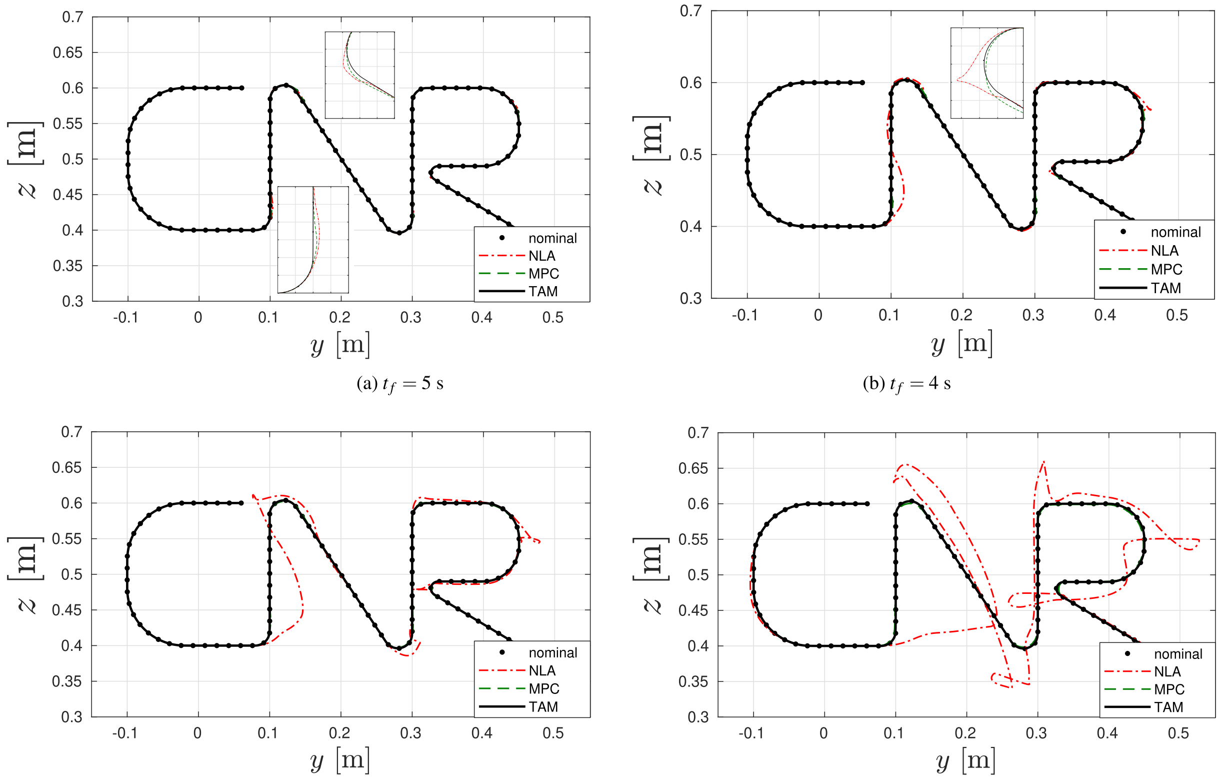
timing law γ^0 is a seven-trait-acceleration function as shown in Fig. 3. Except for the initial and final part of the trajectory, the longitudinal velocity of the timing law is constant, as typical of gluing and cutting.

Different values of t_f are evaluated to assess the performance of the algorithms under increasing severity of the desired trajectory (note that smaller values correspond to more demanding tasks, as the same path should be covered in less time). In a real-case scenario, the value of t_f and the corresponding velocity profile should be chosen by the offline trajectory planner (Step 2 in Section 1). Ideally, the optimal value of t_f is the minimum that does not lead to constraint violations but, especially when the available planning time is limited, an approximated solution is accepted. If the resulting trajectory is too slow, constraints will not probably be violated, but throughput of the process is not optimized. Vice versa, if the resulting trajectory is too fast, quality of the process is lost because of saturation. By using an online scaling algorithm in Step 3, Step 2 is allowed to compute fast trajectories that will be adapted online to respect the constraints.

4.3. Results

4.3.1. Path-following errors

Maximum and mean path-following errors ($e_{\max}$ and e_{mean}) are shown in Tables 1 and 2. In all cases, TAM and MPC have similar path-following errors, and they both significantly outperform NLA. In the least demanding task ($t_f = 5$ s), TAM and MPC's errors are around ten times smaller than NLA's error; in the most demanding task ($t_f = 2$ s), this difference grows of three orders of magnitude. These results can be better visualized in Fig. 4. For $t_f = 5$ s, all algorithms follow the desired path; zoomed areas show that NLA slightly deviates from the path during narrow curves (Fig. 4a). As the task becomes more demanding, NLA gives larger and larger deformations, while TAM and MPC are still able to preserve the desired geometry (Fig. 4b and c). For $t_f = 2$ s, the shape drawn by NLA is unrecognizable, while TAM and MPC follow the nominal path with errors comparable to $t_f = 5$ s.



(a) $t_f = 5$ s (b) $t_f = 4$ s (c) $t_f = 3$ s (d) $t_f = 2$ s

Fig. 4. Simulation results. Path performed by the different methods: nominal path (black dots); TAM (solid black lines); NLA (dotted red lines); MPC (dashed green lines). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 3
Trajectory execution time for different nominal trajectory times t_f.

t_f	t_{real}[s]			t_{real}/t_f			$\frac{t^{\mathrm{TAM}}_{\mathrm{real}}-t^{\mathrm{NLA}}_{\mathrm{real}}}{t_f}\cdot 100$
[s]	TAM	NLA	MPC	TAM	NLA	MPC	
5.0	5.79	5.42	5.25	1.15	1.08	1.05	+7.4
4.0	5.22	4.81	4.54	1.30	1.20	1.14	+10.2
3.0	4.88	4.36	3.94	1.63	1.45	1.31	+17.3
2.0	4.71	5.15	3.85	2.35	2.58	1.93	−22.0

The large errors of NLA are due to the inability of the scaling algorithm to slow down the trajectory in advance. On the contrary, TAM and MPC foresee the presence of steep changes and narrow curves and slows down the task if necessary. We can therefore conclude that the use of TAM allows the original algorithm to achieve path-following performance that are similar to look-ahead methods.

4.3.2. Execution times

Regarding the execution time, results for all tasks are shown in Table 3. The percentage difference between TAM and NLA is also computed as

$$\frac{t^{\mathrm{TAM}}_{\mathrm{real}}-t^{\mathrm{NLA}}_{\mathrm{real}}}{t_f}\cdot 100$$

where $t^{\mathrm{TAM}}_{\mathrm{real}}$ and $t^{\mathrm{NLA}}_{\mathrm{real}}$ are TAM and NLA's execution times. MPC gives the best results, as the look-ahead optimization-based strategy is able to reduce the slowdown. Apparently, NLA outperforms TAM in terms of execution time for $t_f = 3, 4, 5$ s. This is true for less demanding cases (*e.g.*, $t_f = 5$ s), for which the tracking error of all methods are satisfactory; for more demanding cases, however, the shorter execution time of NLA is jeopardized by significant path errors as visible from Fig. 4. For very demanding tasks (*e.g.*, $t_f = 2$ s), NLA execution time even overcomes TAM's execution time, because of the long time taken to converge to the final position when the error is very large (see $t_f = 2$ s). We can therefore conclude that the significant improvement of tracking accuracy provided by TAM partially comes at the cost of a slightly larger execution time

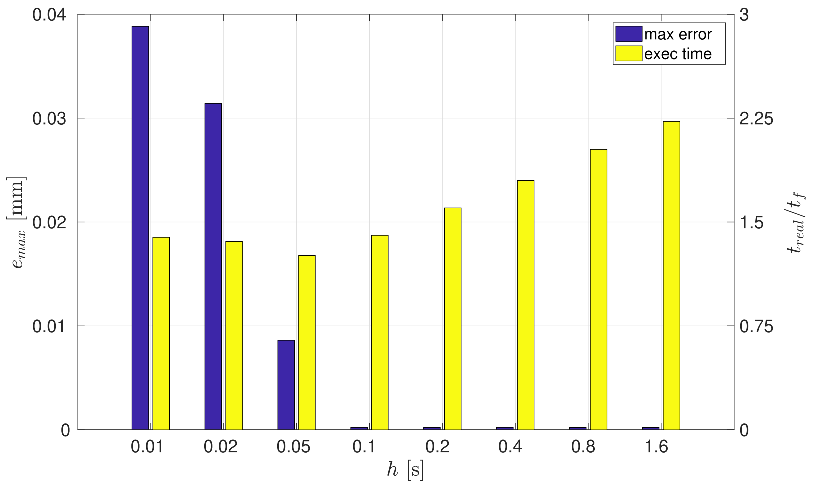


Fig. 5. Results for different length of the horizon h. Maximum position error (blue bars); ratio between the actual and the nominal execution time (yellow bars). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

with respect to NLA for less-demanding tasks. When the task is very demanding, however, TAM results are comparable with optimization-based look-ahead methods.

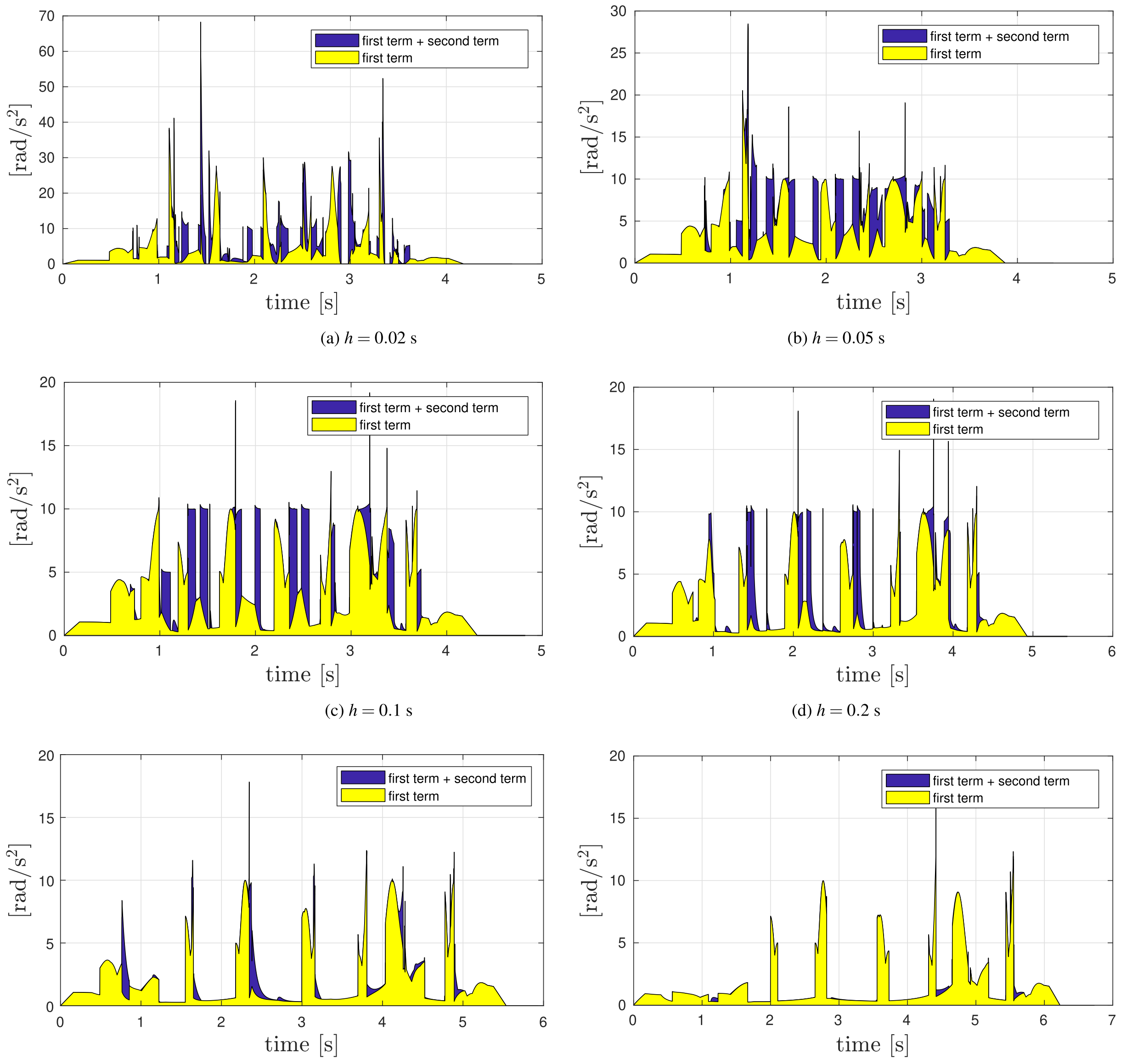
4.3.3. Effect of the length of the look-ahead window

The length of the look-ahead window h is a tunable parameter that should be chosen by the user. In Section 3, we assumed that its length was long enough for the system to reach the modified reference timing law with $\dot{\nu}_p = 0$. Moreover, the proposed method relies on the approximation $\gamma_p \approx \gamma(t_0) + h\dot{\gamma}^0(t_0)$ to compute the values of $q_d(\gamma_p)$, $q_d'(\gamma_p)$, $q_d''(\gamma_p)$ in (8)–(10). It is therefore clear that the choice of h affects the performance of the method: in principle, if h is too small, the modified timing law will not be feasible; if it is too large, the aforementioned approximation might lead to worsened results. In this paper, the choice of the length of h was made empirically owed the application at hand, and a systematic tuning procedure is beyond the scope of this paper; however, we run several tests with different values of h to show how its choice affects the validness of the performance of the method. In particular, we consider the trajectory shown in Fig. 2 with $t_f = 3$ s and we plot the results for different values of h ranging from 0.01 to 1.6 s. Fig. 5 shows the maximum error (blue bars) and the ratio between the execution and the nominal time t_{real}/t_f (yellow bars) for all values of h. We can observe a large error for short horizons (in principle, as $h \to 0$, TAM is equivalent to NLA). As the horizon grows, the error converges to smaller values. On the other hand, the execution time tends to increase in h, as the scaling method acts too early in slowing down the trajectory. As a rule of thumb, the correct value of h is the smallest that guarantees the desired path-following performance. From our experience with industrial manipulators, values ranging from 100 to 200 ms usually lead to satisfactory results.

To motivate the poor performance obtained with small values of h, we need to verify whether the approximation made from (16) to (17) is valid or not. To do this, recall (10):

$$\ddot{q}(t_p) = \underbrace{q_d''(\gamma_p)\,\nu^2}_{\text{first term}} \quad + \quad \underbrace{q_d'(\gamma_p)\,\dot{\nu}}_{\text{second term}} \tag{33}$$

Our approximation is based on the assumption that the second term of the equation is negligible with respect to the first one if the scaling method acts sufficiently in advance so that $\dot{\nu} \approx 0$ before the acceleration limit is hit. To verify it empirically, we compute the two terms of (33) *a posteriori* for different values of h. Results are in Fig. 6. The plots show the trend of the first and second terms (blue and yellow areas respectively) during the execution of the trajectory. It is possible to observe that for small values of h, the second term of (33) is relevant, and this motivates the bad performance of the method. For larger values of h, however, the first term is much more relevant than the second one, and the approximation is therefore valid. This leads to small path-following errors as shown by the simulation results of Section 4.3. It is worth noticing that the range of values suggested for h (100–200 ms) actually shows non-negligible values of the second term of (33) during brief portions of the trajectory; nonetheless, the method still has good performance. A possible reasons for this is that, during these transients, either the system is not in a critical condition with respect to the constraints, or the underlying non-look-ahead scaling method handles the constraint violation if it is not too severe.



(a) $h = 0.02$ s

(b) $h = 0.05$ s

(c) $h = 0.1$ s

(d) $h = 0.2$ s

(e) $h = 0.4$ s

(f) $h = 0.8$ s

Fig. 6. Evaluation of the approximation on (33); results for different length of the horizon h. First term of (33)(yellow area); sum of first and second terms of (33) (blue area). Results for $h = 0.01$ and 1.6 s are not shown for the sake of brevity. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

4.4. Computing time

Limiting the computing time is key in online implementation, as the method iterates at each sampling time. A qualitative analysis of the computing times taken by the different approaches is provided. Please note that the information on computing times serve only for a qualitative, rough comparison between the approaches; actual computing times depend on implementation aspects, and a deep statistical analysis about this is not within the scope of this paper. The tests were performed on a standard laptop mounting an i7-8565U CPU. Quadratic programs were solved by the parametric active-set solver QpOASES [31]. Table 4 shows the maximum, mean, and the standard deviation of the computing time taken to perform each cycle. As expected, NLA and TAM have comparable computing times. The reason is that most of the computing time of both methods is taken to solve OCP (32), while the growth in computation owed to the proposed reference modification is negligible. Compared to the look-ahead method (MPC), NLA and TAM give significantly smaller computing times. This is an expected result, as the look-ahead OCP is larger in terms of number of optimization variables and number of constraints.

Table 4
Comparison of computing times.

Method	$t_{\max}[\mu s]$	$t_{\text{mean}}[\mu s]$	std dev. $[\mu s]$
TAM	212	64	8
NLA	201	61	8
MPC	2320	560	55

Table 5
Translation error for different nominal trajectory times t_f.

(a) From planned motion

t_f	$e_{\max}$[m]			e_{mean}[m]		
[s]	TAM	NLA	MPC	TAM	NLA	MPC
5.0	$5.1{\cdot}10^{-5}$	$5.7{\cdot}10^{-5}$	$4.9{\cdot}10^{-5}$	$6.1{\cdot}10^{-6}$	$6.9{\cdot}10^{-6}$	$5.9{\cdot}10^{-6}$
3.5	$7.1{\cdot}10^{-5}$	$8.9{\cdot}10^{-5}$	$6.8{\cdot}10^{-5}$	$1.3{\cdot}10^{-5}$	$2.6{\cdot}10^{-5}$	$1.2{\cdot}10^{-5}$
3.0	$7.6{\cdot}10^{-5}$	$1.0{\cdot}10^{-4}$	$9.1{\cdot}10^{-5}$	$1.5{\cdot}10^{-5}$	$4.9{\cdot}10^{-5}$	$2.2{\cdot}10^{-5}$
2.5	$9.2{\cdot}10^{-4}$	$1.5{\cdot}10^{-2}$	$1.2{\cdot}10^{-4}$	$2.3{\cdot}10^{-5}$	$7.5{\cdot}10^{-4}$	$3.4{\cdot}10^{-5}$
2.0	$1.1{\cdot}10^{-4}$	$4.1{\cdot}10^{-2}$	$1.9{\cdot}10^{-4}$	$1.1{\cdot}10^{-5}$	$3.8{\cdot}10^{-3}$	$1.3{\cdot}10^{-5}$
1.5	$1.3{\cdot}10^{-4}$	$5.5{\cdot}10^{-2}$	$2.1{\cdot}10^{-4}$	$2.4{\cdot}10^{-5}$	$5.1{\cdot}10^{-3}$	$3.1{\cdot}10^{-5}$

(b) From measured motion

t_f	$e_{\max}$[m]			e_{mean}[m]		
[s]	TAM	NLA	MPC	TAM	NLA	MPC
5.0	$1.6{\cdot}10^{-3}$	$1.4{\cdot}10^{-3}$	$1.5{\cdot}10^{-3}$	$2.5{\cdot}10^{-4}$	$2.8{\cdot}10^{-4}$	$2.5{\cdot}10^{-4}$
3.5	$2.6{\cdot}10^{-3}$	$2.3{\cdot}10^{-3}$	$2.3{\cdot}10^{-3}$	$4.5{\cdot}10^{-4}$	$4.5{\cdot}10^{-4}$	$4.1{\cdot}10^{-4}$
3.0	$3.3{\cdot}10^{-3}$	$2.0{\cdot}10^{-2}$	$3.0{\cdot}10^{-3}$	$6.1{\cdot}10^{-4}$	$1.8{\cdot}10^{-3}$	$5.9{\cdot}10^{-4}$
2.5	$3.7{\cdot}10^{-3}$	$6.1{\cdot}10^{-2}$	$3.6{\cdot}10^{-3}$	$6.9{\cdot}10^{-4}$	$9.3{\cdot}10^{-3}$	$8.5{\cdot}10^{-4}$
2.0	$2.4{\cdot}10^{-3}$	$6.1{\cdot}10^{-2}$	$3.1{\cdot}10^{-3}$	$4.5{\cdot}10^{-4}$	$9.6{\cdot}10^{-3}$	$4.4{\cdot}10^{-4}$
1.5	$2.3{\cdot}10^{-3}$	$7.6{\cdot}10^{-2}$	$4.0{\cdot}10^{-3}$	$5.1{\cdot}10^{-4}$	$9.6{\cdot}10^{-3}$	$7.7{\cdot}10^{-4}$

Table 6
Orientation error for different nominal trajectory times t_f.

(a) From planned motion

t_f	$e_{\max}$[rad]			e_{mean}[rad]		
[s]	TAM	NLA	MPC	TAM	NLA	MPC
5.0	$2.6{\cdot}10^{-6}$	$2.6{\cdot}10^{-6}$	$1.9{\cdot}10^{-6}$	$1.0{\cdot}10^{-6}$	$1.0{\cdot}10^{-6}$	$7.1{\cdot}10^{-7}$
3.5	$7.0{\cdot}10^{-6}$	$8.6{\cdot}10^{-6}$	$6.7{\cdot}10^{-6}$	$2.2{\cdot}10^{-6}$	$3.6{\cdot}10^{-6}$	$1.2{\cdot}10^{-6}$
3.0	$2.4{\cdot}10^{-5}$	$4.3{\cdot}10^{-3}$	$2.1{\cdot}10^{-5}$	$3.2{\cdot}10^{-6}$	$1.4{\cdot}10^{-4}$	$1.1{\cdot}10^{-6}$
2.5	$2.5{\cdot}10^{-5}$	$3.0{\cdot}10^{-2}$	$6.0{\cdot}10^{-5}$	$3.1{\cdot}10^{-6}$	$3.7{\cdot}10^{-3}$	$8.9{\cdot}10^{-6}$
2.0	$2.2{\cdot}10^{-5}$	$1.2{\cdot}10^{-1}$	$8.3{\cdot}10^{-4}$	$3.0{\cdot}10^{-6}$	$2.2{\cdot}10^{-2}$	$1.8{\cdot}10^{-4}$
1.5	$2.4{\cdot}10^{-5}$	$1.8{\cdot}10^{-1}$	$2.3{\cdot}10^{-3}$	$3.0{\cdot}10^{-6}$	$3.4{\cdot}10^{-2}$	$3.4{\cdot}10^{-4}$

(b) From measured motion

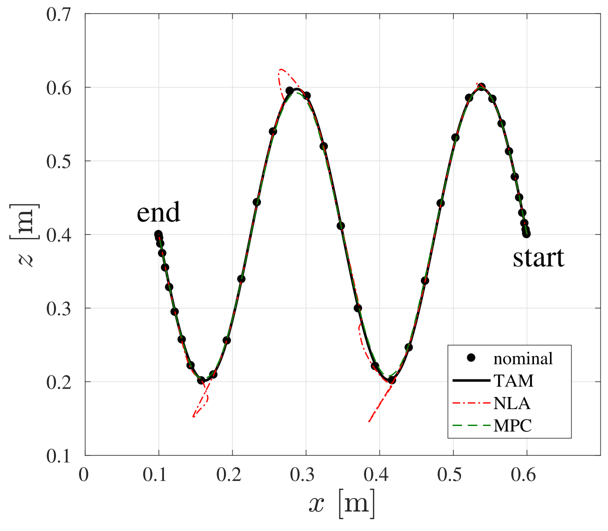
t_f	$e_{\max}$[rad]			e_{mean}[rad]		
[s]	TAM	NLA	MPC	TAM	NLA	MPC
5.0	$3.6{\cdot}10^{-3}$	$3.6{\cdot}10^{-3}$	$3.5{\cdot}10^{-3}$	$7.0{\cdot}10^{-4}$	$7.1{\cdot}10^{-4}$	$6.7{\cdot}10^{-4}$
3.5	$7.7{\cdot}10^{-3}$	$7.9{\cdot}10^{-3}$	$6.8{\cdot}10^{-3}$	$1.2{\cdot}10^{-3}$	$1.3{\cdot}10^{-3}$	$1.4{\cdot}10^{-3}$
3.0	$8.3{\cdot}10^{-3}$	$1.7{\cdot}10^{-2}$	$6.4{\cdot}10^{-3}$	$1.9{\cdot}10^{-3}$	$2.6{\cdot}10^{-3}$	$2.0{\cdot}10^{-3}$
2.5	$1.3{\cdot}10^{-2}$	$3.5{\cdot}10^{-2}$	$9.9{\cdot}10^{-3}$	$2.1{\cdot}10^{-3}$	$4.5{\cdot}10^{-3}$	$2.2{\cdot}10^{-3}$
2.0	$7.2{\cdot}10^{-3}$	$6.3{\cdot}10^{-2}$	$9.9{\cdot}10^{-3}$	$1.3{\cdot}10^{-3}$	$8.3{\cdot}10^{-3}$	$3.0{\cdot}10^{-3}$
1.5	$7.0{\cdot}10^{-3}$	$8.2{\cdot}10^{-2}$	$1.2{\cdot}10^{-2}$	$1.2{\cdot}10^{-3}$	$1.1{\cdot}10^{-2}$	$1.4{\cdot}10^{-3}$

5. Experimental results

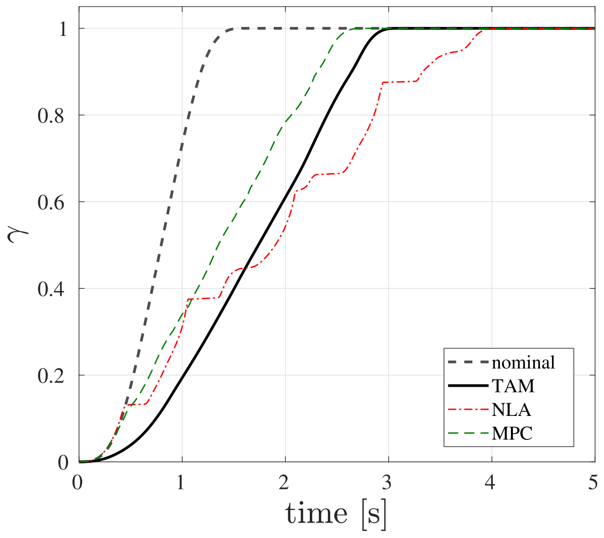
In this section, we assess the algorithms' performance on a real robot with a (non-ideal) controller to verify whether the improvements given by the proposed method are still significant, despite the tracking error introduced by the motion controller. The experiments are shown also in the video accompanying this paper.

5.1. Setup

Experimental tests were performed on a 6-degree-of-freedom manipulator. The setup consists of a Universal Robots UR10, version 3.5, controlled by means of a ROS-based control architecture. The controller consists of a cascade scheme: a position



(a) Path



(b) Timing law

Fig. 7. Resulting path and timing law for $t_f = 1.5$ s.

control loop runs in ROS Kinetic on Ubuntu 16.04 and communicates with the robot velocity controller via a TCP connection with sampling rate equal to 125 Hz. The proposed method could actually run at higher sampling rates (according to the computing times shown in Section 4.4), but 125 Hz is the maximum sampling rate accepted by TCP interface of the robot at hand, which therefore acts as a bottleneck for the whole system. The outer loop is a proportional controller with sampling rate equal to 125 Hz and gain equal to 7, plus a velocity feedforward term. The inner velocity loop is the built-in control system. The outputs of the tested methods are given as reference to the low-level controller with sampling period $T = 8$ ms (*i.e.*, the sampling period of the outer loop). The measured robot configuration is acquired from the robot encoders with sampling rate equal to 125 Hz, and the velocity and acceleration are obtained *a posteriori* via numerical differentiation. Torques are estimated by multiplying the motor currents by the torque constant of each motor. Currents are measured through the robot built-in current sensor and communicated at 125 Hz through the TCP interface of the robot; the values of the torque constants of each motor can be found in [28]. As already mentioned, the dynamics parameters needed in the torque limits (23) are publicly available on the website of the robot manufacturer and are discussed in details in [28]. Velocity, acceleration, and torque limits have been set to (29).

Table 7
Trajectory execution time for different nominal trajectory times t_f.

t_f	t_{real}[s]			t_{real}/t_f			$\frac{t_{\text{real}}^{\text{TAM}}-t_{\text{real}}^{\text{NLA}}}{t_f}\cdot 100$
[s]	TAM	NLA	MPC	TAM	NLA	MPC	
5.0	5.00	5.00	5.00	1.00	1.00	1.00	0.0
3.5	3.50	3.50	3.50	1.00	1.00	1.00	0.0
3.0	3.16	3.09	3.06	1.05	1.03	1.02	+2.2
2.5	3.04	3.20	2.84	1.22	1.28	1.14	-6.3
2.0	3.03	3.70	2.78	1.51	1.85	1.39	-33.7
1.5	2.94	3.84	2.63	1.96	2.56	1.75	-60.3

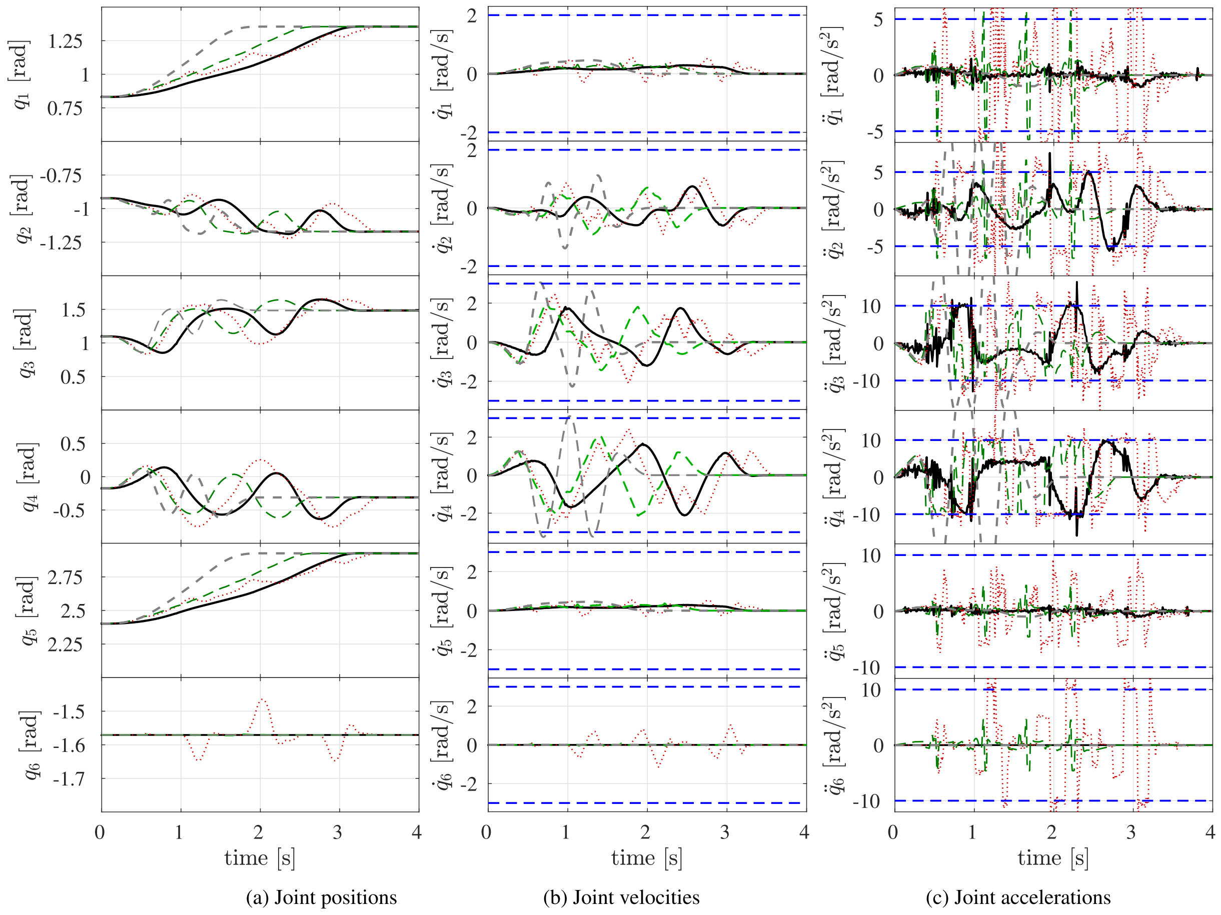


Fig. 8. Measured joint positions, velocities, and accelerations for $t_f = 2$ s. Dashed gray line: nominal reference. Solid black line: TAM. Dotted red line: NLA. Dashed green line: MPC. Dashed blue line: acceleration limits. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

5.2. Compared methods, tests, and settings

The proposed method (TAM), the non-look-ahead method (NLA), and the optimization-based look-ahead method (MPC) are compared as described in Section 4.2. Maximum and mean path-following errors ($e_{\max}$ and e_{mean}) and the execution time (t_{real}) are evaluated. Path-following errors are computed for both the planned and measured trajectories. The former refers to the trajectory computed by the scaling algorithms and then given as reference to the robot controller. The latter are computed from the actual joint coordinates measured by the joints' encoders. Cartesian positions and orientations are then computed by applying forward kinematics to the joint coordinates at each time instant. Definitions of $e_{\max}$, e_{mean}, and t_{real} are given in Section 4.2.

The robot task is a sine curve in the *xz*-plain defined as:

$$\Gamma_{xyz} = \left(\pi/3 + 0.6 - 0.5\gamma,\ \ 0.8,\ \ 0.4 + 0.2\,\sin(4\pi\gamma) \right) \text{m}, \tag{34}$$

where $\gamma \in [0, 1]$. During the task, the end-effector orientation should be orthogonal to the *yz*-plain. The nominal timing law γ^0 is a seven-trait-acceleration function as shown in Fig. 3. Different values of t_f are tested (note that the trajectory becomes more and more demanding as t_f decreases).

5.3. Results

Results are shown in Tables 5 and 6. The case $t_f = 5$ s provides a nominal trajectory that is always within the robot limits. It is therefore taken as reference case to evaluate the performance of the following trajectories. For the less demanding case ($t_f = 3.5$ s), all methods give similar results and are successful in slowing down the trajectory to preserve the geometrical path (errors are comparable to the non-saturated case $t_f = 5$ s). As the nominal trajectories get more demanding, NLA's errors grow significantly, while TAM and MPC are still able to follow the path accurately. For example, for $t_f = 1.5$ s, TAM's path-following error is in the order of 0.1 mm, while NLA drifts from the nominal path of about 5 cm. This can be noticed in Fig. 7(a), which shows the path performed by the three planners: TAM and MPC follow the curve accurately; NLA gives large overshoots during narrow curves, according to the behaviors analyzed in Section 4.3. It is worth stressing that the improvement due to the proposed method is evident also from the measured data (Tables 5b and 6b). In other words, TAM's improvement is relevant not only under ideal conditions (*i.e.*, with ideal motion control as in Section 4), but also when the motion tracking error is taken into account.

Regarding the execution times, simulation results are confirmed again. Results are shown in Table 7. MPC gives overall better results. TAM and NLA have comparable execution times in less demanding tasks, while TAM outperforms NLA as the task becomes more demanding (up to -60% for $t_f = 1.5$ s). This is clear also from Fig. 7(b), which shows the resulting timing laws for $t_f = 1.5$ s: MPC and TAM's timing law resembles the nominal one (dilated in time); MPC is the faster among the three; NLA has an irregular and slow trend.

Finally, as an example, the trends of the joint configuration, velocities, and accelerations for $t_f = 2$ s are represented in Fig. 8. Dashed gray lines are the nominal timing laws (γ^0 and its velocity and acceleration). From Fig. 8(c), it is interesting to point out that the nominal timing law significantly exceeds the acceleration limits (dashed blue lines). Nonetheless, thanks to the scaling algorithms, the joint accelerations and velocities are always kept within the bounds, except for few acceleration spikes due to measurement noise and the motion controller actions. It is also worth stressing that TAM's trends are much smoother than NLA's ones, with consequent minor stress of the robot actuators.

6. Conclusions

We have proposed a novel trajectory scaling method that modifies the reference timing law based on future information about the desired task. This can be used to confer a predictive feature on the existing trajectory scaling algorithms, but without falling into the computational disadvantages of model predictive control methods (*i.e.*, higher computational burden and the need of solving nonlinear optimization problems). Simulation and experimental results have shown that the proposed method significantly improves the path-following error and the total execution time of the trajectories if compared with non-look-ahead methods, and it shows lower computing time compared to optimization-based look-ahead methods.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was partially supported by ShareWork project (H2020, European Commission – G.A. 820807) and Pickplace project (H2020, European Commission – G.A. 780488).

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.mechmachtheory.2020.104152.

References

[1] M. Elbanhawi, M. Simic, Sampling-based robot motion planning: a review, IEEE Access 2 (2014) 56–77.
[2] J.D. Gammell, S.S. Srinivasa, T.D. Barfoot, Informed RRT*: optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic, in: 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2014, pp. 2997–3004.
[3] S. Zhang, A.M. Zanchettin, R. Villa, S. Dai, Real-time trajectory planning based on joint-decoupled optimization in human-robot interaction, Mech. Mach. Theory 144 (2020) 103664.
[4] N. Zhang, W. Shang, Dynamic trajectory planning of a 3-DOFunder-constrained cable-driven parallel robot, Mech. Mach. Theory 98 (2016) 21–35.
[5] C. Byner, B. Matthias, H. Ding, Dynamic speed and separation monitoring for collaborative robot applications–concepts and performance, Robot. Comput. Integr. Manuf. 58 (2019) 239–252.

[6] J.A. Marvel, R. Norcross, Implementing speed and separation monitoring in collaborative robot workcells, Robot. Comput. Integr. Manuf. 44 (2017) 144–155.
[7] D. Verscheure, B. Demeulenaere, J. Swevers, J. De Schutter, M. Diehl, Time-optimal path tracking for robots: a convex optimization approach, IEEE Trans. Autom. Control 54 (2009) 2318–2327.
[8] Q. Pham, A general, fast, and robust implementation of the time-optimal path parameterization algorithm, IEEE Trans. Robot. 30 (6) (2014) 1533–1540.
[9] A. Gallant, C. Gosselin, Extending the capabilities of robotic manipulators using trajectory optimization, Mech. Mach. Theory 121 (2018) 502–514.
[10] J. Huang, P. Hu, K. Wu, M. Zeng, Optimal time-jerk trajectory planning for industrial robots, Mech. Mach. Theory 121 (2018) 530–544.
[11] M. Boryga, A. Graboś, Planning of manipulator motion trajectory with higher-degree polynomials use, Mech. Mach. Theory 44 (7) (2009) 1400–1419.
[12] Ü. Dinşer, M. Çevik, Improved trajectory planning of an industrial parallel mechanism by a composite polynomial consisting of Bézier curves and cubic polynomials, Mech. Mach. Theory 132 (2019) 248–263.
[13] Y. Abbasi-Yadkori, J. Modayil, C. Szepesvari, Extending rapidly-exploring random trees for asymptotically optimal anytime motion planning, in: 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2010, pp. 127–132.
[14] ISO/TS 15066:2016 Robots and Robotic Devices–Collaborative RobotsStandard, International Organization for Standardization, Geneva, CH, 2016.
[15] B. Matthias, T. Reisinger, Example application of ISO/TS 15066 to a collaborative assembly scenario, in: Proceedings of ISR 2016: 47st International Symposium on Robotics, VDE, 2016, pp. 1–5.
[16] G. Antonelli, S. Chiaverini, G. Fusco, A new on-line algorithm for inverse kinematics of robot manipulators ensuring path tracking capability under joint limits, IEEE Trans. Robot. Autom. 19 (2003) 162–167.
[17] F. Flacco, A. De Luca, O. Khatib, Control of redundant robots under hard joint constraints: saturation in the null space, IEEE Trans. Robot. 31 (2015) 637–654.
[18] C. Guarino Lo Bianco, O. Gerelli, Online trajectory scaling for manipulators subject to high-order kinematic and dynamic constraints, IEEE Trans. Robot. 27 (2011) 1144–1152.
[19] C. Guarino Lo Bianco, F. Ghilardelli, A discrete-time filter for the generation of signals with asymmetric and variable bounds on velocity, acceleration, and jerk, IEEE Trans. Ind. Electron. 61 (2014) 4115–4125.
[20] C. Guarino Lo Bianco, F. Ghilardelli, A scaling algorithm for the generation of jerk-limited trajectories in the operational space, Robot. Comput. Integr. Manuf. 44 (2017) 284–295.
[21] C. Guarino Lo Bianco, F. Ghilardelli, Techniques to preserve the stability of a trajectory scaling algorithm, in: Proceedings of the IEEE International Conference on Robotics and Automation, 2013, pp. 870–876. Karlsruhe (Germany)
[22] T. Faulwasser, T. Weber, P. Zometa, R. Findeisen, Implementation of nonlinear model predictive path-following control for an industrial robot, IEEE Trans. Control Syst. Technol. 25 (2017) 1505–1511.
[23] M. Faroni, M. Beschi, N. Pedrocchi, A. Visioli, Predictive inverse kinematics for redundant manipulators with task scaling and kinematic constraints, IEEE Trans. Robot. 35 (1) (2019) 278–285.
[24] M. Faroni, M. Beschi, C. Guarino Lo Bianco, A. Visioli, Predictive joint trajectory scaling for manipulators with kinodynamic constraints, Control Eng. Pract. 95 (2020) 104264.
[25] A.M. Zanchettin, N.M. Ceriani, P. Rocco, H. Ding, B. Matthias, Safety in human-robot collaborative manufacturing environments: metrics and control, IEEE Trans. Autom. Sci. Eng. 13 (2) (2016) 882–893.
[26] M. Lippi, A. Marino, Safety in human-multi robot collaborative scenarios: a trajectory scaling approach, in: Proc. IFAC SYROCO, 2018. Budapest (Hungary)
[27] M. Faroni, M. Beschi, N. Pedrocchi, An MPC framework for online motion planning in human-robot collaborative tasks, in: Proceedings of the IEEE International Conference on Emerging Technologies and Factory Automation, 2019. Zaragoza (Spain)
[28] C. Gaz, E. Magrini, A. De Luca, A model-based residual approach for human-robot collaboration during manual polishing operations, Mechatronics 55 (2018) 234–247.
[29] F.-T. Cheng, T.-H. Chen, Y.-Y. Sun, Resolving manipulator redundancy under inequality constraints, IEEE Trans. Robot. Autom. 10 (1994) 65–71.
[30] F. Caccavale, B. Siciliano, L. Villani, The role of Euler parameters in robot control, Asian J. Control 1 (1999) 25–34.
[31] H. Ferreau, C. Kirches, A. Potschka, H. Bock, M. Diehl, qpOASES: a parametric active-set algorithm for quadratic programming, Math. Program. Comput. 6 (2014) 327–363.