Predictive Joint Trajectory Scaling for Manipulators with Kinodynamic Constraints

Marco Faroni, Manuel Beschi, Corrado Guarino Lo Bianco, Antonio Visioli

Paper accepted for publication in Control Engineering Practice on November 25, 2019.

Predictive Joint Trajectory Scaling for Manipulators with Kinodynamic Constraints

Marco Faroni[a], Manuel Beschi[a], Corrado Guarino Lo Bianco[b], Antonio Visioli[c,*]

[a] *Istituto di Sistemi e Tecnologie Industriali Intelligenti per il Manifatturiero Avanzato, National Research Council, Milan, Italy.*
[b] *Dipartimento di Ingegneria e Architettura, Università degli Studi di Parma, Parma, Italy.*
[c] *Dipartimento di Ingegneria Meccanica e Industriale, University of Brescia, Brescia, Italy.*

Abstract

Trajectory scaling techniques adapt online the robot timing law to preserve the desired geometric path when the desired motion does not respect the robot limits. State-of-the-art non-predictive methods typically provide far-from-optimal solutions, while high computational burdens are the main bottleneck for the implementation of receding horizon strategies. This paper proposes a predictive approach to trajectory scaling subject to joint velocity, acceleration, and torque limitations. Computational complexity is dramatically reduced by means of the parametrization of inputs and outputs and the iterative linearization of the optimal control problem around the previous output prediction. This allows the online implementation of the method for sampling periods in the order of one millisecond. Numerical and experimental results on a six-degree-of-freedom robot show the effectiveness of the method.

Keywords: Motion planning, robot manipulators, trajectory scaling, joint constraints, predictive control.

1. Introduction

Throughput maximization plays a key role in industrial processes. Robot manipulators are often taken to the limit and decrements of the tracking performance arise when the required task is too demanding with respect to the robot physical capabilities. This may worsen the quality of the process.

The design of the task is usually decomposed into two stages. First, the geometric path is devised, accounting for high-level requirements (*e.g.*, task specifications, obstacle avoidance). Second, the timing law along such path is defined as a suitable mono-dimensional function over time.

Limitations of the robot kinematics and dynamics variables are usually considered in this second step. As a general approach, such limits can be included in an optimization problem, which outputs the optimized velocity profile along the geometric curve [1, 2, 3, 4, 5]. Such approach suffers from high computational burdens and needs to know the whole trajectory *a priori*. Offline implementation is therefore required, but this is a relevant limit in view of recent developments in the robotic field, where the motion is often planned or modified online (*e.g.*, collaborative robotics and robots in unstructured environments).

A possible way to tackle this issue consists in adapting the velocity profile of the robot online according to the robot limits and changes in the environment. Diverse methods have been presented in the literature under the name of *online trajectory scaling* or *path following control*.

Online trajectory scaling comes from a motion plan-

*Corresponding author
Email addresses: marco.faroni@stiima.cnr.it (Marco Faroni), manuel.beschi@stiima.cnr.it (Manuel Beschi), guarino@ce.unipr.it (Corrado Guarino Lo Bianco), antonio.visioli@unibs.it (Antonio Visioli)

ning context. It assumes the robot is equipped with a feedback controller and feeds it with an open-loop reference. This approach is clearly oriented to robot manipulators which come with a robust (often closed-architecture) motion controller. It is worth pointing out that, in this way, stability issues do not arise, as the underlying controller is robust and stable. Moreover, in trajectory scaling both path and timing law are always specified, and the algorithm adapts the motion by giving priority to the geometric reference, but still trying to follow the given velocity profile at best. The reason for having a fully-specified trajectory comes from practical considerations. For example, in additive manufacturing, the amount of deposited material linearly depends on the robot speed. Thus, the nominal timing law is a technological requirement and it should be modified as less as possible. Moreover, timing of the robotic processes often comes from the scheduling of the tasks and should be relaxed only to prevent quality degradation of the process or due to safety issues.

Path following control addresses the general problem of moving a dynamic system along a reference geometric curve using the velocity profile as an additional control variable [6, 7, 8, 9, 10]. These methods are intended to be implemented as feedback controllers. They therefore require stability and robustness and the possibility of acting directly on the low-level controller of the robot. However, nothing prevents one from using them to generate an open-loop reference for the robot controller. Moreover, path following control usually computes the velocity profile from scratch according to a given criterion (usually the minimization of the execution time).

Both online trajectory scaling and path following methods are implemented with sampling periods in the order of few milliseconds. To do so, they sacrifice optimality in favor of low computational complexity. Most scaling methods in the literature uses a non-predictive approach: the motion is modified without taking into account the nominal trajectory and the presence of constraints at future time instants; only the desired motion at the current time instant is considered. Examples of such approach can be found in [11, 12, 13, 14, 15, 16, 17]. The drawback is that the solution is only locally optimal with respect to the global execution of the task. As a matter of fact, without taking into account the future evolution of the trajectory and the constraints, they cannot prevent the system from disadvantageous situations, which could lead to infeasibility and saturation of the actuators. This issue can be mitigated, for example, by using a receding horizon approach. In this respect, [18] proposed a predictive inverse kinematics resolution method for robotic manipulators under position, velocity, and acceleration limits. This method also considers task scaling in order to preserve the nominal geometric task in case saturations cannot be avoided. The method is implemented online with sampling periods in the order of one millisecond also on highly redundant manipulators.

Receding horizon strategies have also been investigated in path following control. In particular, [9] proposed a Nonlinear MPC (NMPC) path-following approach where also the case of assigned velocity profile is addressed. In [10], the method is implemented on a three-degree-of-freedom manipulator. To reach real-time computing speed, the nonlinear input function is parametrized as a staircase function whose intervals are ten times larger than the sampling period, then the optimal control problem is solved by means of a single-shooting Sequential Quadratic Programming (SQP) solver with maximum number of iterations equal to one. It is worth pointing out that the computational time exponentially grows in the number of variables and analytical equations of the kinematics and dynamics of the robot and their derivatives is not always viable for manipulators with many degrees of freedom.

In this work, we propose a different approach that allows the real-time implementation also on complex manipulators. First of all, the method specifically addresses the trajectory scaling problem (*i.e.*, both path and velocity profile are specified and the method generates online a reference for the robot controller). The proposed method is based upon the receding horizon approach described in [18] and aims to include also torque boundaries in the MPC framework. The method consider a virtual model of the manipulator where the joint accelerations are the control variables (*i.e.*, each joint is modeled as a double integrator). In this way, the predictive equations of the joint velocities and positions are linear in the control variables. The trajectory scaling property is expressed at velocity level: the nominal joint velocity is multiplied by a scalar variable; by setting it to a value smaller than one, it is possible to slow down the trajectory without modifying the path. Torque limits are considered via the inverse dynamics equations of the manipulator. The resulting non-

linear constraints are then linearized at each iteration by using joint positions and velocities predicted at the previous cycle, similarly to [8, 19, 20]. The use of a zero-order linearization reduces the computational burden compared to standard SQP solvers.

Although the method proposed up to this point reduces the online optimization problem to a Quadratic Program (QP), robotic systems typically present very fast sampling periods, and this represents the main limit to the application of MPC in this field. As a matter of fact, long-enough horizons imply a large number of optimization variables, with consequent high computing times [21]. In order to speed up the algorithm, the computational complexity of the QP needs to be reduced. Several complexity reduction techniques have been proposed in the literature. The most common approach consists in the parametrization of the input function, in order to reduce the number of variables involved in the optimization [22, 23]. As already mentioned, in [10] inputs are parametrized as step-wise constant functions with discretization intervals larger than the sampling period. However, from the perspective of receding horizon control, a parametrization which gives more importance to the first part of the predictive horizon might improve the overall results. An alternative approach was proposed in [19, 24], where inputs and outputs were parametrized as continuous-time staircase functions with non-constant intervals. In this way, a parametrization with shorter intervals at the beginning of the horizon and larger intervals at the end can be chosen. Following this idea, in this paper, a discrete-time approach is devised to obtain a staircase parametrization of inputs and outputs. Compared to [19, 24], the derivation and the implementation are much simpler, for they only need the construction of two binary matrices, without modifying the classic derivation of the predictive equations. Such matrices can be easily computed offline by following the algorithmic procedures presented in the paper.

Summarizing, differently from other receding horizon approaches (*e.g.*, [10]), the proposed method uses a faster linearization method (the zero-order linearization does not require the computation of the gradients of constraints and cost functions), and the choice of the joint acceleration as virtual input and the velocity-based cost function reduce the complexity of the control problem. Moreover, the use of the uneven distribution of the nodes along the horizon improves the performance of the method, especially for longer horizons. These advantages are highlighted by the numerical and experimental results presented in this work.

The paper is organized as follows. Section 2 defines the trajectory scaling problem to be addressed. Section 3 illustrates the proposed method. Then, numerical results on a Universal Robot UR10 manipulator are discussed in Section 4.1. In particular, the method is tested on different trajectories and the results are compared against a state-of-the-art non-predictive method and [16, 17] and the NMPC path-following method [10]. Moreover, different implementations and the effect of different input parametrizations are discussed. Section 4.2 shows experimental results on a Universal Robot UR10 robot. Conclusions are presented in Section 5. Finally, the complexity reduction technique adopted for the implementation of the method is described in Appendix A.

2. Preliminaries

Consider a robot manipulator with n joints and denote with q, $\dot{q}$, $\ddot{q}$, and $\tau \in \mathbb{R}^n$ its joint configuration, velocity, acceleration, and torque vectors, respectively. The dynamics of the robot is described by the model:

$$\dot{x} = f(x,u) \tag{1}$$

where $x := (q, \dot{q}) \in \mathbb{R}^{2n}$ and $u := \tau$. The robot is subject to limitations of its states and commands. Typically, the following constraints are considered:

$$\dot{q}_{\min} \leq \dot{q} \leq \dot{q}_{\max} \tag{2}$$

$$\ddot{q}_{\min} \leq \ddot{q} \leq \ddot{q}_{\max} \tag{3}$$

$$\tau_{\min} \leq \tau \leq \tau_{\max} \tag{4}$$

where $\dot{q}_{\min}$, $\ddot{q}_{\min}$, $\tau_{\min} \in \mathbb{R}^n$ are the lower velocity, acceleration, and torque limits of the joints, and $\dot{q}_{\max}$, $\ddot{q}_{\max}$, $\tau_{\max} \in \mathbb{R}^n$ are upper velocity, acceleration, and torque limits of the joints.

Consider now a desired curve q_d in the joint space, such that

$$q_d : \Sigma \subseteq \mathbb{R} \to \mathbb{R}^n, \quad s \mapsto q_d := q_d(s) \tag{5}$$

where s parametrizes the curve. Curve q_d represents the geometrical path in the joint space, while the trend of variable s over the time determines the timing law (*i.e.*, the velocity profile) along the path.

In path following control, s is used as an additional control variable to minimize a certain criterion (*e.g.*, the total time). However, when it comes to robot manipulators, the timing law along the path is usually assigned due to technological requirements. In particular, the trajectory is completely defined by assigning a reference timing law s_{ref}. It is common to allow a relaxation of the timing law in case the desired motion is infeasible with respect to the robot limits. This allows avoiding geometrical errors that would arise due to saturation of the joint limits. The approach proposed in [9] addresses such problem by devising an NMPC strategy. Given the desired velocity profile $\dot{s}_{\text{ref}}$, the resulting Optimal Control Problem (OCP) is:

$$\underset{\tau,s}{\text{minimize}} \int_{t_0}^{t_o+T_p} \left(\left\| \left(q - q_d(s)\,,\, \dot{s} - \dot{s}_{\text{ref}}\right) \right\|_Q^2 + \left\| (\tau, \ddot{s}) \right\|_R^2 \right) dt \tag{6}$$

subject to (1), (2), (3), (4) and where t_0 is the current time instant, $T_p > 0$ is the predictive horizon, Q and R are positive definite matrices, and $\|x\|_Q^2 = x^T Q x$. The OCP minimizes the geometrical error and the deviation with respect to the assigned velocity in a weighted fashion. Priority can be given to the former by adjusting the weights in Q.

A different widespread approach consists in considering a virtual kinematic model of the robot (*i.e.*, each joint is chain of integrators) and adapting the desired motion at the current iteration to respect the robot limits. To this purpose, differentiating (5) with respect to time gives:

$$\dot{q}_d = q_d'(s)\,\dot{s} \tag{7}$$

where $q_d' = dq_d/ds$ and $\dot{s} = ds/dt$. The term q_d' is geometrically tangent to curve q_d, while the value of the scalar $\dot{s}$ determines the amplitude of vector $\dot{q}_d$. Thus, by varying the value of $\dot{s}$, the timing law can be punctually modified, while keeping the same geometrical direction.

A possible choice of the curve parametrization s is the time of the nominal trajectory. In this case, q_d coincides with the nominal trajectory parametrized with respect to the desired execution time and $q_d'(s)$ coincides with the nominal joint velocity profile. In the discrete-time scaling method (with sampling period T), a decision variable v can be defined in such a way that:

$$s(k+1) = s(k) + T v(k) \tag{8}$$

and at each sampling time, v scales all the desired joint velocities by the same factor, thus deforming the velocity profile but preserving the curve shape. When no scaling occurs, $v(k)$ is equal to 1 and the joint velocity is the same as the desired one, whereas $v(k) < 1$ slows down the trajectory and $v(k) > 1$ speeds it up. By choosing a proper value of $v(k)$, the current reference motion can be made feasible with respect to the robot limits. This principle is exploited, for example, in [11, 12]. Notice that an approach of this kind is computationally very light (the online control problem comes down to a small QP as shown in [12]), however it is clear that the solution might be very far from being optimal, as the problem only considers the reference motion one step at a time. To tackle this issue, in this work, we devise a receding horizon method based on such kinematic approach.

3. Predictive trajectory scaling

3.1. Kinematic model of the manipulator

We consider a kinematic model of the manipulator in which the joint acceleration is considered as a virtual input to the system, while the motor torque limits are considered as nonlinear constraints. To do so, each joint is modeled as a chain of two discrete-time integral systems with sampling period T and all joints are coupled in a single state-space representation. The following virtual linear model of the robot therefore results:

$$\begin{aligned} x(k+1) &= A\,x(k) + B\,u(k) \\ q &= \begin{pmatrix} I_n & 0_{n\times n} \end{pmatrix} x \\ \dot{q} &= \begin{pmatrix} 0_{n\times n} & I_n \end{pmatrix} x \end{aligned} \tag{9}$$

where $x := (q, \dot{q}) \in \mathbb{R}^{2n}$, $\ddot{q} = u$ and:

$$A := \begin{pmatrix} I_n & T I_n \\ 0_{n\times n} & I_n \end{pmatrix} \in \mathbb{R}^{2n\times 2n}, \tag{10}$$

$$B := \begin{pmatrix} 0.5T^2 I_n & T I_n \end{pmatrix}^T \in \mathbb{R}^{2n\times n}, \tag{11}$$

where I_n and $0_{n\times n}$ denote the $n \times n$ identity and null matrices respectively.

The relation between the robot states and the joint torques is expressed via the inverse dynamics of the manipulator in the form:

$$\tau = H(q)\ddot{q} + l(q, \dot{q}) \tag{12}$$

where $H(q) \in \mathbb{R}^{n\times n}$ is the symmetric positive-definite inertia matrix, and $l(q,\dot{q}) \in \mathbb{R}^n$ takes into account centrifugal, Coriolis, friction, gravitational and external torques.

3.2. Optimal control problem

Consider task (5) where s is chosen as the time of the nominal trajectory. Consider now a predictive horizon of $p \in \mathbb{N}$ sampling periods ahead such that $T_p = pT$. To extend the scaling principle described in Section 2 to the predictive framework, we consider (7) at each sampling time in the horizon, that is:

$$\dot{q}(k+i) = v(k+i)\, q_d'\big(s(k+i)\big) \quad \forall i = 1, \dots, p. \tag{13}$$

If this equation holds, the robot moves along the desired geometrical curve. Moreover, as mentioned in Section 2, if $v = 1$, then also the assigned velocity profile is met.

Thus, a minimization problem to achieve this behavior can be written as:

$$\underset{u,v}{\text{minimize}} \sum_{i=1}^{p} \Big\| \big(\dot{q}(k+i) - q_d'(s(k+i))v(k+i)\,,\, 1 - v(k+i)\big) \Big\|_Q^2 + \sum_{i=1}^{p} \Big\| u(k+i-1) \Big\|_R^2 \tag{14}$$

subject to (9). As in (6), weights in Q should be chosen to assign higher priority to the geometrical error rather than the assigned velocity profile.

A controller based on the minimization of (14) would only consider the velocity reference signal q_d'. This means that it would not be able to recover from position errors. However, position errors could occur due to replanning, out-of-path initial configurations, or numerical drifts. In order to recover from such errors, the cost function in (14) is modified by adding a quadratic term for the position error at the successive time instant $k+1$, namely:

$$\Big\| q_d\big(s(k+1)\big) - q(k+1) \Big\|_P^2 \tag{15}$$

where $P \in \mathbb{R}^{n \times n}$ is a positive-definite diagonal matrix.

The inclusion of velocity and acceleration limits in the OCP is straightforward, due to the linearity of the kinematic model (9) in u. Position bounds are not considered in this work, as the path is devised in the joint space and is reasonably assumed to be compliant with the joint position ranges of the robot. Nevertheless, they could be easily included in the formulation, although a further analysis is required to ensure compatibility of the constraints, as detailed in [25].

To take into account the torque limits of the joint, we consider the inverse dynamics (12). The complete OCP can be therefore written as:

$$\begin{aligned}
\underset{u,v}{\text{minimize}} \quad & \sum_{i=1}^{p} \Big\| \big(\dot{q}(k+i) - q_d'(s(k+i))v(k+i)\,,\, 1 - v(k+i)\big) \Big\|_Q^2 \\
& + \sum_{i=1}^{p} \Big\| u(k+i-1) \Big\|_R^2 + \Big\| q_d\big(s(k+1)\big) - q(k+1) \Big\|_P^2 \\
\text{subject to} \quad & x(k+1) = A\,x(k) + B\,u(k) \\
& s(k+1) = s(k) + T v(k) \\
& q = \begin{pmatrix} I_n & 0_{n\times n} \end{pmatrix} x \\
& \dot{q} = \begin{pmatrix} 0_{n\times n} & I_n \end{pmatrix} x \\
& \dot{q}_{\min} \le \dot{q} \le \dot{q}_{\max} \\
& \ddot{q}_{\min} \le u \le \ddot{q}_{\max} \\
& \tau_{\min} \le H(q)u + l(q,\dot{q}) \le \tau_{\max} \\
& 0 \le v \le 1
\end{aligned} \tag{16}$$

for all $i = 1, \dots, p$. Note that v is imposed to be non-negative to avoid motion inversion and to be smaller than or equal to 1 to allow only the slowdown of the original timing law.

Notice that if at time k, $\dot{q}_{\min} \le \dot{q}(k) \le \dot{q}_{\max}$ and $\forall q$ in the robot workspace $\tau_{\max} - l(q,\dot{q}) \ge 0$ and $\tau_{\min} - l(q,\dot{q}) \le 0$, then (16) is always feasible. See [25] for details.

3.3. QP-approximation of the OCP

Problem (16) is nonlinear in the joint command u and v, as H and l nonlinearly depend on q and $\dot{q}$ (which, in turn, depend on u) and q_d' may be a nonlinear function. The problem can be solved by means of sequential quadratic programming or interior point methods but the computation of an optimal solution is typically too slow for real-time implementation, especially for robot with many degrees of freedom. We aim to approximate the problem with a QP in order to speed up the solution and allow real-time computation. First, notice that velocity and acceleration limits are linear in u, thanks to the choice of the acceleration as virtual input in (9). The only nonlinear constraint is the torque one. We use a zero-order linearization of H and l based on the solution obtained at the previous cycle. Denoting with (u^*, v^*) the solution of the OCP at time $k-1$, the approximations of H and l at cycle

k are computed as:

$$H\big(q(k+i)\big) = H\big(\hat{q}(k+i)\big) \tag{17}$$
$$l\big(q(k+i),\dot{q}(k+i)\big) = l\big(\hat{q}(k+i),\hat{\dot{q}}(k+i)\big) \tag{18}$$

for all $i = 1\ldots,p$, where:

$$\hat{q}(k+i) = \begin{pmatrix} I_n & 0_{n\times n}\end{pmatrix}\left(A^i x(k) + A^{i-1}B^i u^*\right) \tag{19}$$
$$\hat{\dot{q}}(k+i) = \begin{pmatrix} 0_{n\times n} & I_n\end{pmatrix}\left(A^i x(k) + A^{i-1}B^i u^*\right) \tag{20}$$

We use the same approach to compute future values of q_d' by using the last prediction of s. Note that at the first iteration, the predictions of q and $\dot{q}$ are initialized to the robot initial state and the prediction of s is the time of the nominal trajectory.

3.4. Complexity reduction of the OCP

The practical implementation of MPC strategies to robotic applications needs a strong reduction of the computational complexity of the algorithm [19, 21]. The approximation described in Section 3.3 converts the nonlinear OCP into a QP with $p(n+1)$ variables and $2p(n+1)$ constraints. However, due to small sampling periods used in robotics, long-enough horizons imply large values of p, which do not make the real-time implementation practical. To reduce the size of the OCP, we adopt a customized complexity-reduction technique based on the staircase parametrization of inputs and outputs. To this purpose, consider that predictive equations can be derived from the state-space model (9) by forward solving the difference equation as described in [23, 26] in the following form:

$$q_{1\to p} = F_{\text{pos}}\, x(k) + G_{\text{pos}}\, u_{1\to p} \tag{21}$$
$$\dot{q}_{1\to p} = F_{\text{vel}}\, x(k) + G_{\text{vel}}\, u_{1\to p} \tag{22}$$

where $q_{1\to p}$, $\dot{q}_{1\to p} \in \mathbb{R}^{np}$, are respectively the prediction of q, $\dot{q}$ at the next p sampling times, $u_{1\to p} \in \mathbb{R}^{np}$ is the input vector along the horizon and $F_{\text{pos}}, F_{\text{vel}} \in \mathbb{R}^{np\times 2p}$ and $G_{\text{pos}}, G_{\text{vel}} \in \mathbb{R}^{np\times np}$, are the free and forced response matrices calculated from (9) as detailed in [24]. By using an Input Blocking (IB) strategy, the size of the optimization vector is reduced. Then, only a subset of prediction time instants $\{\theta_1,\ldots,\theta_h\} \subseteq \{1,\ldots,p\}$, $h < p$, is considered . The resulting approximated predictive equations can be written as:

$$q_\theta = VF_{\text{pos}}\, x(k) + VG_{\text{pos}} R\, u_\Delta \tag{23}$$
$$\dot{q}_\theta = VF_{\text{vel}}\, x(k) + VG_{\text{vel}} R\, u_\Delta \tag{24}$$

where q_θ, $\dot{q}_\theta \in \mathbb{R}^{nh}$ are the prediction of q, $\dot{q}$ at time instants $t_0+\theta_i$, $u_\Delta \in \mathbb{R}^{nv}$, is the parametrized control input, and $V \in \mathbb{R}^{nh\times np}$, $R \in \mathbb{R}^{np\times nh}$ are constant binary matrices that can be computed by following the procedure described in Appendix A. It is worth noticing that (23), (24) can be directly used to compute the predicted outputs in Section 3.3, instead of (19), (20).

4. Results and discussion

In this section, the proposed approach is tested in simulation and experiments. First, a comparison of different receding horizon strategies is presented to demonstrate the validness and the advantages of the proposed approach. Second, the proposed method is implemented online on an experimental platform and compared to a state-of-the-art non-predictive method.

4.1. Comparison of receding horizon strategies

Numerical simulations have been performed on a Universal Robot UR10 manipulator ($n = 6$). The simulations have been performed by using ROS-Gazebo [27]. The low-level controller of the robot consists of a cascade P-PI control architecture with inertia matrix decoupling running at a sampling period equal to 1 ms. The proportional and integral gains of the inner velocity loop are equal to 250 $\frac{\text{rad/s}^2}{\text{rad/s}}$ and 15.6 $\frac{\text{rad/s}^2}{\text{rad}}$ respectively. The proportional gain of the outer position loop is equal to 7 $\frac{\text{rad/s}}{\text{rad}}$. The dynamic model of the manipulator is given by the manufacturer. The robot limits are set to:

$$\dot{q}_{\max} = -\dot{q}_{\min} = (\ 2,\ 2,\ 3,\ 3,\ 3,\ 3\ )\ \text{rad/s}, \tag{25}$$
$$\tau_{\max} = -\tau_{\min} = (\ 200,\ 200,\ 100,\ 50,\ 50,\ 50\ )\text{Nm} \tag{26}$$
$$\ddot{q}_{\max} = -\ddot{q}_{\min} = (\ 5,\ 5,\ 10,\ 10,\ 10,\ 10\ )\ \text{rad/s}^2. \tag{27}$$

The desired trajectory q_d consists of a sinusoidal geometrical path parametrized with respect to the normalized angular length $\gamma \in [0,\ 1]$ such that:

$$q_d = q_{\text{start}} + \Omega \sin(\omega\gamma), \tag{28}$$

Table 1: Tuning parameters of the compared methods.

MPFC	PTS	QP-PTS
Q=diag$(10^8 I_6, 10^5)$	Q=diag$(10^7 I_6, 10^5)$	Q=diag$(10^7 I_6, 10^5)$
R=diag$(0.5 I_6, 10^{-7})$	R=$0.5 I_6$	R=$0.5 I_6$
	P=$2.5 \cdot 10^9 I_6$	P=$10^9 I_6$

Table 2: Selected time instants along the predictive horizon (obtained from (A.2)).

T_p [s]	h	θ/T
0.1	3	1, 26, 100
	5	1, 7, 26, 57, 100
	10	1, 2, 6, 12, 21, 32, 45, 61, 79, 100
0.4	3	1, 101, 400
	5	1, 26, 101, 225, 400
	10	1, 6, 21, 45, 80, 124, 178, 242, 316, 400
1.0	3	1, 251, 1000
	5	1, 63, 251, 563, 1000
	10	1, 13, 50, 112, 198, 309, 445, 605, 709, 1000

where $q_{\text{start}} = (\,0,\,-2,\,0,\,-1.5,\,0,\,0\,)$ rad is the initial joint configuration, $\Omega = (\,1.0,\,0.5,\,1.0,\,0.5,\,2.5,\,1.5\pi\,)$ rad and $\omega = 2\pi$ are respectively the sine amplitude and frequency in radians, and $\gamma\colon\,[0, s_{\text{end}}] \to [0,1]$, $s \mapsto \gamma(s)$ is a polynomial timing law defined as:

$$\gamma(s) = \frac{6}{s_{\text{end}}^5} s^5 - \frac{15}{s_{\text{end}}^4} s^4 + \frac{10}{s_{\text{end}}^3} s^3 \tag{29}$$

where s_{end} is the nominal total time of the desired trajectory. The trajectory can be made more or less demanding by choosing smaller or larger values of s_{end}.

The following algorithms are compared:

- The nonlinear model predictive path following control method (MPFC) [9]. As in [10], the method has been implemented in C++ and ACADO [28], the OCP is solved via a single-shooting single-iteration SQP solver, and the input function is parametrized with an equally spaced step-wise function along the predictive horizon.
- The proposed method solving (16), implemented in C++ and ACADO with a single-shooting single-iteration SQP solver and the aforementioned equally spaced parametrization. This configuration will also be referred to as predictive trajectory scaling (PTS).
- The proposed method solving (16), implemented as above, to which the staircase parametrization described in Section 3.4 is applied. This method will also be called PTS with input blocking (PTS-IB).
- The proposed method to which the QP-approximation described in Section 3.3 is applied. The method is implemented in C++ and the QP algorithm by Goldfarb and Idnani [29] is used. This configuration will also be referred to as QP-PTS (and QP-PTS-IB if input blocking is applied).

The tuning parameters of the algorithms are shown in Table 1. All the methods generate an open-loop feedforward signal that is given as reference to the robot controller. We evaluate the mean and maximum geometrical errors e_{mean} and e_{max} and the average scaling $\dot{s}_{\text{mean}}$. The error is measured as the Euclidean norm of the vector given by the difference between the measured joint configuration and the closest point on the desired path. The average scaling is evaluated in terms of ratio between the original desired time s_{end} and the actual time taken by the scaling algorithm to complete the whole path.

First, we consider a predictive horizon equal to 0.1 s and a number of steps of the OCP discretization equal to 10. Different values of s_{end} are considered. Results are shown in Figure 1. As expected, as s_{end} decreases the desired task becomes more and more demanding, and a heavier scaling is needed to perform the desired path with reduced errors. For this reason, $\dot{s}_{\text{mean}}$ becomes smaller as s_{end} decreases. In all tested cases, MPFC, PTS, and QP-PTS do not show significant differences in terms of performance.

The effect of different lengths and number of steps in the OCP discretization is also evaluated. As an example, we consider the method QP-PTS in the most demanding case $s_{\text{end}} = 7$ s. Results for predictive horizon equal to 0.1 s, 0.4 s and 1 s and number of steps $h = 3, 5, 10$ are shown in Figure 2. The chosen time instants along the horizon are computed through (A.2) and are shown in Table 2. By comparing the proposed and the evenly spaced parametrizations, it is possible to see that the latter leads to poorer results as the horizon increases, as the OCP is discretized more and more roughly at the beginning of the predictive horizon. In particular, the average scaling and the error worsen as the ratio between the horizon length and the number of instants grows. This implies that, when

the evenly spaced parametrization is used, longer predictive horizons may lead to a decay of the control performance. On the contrary, the use of longer horizon usually leads to better performance (being equal the number of optimization variables) when the proposed parametrization is adopted. Similar conclusions can be drawn in case the parametrization with unevenly spaced intervals is applied to PTS and MPFC. However, it is worth pointing out that the use of a nonlinear dynamic model as in MPFC waves the advantages of the linear formulation in terms of computing efficiency and ease of implementation (as a matter of fact, the parametrization technique proposed in Appendix A for linear systems only requires the construction of the binary matrices R and V).

Table 3 shows an analysis of the computing time taken by the different methods for the case $s_{\text{end}} = 7$ s, predictive horizon equal to 0.1 s and number of discretization steps equal to 10. The PTS method is taken as benchtest and the computing times are expressed in percentage with respect to it. Maximum and mean computing times are shown. Results show that the PTS method gives a significant reduction of the computing time (the measured maximum time is one half compared to MPFC) due to the design choices (acceleration as virtual input), being equal the software and hardware platforms. Table 3 also shows computing times for the cases MPFC and PTS when the number of maximum iterations of the SQP is increased up to 10 (MPFC-10iter and PTS-10iter in Table 3). As expected, the computing times grow compared to their single-iteration counterpart, but the advantage given by the PTS approach is even more evident. In this respect, it is worth mentioning that the higher number of maximum iterations did not lead to significant improvements in terms of error and average scaling (less than 2%). Finally, Table 4 shows the computing time taken by the C++ implementation of the QP-PTS-IB method for $h = 3, 5, 10$, showing the suitability of the method in real-time applications.

Table 3: Computing times of the different receding horizon methods (for all methods: predictive horizon $T_p = 0.1$ s and number of prediction time instants $h = 10$). Values are expressed in percentage with respect to the PTS case

Method	$t_{\max}$	t_{mean}
PTS	100%	100%
PTS-IB	100%	100%
PTS-10iter	1540%	1020%
MPFC	200%	160%
MPFC-10iter	3290%	1810%

Table 4: Computing times of QP-PTS method for different values of the number of prediction instants h.

Method	$t_{\max}$ [ms]	t_{mean} [ms]
QP-PTS-IB ($h = 10$)	3.2	0.79
QP-PTS-IB ($h = 5$)	0.61	0.15
QP-PTS-IB ($h = 3$)	0.27	0.06

4.2. Implementation and experiments

Experimental tests were also conducted on a Universal Robot UR10 Version 3.5 ($n = 6$). The robot trajectory is controlled by means of a ROS-based control architecture. Namely, a position controller runs in a ROS Kinetic Ubuntu 16.04. The controller communicates with the robot by means of a TCP connection with sampling period $T = 8$ ms. At each sampling period, the scaling algorithms compute the desired joint positions and velocities. The controller takes the scaling algorithm position output as reference and receives the actual joint position. The controller output is the sum of a proportional action, with gain equal to 7 $\frac{\text{rad/s}}{\text{rad}}$, and a feedforward term equal to the scaling algorithm velocity output. The robot velocity, acceleration, and torque limits have been set as in the previous section.

Owing the problem at hand and the available hardware, both MPFC and PTS execution times exceeded 8 ms and were therefore not suitable for experimental implementation. However, the previous section has shown that the proposed QP-PTS method gives similar results compared to other more computationally demanding strategies, thus significant differences are not expected to show up in the experiments. Given the impossibility of implementing such strategies in real time on the considered 6-degree-of-freedom robot, we compare the QP-PTS method against a state-of-the-art non-predictive technique [16, 17]. This method modifies the velocity profile along the geometrical path, according to the limits of the robot joints, based on the nominal trajectory evaluated at the current time instant. In particular, joint limits are converted into monodimensional constraints on the longitudinal acceleration. If the constraints are compatible, the trajectory can be

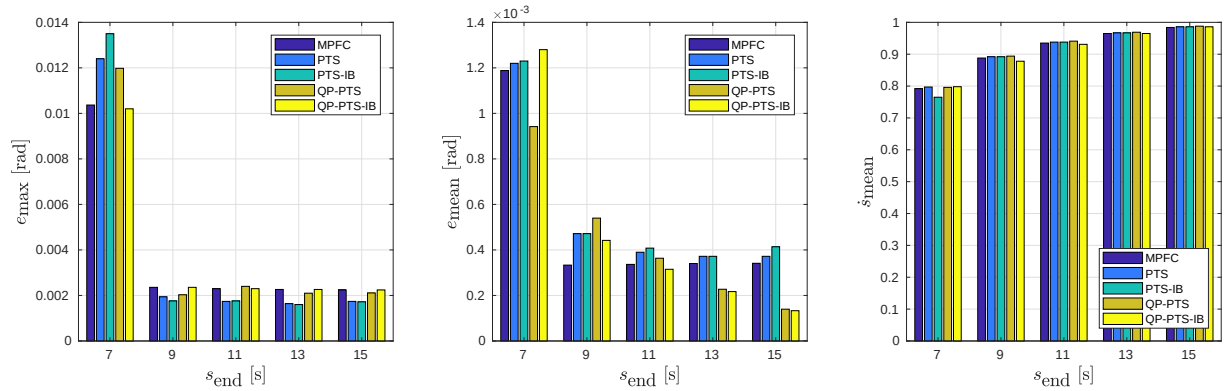


Figure 1: Comparison of receding horizon methods for different values of s_{end} (for all methods: predictive horizon $T_p = 0.1$ s, number of prediction instants $h = 10$). Evaluated variables: maximum and mean path error $e_{\max}$ and e_{mean} and average scaling $\dot{s}_{\text{mean}}$.

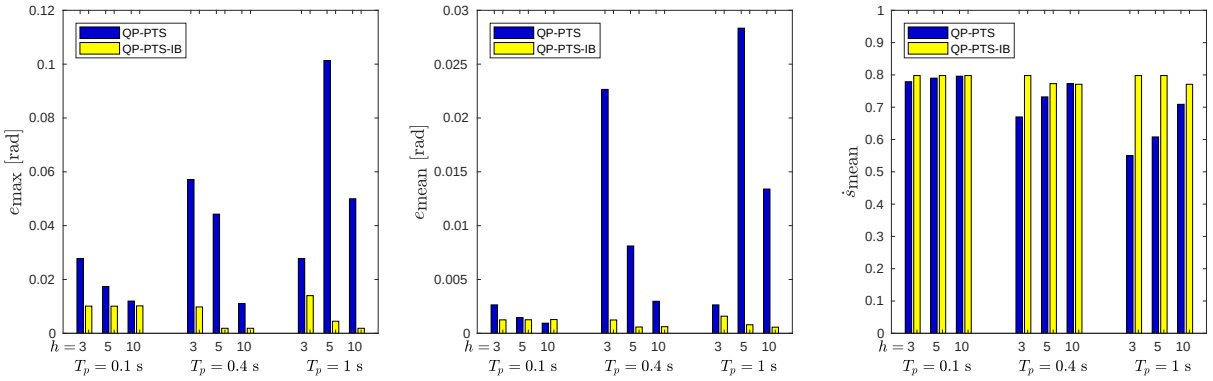


Figure 2: Comparison of QP-PTS and QP-PTS-IB for different predictive horizon T_p and number of prediction instants h. Evaluated variables: maximum and mean path error $e_{\max}$ and e_{mean} and average scaling $\dot{s}_{\text{mean}}$.

scaled without path error, otherwise a deviation arises. In this case, path tracking is recovered in minimum time by means of a nonlinear filter that drives the position and velocity errors to zero.

The QP-PTS-IB method is set with $h = 5$ and predictive horizon equal to $T_p = 50T = 0.4$ s. The selected prediction time instants are computed from (A.2) and are given by $\theta = \{0.008, 0.016, 0.056, 0.160, 0.4\}$ s. Two nominal sinusoidal paths are considered. Referring to (28), Task A has $\Omega_A = (\ 0.3,\ 0.6,\ 0.7,\ 0.65,\ 0.75,\ 0.8\ )$ rad, $\omega_A = 2\pi$, and Task B has $\Omega_B = -\Omega_A$, $\omega_B = 3\pi$. Function γ is a seven-trait timing law in s, where s_{end} is equal to 3.5 s for Task A and 4 s for Task B.

Figure 3 shows the measured joint positions and torques for Task A. The dashed gray line represents the nominal trajectory without scaling (the nominal torques in Figure 3b have been computed from the nominal trajectory by means of the dynamic model of the robot). As expected, the scaling methods generate a dilation of the nominal timing law in order to prevent saturation of the joint actuators. However, according to the trend of γ shown in Figure 5a, Figure 3a shows that the joint position trend given by the predictive method is significantly closer to the nominal trajectory, while the non-predictive method gives a heavier scaling with consequent significant deformation of the timing law. In particular, the total time taken by the predictive method to reach the final state is equal to 3.57 s, while the time taken by the non-predictive method results to be equal to 4.12 s. Moreover, in Figure 3, it is possible to see that both the scaling methods keep the joint torques within the bounds (except for some spikes reasonably due to the robot controller and model mismatches). It is worth noticing that the predictive method gives a much smoother trend of the joint variables compared to the non-predictive case, and this results in reduced stress of the actuators.

The values of the measured maximum and mean path errors are shown in Table 5a. It is worth noticing that the errors are comparable, and this is mainly due to the tracking error introduced by the robot controller. However, in order to obtain similar results in terms of path-following errors, it is necessary for the non-predictive method to adopt heuristic strategies to prevent joint saturations, as detailed in [30]. On the other hand, such strategies necessarily give conservative solutions in terms of scaling, as shown by the comparison of the two methods.

Table 5: Experimental comparison of the non-predictive and the proposed predictive methods.

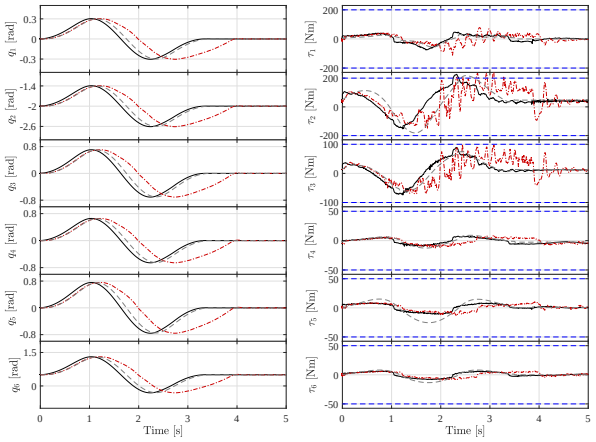
(a) Task A

	$e_{\max}$ [rad]	e_{mean} [rad]	$\dot{s}_{\text{mean}}$
QP-PTS-IB	$1.41 \cdot 10^{-2}$	$5.20 \cdot 10^{-4}$	0.98
Non-predictive	$1.91 \cdot 10^{-2}$	$2.01 \cdot 10^{-2}$	0.85

(b) Task B

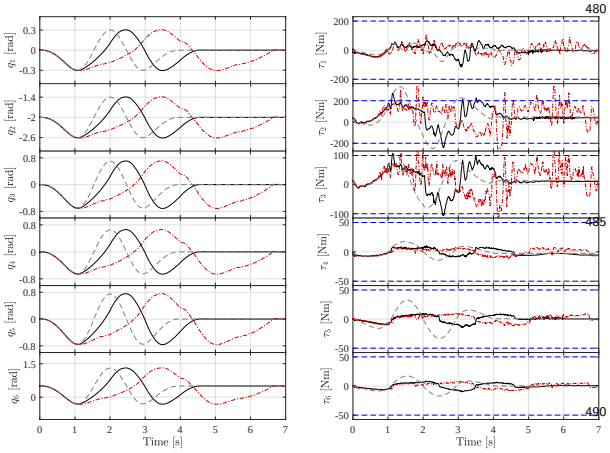
	$e_{\max}$ [rad]	e_{mean} [rad]	$\dot{s}_{\text{mean}}$
QP-PTS-IB	$1.91 \cdot 10^{-3}$	$8.53 \cdot 10^{-4}$	0.83
Non-predictive	$1.06 \cdot 10^{-2}$	$1.24 \cdot 10^{-3}$	0.59

Similar conclusions can be drawn for Task B. Figure 4 shows the comparison between the joint positions and torques obtained by the different methods and the measured errors are shown in Table 5b. In this case, significant saturations of the second joint are expected (dashed gray line in Figure 4b). Both methods operate in such a way that torque saturations are avoided, but the predictive method gives a better scaling of the original trajectory, as also shown by the trend of γ in Figure 5b. In this way, the scaled trajectory results to be much more similar to the nominal one and this is also proven by the fact that the time taken to complete the path is equal to 4.82 s for the predictive method and 6.78 s for the non-predictive one. Moreover, joint variables given by the predictive method are significantly smoother.



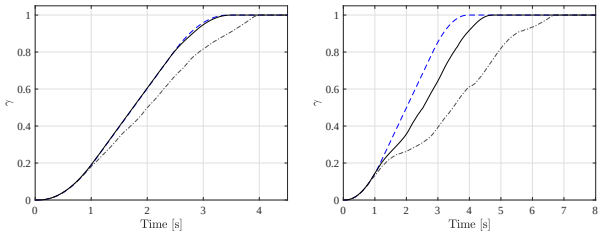
(a) Joint positions (b) Joint torques

Figure 3: Comparison of measured joint positions and torques for Task A. Dashed gray line: nominal reference. Solid black line: QP-PTS-IB method. Dash-dot red line: non-predictive scaling method. Dashed blue line: torque limits.



(a) Joint positions (b) Joint torques

Figure 4: Comparison of measured joint positions and torques for Task B. Dashed gray line: nominal reference. Solid black line: QP-PTS-IB method. Dash-dot red line: non-predictive scaling method. Dashed blue line: torque limits.



(a) Task A (b) Task B

Figure 5: Comparison of the resulting timing laws for tasks A and B. Dashed blue line: nominal timing law. Solid black line: timing law obtained by QP-PTS. Dash-dot gray line: timing law obtained by the non-predictive method.

5. Conclusions

This paper has proposed a new predictive trajectory scaling technique for robot manipulators under velocity, acceleration, and torque limits. The technique is able to online modify the desired timing law of the trajectory, in order to preserve the geometrical path if the desired motion is infeasible with respect to the robot limits. Compared to other receding horizon techniques, the proposed approach gives a dramatic reduction of the computational complexity, allowing the real-time implementation on complex systems without decays from the performance viewpoint, as shown by the numerical tests. Moreover, experiments show that the proposed method outperforms state-of-the-art non-predictive technique.

Appendix A. Implementation of the complexity-reduction technique

As mentioned in Section 3.1, the size of the OCP is reduced to decrease the required computing time of the algorithm. An IB technique reduces the number of control variables [24, 31]. and the number of prediction time-instants is reduced as well by choosing only a small number of nodes along the predictive horizon.

Appendix A.1. Reducing the number of optimization variables

The basic idea of IB is that the control input is considered to be constant along several sampling periods

in the optimization problem. In other words, the control input is parametrized as a staircase function along the control horizon, and the steps of the parametrization occur at multiples of the sampling period. To implement such technique, consider an auxiliary input vector $u_\Delta := (u_{\Delta 1}, \dots, u_{\Delta h}) \in \mathbb{R}^{nh}$, with $h < p$, such that:

$$u_{1\to p} = R\, u_\Delta \tag{A.1}$$

where $R \in \mathbb{R}^{np\times nh}$ is the so-called *blocking matrix*. It is a Boolean matrix that parametrizes the optimization variable $u_{1\to p}$ by means of a new smaller vector u_Δ. Now, define a blocking vector $\Delta = (\Delta_1, \dots, \Delta_h)$, whose elements represents the number of sampling periods along which each control input is forced to be constant. A simple procedure to compute the blocking matrix R given the blocking vector Δ is shown in Algorithm 1.

For the sake of clarity, consider a SISO system and a control horizon $p = 5$. Imposing $u(k+2) = u(k+3), u(k+4) = u(k+5)$ gives $\Delta = [1, 2, 2]$. Therefore, R results to be:

$$R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \in \mathbb{R}^{np\times nh}$$

Algorithm 1 Computation of the Blocking Matrix R

Input: $n,\ \Delta$
Output: R
  $i_{\text{row}} = 1$, $h = \text{length}(\Delta)$, $p = \sum_i \Delta(i)$
  $R = 0_{np\times nh}$
  for $i = 1$ to v **do**
    columns $= [n(i-1)+1, \dots, ni]$
    for $j = 1$ to $\Delta(i)$ **do**
      rows $= [i_{\text{row}}, \dots, i_{\text{row}} + n - 1]$
      $R[\text{rows; columns}] = I_n$
      $i_{\text{row}} = i_{\text{row}} + n$
    end for
  end for

Algorithm 2 Computation of the selection matrix V

Input: $n,\ p,\ \theta$
Output: V
  $h = \text{length}(\theta)$
  $V = 0_{nh\times np}$
  for $i = 1$ to h **do**
    rows $= [n(i-1)+1, \dots, ni]$
    columns $= [n(\theta(i)-1)+1, \dots, n\theta(i)]$
    $V[\text{rows; columns}] = I_n$
  end for

Appendix A.2. Reducing the number of prediction time instants

The computational burden of the OCP also depends on the number of prediction time instants. To reduce it, only a few nodes along the horizon are considered in the optimization. Given the predictive matrices in (21), each ith block of n rows corresponds to the prediction of the n outputs at time instant $k+i$. Now consider an ordered subset $\theta \subseteq \{1, \dots, p\}$, with cardinality equal to $h < p$ (*i.e.*, a subset of h predictive time instants). Aiming to select the output prediction only at times $k+i,\ i \in \theta$, it is possible to introduce a selection matrix $V \in \mathbb{R}^{np\times nh}$, which selects the rows corresponding to such time instants.

Matrix V is a Boolean matrix that can be easily computed as illustrated in Algorithm 2. As an illustrative example, consider a SISO system and a predictive horizon $p = 5$. To consider only the 1st, 3rd, and 5th time instants in the prediction, it results $\theta = \{1, 3, 5\}$ and $h = 3$, which gives:

$$V = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \in \mathbb{R}^{nh\times np}$$

Combining both the blocking matrix R and the selection matrix V, the new predictive equations (23), (24) result.

Appendix A.3. Choice of R and V

It is necessary to choose Δ and θ such that the reduced predicted equations (23) and (24) give a good approximation of the original equations (21) (22). Note that Δ and θ are chosen *a priori*, so they do not change during the execution of the algorithm (and so do the matrices R and V).

An optimal choice cannot be obtained, since the reference trajectories are non-repetitive in general. Based on the consideration that the parametrization should give more importance to the initial part of the horizon rather

than to the final one and that the first control step should coincide with the first sampling period, θ and Δ are chosen via the following parabolic equation:

$$\theta_i = \frac{p-1}{(h-1)^2} i^2 - 2\frac{p-1}{(h-1)^2} i + \left(\frac{p-1}{(h-1)^2} + 1 \right) \quad \text{(A.2)}$$
$$\Delta = \left[1, \theta_2 - \theta_1, \ldots, \theta_h - \theta_{h-1}\right]$$

for $i = 1, \ldots, h$ and rounding each element to the nearest integer.

Acknowledgment

The research leading to these results was partially funded by the European Union H2020-ICT-2017-1 Pickplace: Flexible, safe and dependable robotic part handling in industrial environment (grant agreement: 780488).

References

[1] D. Verscheure, B. Demeulenaere, J. Swevers, J. De Schutter, M. Diehl, Time-optimal path tracking for robots: A convex optimization approach, IEEE Transactions on Automatic Control 54 (2009) 2318–2327.

[2] D. Costantinescu, E. Croft, Smooth and time-optimal trajectory planning for industrial manipulators along specified paths, Journal of Robotic Systems 17 (2000) 233–249.

[3] A. Casalino, A. M. Zanchettin, P. Rocco, Online planning of optimal trajectories on assigned paths with dynamic constraints for robot manipulators, in: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Daejeon (Korea), 2016, pp. 979–985.

[4] J. E. Bobrow, S. Dubowsky, J. S. Gibson, Time-optimal control of robotic manipulators along specified paths, The International Journal of Robotics Research 4 (1985) 3–17.

[5] H. Pham, Q. Pham, A new approach to time-optimal path parameterization based on reachability analysis, IEEE Transactions on Robotics 34 (2018) 645–659.

[6] D. Lam, C. Manzie, M. C. Good, Model predictive contouring control for biaxial systems, IEEE Transactions on Control Systems Technology 21 (2013) 552–559.

[7] M. Böck, A. Kugi, Real-time nonlinear model predictive path-following control of a laboratory tower crane, IEEE Transactions on Control Systems Technology 22 (2014) 1461–1473.

[8] N. van Duijkeren, R. Verschueren, G. Pipeleers, M. Diehl, J. Swevers, Path-following NMPC for serial-link robot manipulators using a path-parametric system reformulation, in: Proceedings of the European Control Conference, Aalborg (Denmark), 2016, pp. 477–482.

[9] T. Faulwasser, R. Findeisen, Nonlinear model predictive control for constrained output path following, IEEE Transactions on Automatic Control 61 (2016) 1026–1039.

[10] T. Faulwasser, T. Weber, P. Zometa, R. Findeisen, Implementation of nonlinear model predictive path-following control for an industrial robot, IEEE Transactions on Control Systems Technology 25 (2017) 1505–1511.

[11] 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 Transactions on Robotics and Automation 19 (2003) 162–167.

[12] F. Flacco, A. De Luca, O. Khatib, Control of redundant robots under hard joint constraints: Saturation in the null space, IEEE Transactions on Robotics 31 (2015) 637–654.

[13] O. Dahl, L. Nielsen, Torque-limited path following by online trajectory time scaling, IEEE Transactions on Robotics and Automation 6 (1990) 554–561.

[14] O. Dahl, Path-constrained robot control with limited torques-experimental evaluation, IEEE Transactions on Robotics and Automation 10 (1994) 658–669.

[15] H. Arai, K. Tanie, S. Tachi, Path tracking control of a manipulator considering torque saturation, IEEE

Transactions on Industrial Electronics 41 (1994) 25–31.

[16] C. Guarino Lo Bianco, O. Gerelli, Online trajectory scaling for manipulators subject to high-order kinematic and dynamic constraints, IEEE Transactions on Robotics 27 (2011) 1144–1152.

[17] 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 Transactions on Industrial Electronics 61 (2014) 4115–4125.

[18] M. Faroni, M. Beschi, N. Pedrocchi, A. Visioli, Predictive inverse kinematics for redundant manipulators with task scaling and kinematic constraints, IEEE Transactions on Robotics 35 (2019) 278–285.

[19] M. Faroni, M. Beschi, A. Visioli, L. Molinari Tosatti, A predictive approach to redundancy resolution for robot manipulators, in: Proceedings of the IFAC World Congress, Toulouse (France), 2017, pp. 8975–8980.

[20] L. Tamas, I. Nascu, R. De Keyser, The NEPSAC nonlinear predictive controller in a real life experiment, in: Proceedings of the IEEE International Conference on Intelligent Engineering Systems, Budapest (Hungary), 2007, pp. 229–234.

[21] D. Dimitrov, P.-B. Wieber, H. J. Ferreau, M. Diehl, On the implementation of model predictive control for on-line walking pattern generation, in: Proceedings of the IEEE International Conference on Robotics and Automation, Pasadena (USA), 2008, pp. 2685–2690.

[22] L. Wang, Discrete model predictive controller design using Laguerre functions, Journal of Process Control 14 (2004) 131–142.

[23] L. Wang, Model predictive control system design and implementation using MATLAB®, Springer Science & Business Media, 2009.

[24] M. Faroni, M. Beschi, M. Berenguel, A. Visioli, Fast MPC with staircase parametrization of the inputs: Continuous Input Blocking, in: Proceedings of the IEEE International Conference on Emerging Technologies and Factory Automation, Limassol (Cyprus), 2017, pp. 1–8.

[25] M. Faroni, M. Beschi, N. Pedrocchi, A. Visioli, Viability and feasibility of constrained kinematic control of manipulators, Robotics 7 (3) (2018) 1–19.

[26] J. A. Rossiter, Model-based predictive control: a practical approach, CRC press, 2003.

[27] N. Koenig, A. Howard, Design and use paradigms for Gazebo, an open-source multi-robot simulator, in: Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, Sendai (Japan), 2004, pp. 2149–2154.

[28] B. Houska, H. Ferreau, M. Diehl, ACADO Toolkit – An Open Source Framework for Automatic Control and Dynamic Optimization, Optimal Control Applications and Methods 32 (2011) 298–312.

[29] D. Goldfarb, A. Idnani, A numerically stable dual method for solving strictly convex quadratic programs, Mathematical programming 27 (1983) 1–33.

[30] 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, Karlsruhe (Germany), 2013, pp. 870–876.

[31] R. Cagienard, P. Grieder, E. C. Kerrigan, M. Morari, Move blocking strategies in receding horizon control, Journal of Process Control 17 (2007) 563–570.