Investigating the Performances of Control Parameterizations for Nonlinear Model Predictive Control

Franco Fusco, Guillaume Allibert, Olivier Kermorgant, Philippe Martinet

Investigating the Performances of Control Parameterizations for Nonlinear Model Predictive Control

Franco Fusco[1], Guillaume Allibert[1], Olivier Kermorgant[2] and Philippe Martinet[3]

***Abstract*— Solving Direct Shooting Model Predictive Control (MPC) optimization problems online can be computationally expensive if a large horizon is used while also maintaining a dense time sampling. In these cases, it is accepted that trade-offs between computational load and performances should be sought in order to meet real-time feasibility requirements. However, making the problem more tractable for the hardware should not necessarily imply a decrease in performances. One technique that has been proposed in the literature makes use of control input parameterizations to decrease the numerical complexity of nonlinear MPC problems without necessarily affecting the performances significantly. In this paper, we review the use of parameterizations and propose a simple Sequential Quadratic Programming algorithm for nonlinear MPC. We then benchmark the performances of the solver in simulation, showing that parameterizations allow to attain good performances with (significantly) lower computation times than state-of-the-art solvers.**

I. Introduction

Model Predictive Control (MPC) is probably one of the most attractive control design methodologies, as illustrated in the state of the art made by Mayne [1]. The ability of MPC to handle nonlinearities, enforce constraints and provide performance/cost trade-offs are just some of the reasons for its popularity. The MPC strategy consists in computing, at each control iteration, an optimal sequence of future commands that minimizes a given cost function. Feedback is ensured since the cost explicitly depends not only on the control sequence, but also on the current measured (or estimated) states. Only the first control is applied to the system, and the whole process is repeated at the next iteration, indefinitely. Since the control problem is written in the form of a cost minimization, constraints on both the control and the state may explicitly be considered.

Historically, MPC was developed for systems with slow dynamics and required long computation times. Nonetheless, in the last two decades a lot of work has been done to deploy predictive control also on fast systems. Some of these pose additional restrictions and challenges, as the optimization problem must be solved on embedded hardware with limited computing power. One way to improve the efficiency of MPC schemes is to reduce the complexity of the optimization problem, and several methods have been proposed in order to speed up the solving process. A first one consists in approximating the original nonlinear constrained problem to find a solution as quickly as possible, even if it might be a sub-optimal one. Indeed, even in case of a coarse approximation of the optimization, global performances of the closed loop controller can be satisfactory, with computation times that are sufficiently small to allow real time implementations [2], [3]. An alternative approach proposed in the literature is to carefully exploit the particular structure of a problem to improve the efficiency of a solver [4]. Solutions based on offline computations have also been proposed. They explore the state space offline and find an explicit solution to the problem. On-line computations are then reduced to the evaluation of a piece-wise linear function [5].

Control parameterizations can also be an effective solution [6], [7], [8], [9]. The objective is to drastically reduce the number of control variables in the optimization problem while trying to keep the performance loss as small as possible. One of the major advantages of using a parameterization is the ability to decouple the prediction horizon from the control one. This is a very important improvement since for many systems the use of large prediction horizon is mandatory to ensure system stability in closed loop [1]. However, in the classical approach, the two horizons are closely linked and increasing the prediction one necessarily implies enlarging the number of control variables. The complexity of the optimization problem, therefore, increases considerably. Even disregarding the issue of system instabilities, it is often necessary to select a large prediction horizon so as to obtain good closed-loop behavior. This is especially true when dealing with small sampling periods.

In this paper, we investigate the use of different input parameterizations in MPC to control non-linear systems. In particular, we review few existing input parameterizations coupled with a simple single-shooting scheme that can solve the MPC optimization using Sequential Quadratic Programming (SQP) quickly and while enforcing control input constraints such as bounds on the maximal command and its rate. Thanks to the parameterization, the sub-problems arising at each SQP iteration are low-dimensional and can thus be solved extremely quickly. Indeed, we show that using the parameterized approach it is possible to attain sub-millisecond performances despite a very dense time-sampling and outperform existing state-of-the-art algorithms both on a laptop and on a Raspberry Pi device.

The remainder of this paper is organized as follows. In the next section, we firstly detail the adopted nonlinear MPC formulation and continue by recalling the studied

[1]Université Côte d'Azur, CNRS, I3S, France name.surname@i3s.unice.fr

[2]Centrale Nantes, Laboratoire des Sciences du Numérique de Nantes LS2N, Nantes, France olivier.kermorgant@ls2n.fr

[3]Université Côte d'Azur, INRIA Sophia-Antipolis, France, philippe.martinet@inria.fr

control parameterizations. We propose a simple solver for the parameterized MPC optimization problem in section II-C and benchmark its performances in section III. In particular, we analyze how they change in relation to the chosen number of parameters and compare against state of the art solvers from an existing framework. Finally, the last Section reports our conclusions and proposes future perspectives.

II. Nonlinear Model Predictive Control

In this section, we recall the formulation of Direct Shooting methods for constrained Nonlinear MPC, which is at the basis of the employed solver. We then discuss how the introduction of a parameterization can help in reducing the complexity of the described problem and recall some existing parameterizations. We finally propose an algorithm to compute optimal parameters, which is a SQP strategy based on the Gauss-Newton Hessian approximation.

A. Direct Shooting Nonlinear MPC

The problem that Model Predictive Control tries to solve can be stated informally as follows: given an evolution model of the controlled system, find an optimal control trajectory which can steer the current state of a system towards a desired configuration, while meeting feasibility requirements such as actuation limits.

A variety of mathematical formulations can be found in the literature [10], [11], which can be classified in three main categories: (1) indirect methods, (2) dynamic programming and (3) direct approaches. Indirect methods formulate the control problem as a functional optimization that is solved using variational calculus, while dynamic programming formulations exploit Bellman's principle of optimality to obtain a control sequence via a recursive procedure. The solutions to these problems are hard to find in the general case, or can become computationally expensive due to the so-called "*curse of dimensionality*" [12].

Direct methods work instead by transcribing the problem as a finite-dimensional nonlinear optimization. First of all, the controlled system is modeled as a discrete-time one, characterized by a transition function $\boldsymbol{f}$ that, given the state $\boldsymbol{x}_k \in \mathbb{R}^n$ and control $\boldsymbol{u}_k \in \mathbb{R}^m$ at the discrete time step k, returns the corresponding state that should be reached by the system at the next time step $k+1$, *i.e.*,

$$\boldsymbol{x}_{k+1} = \boldsymbol{f}\left(\boldsymbol{x}_k, \boldsymbol{u}_k\right) \tag{1}$$

The "informal" optimal control problem stated in the beginning of this section is then commonly written as the following nonlinear optimization:

$$\min \sum_{k=1}^{N_p} \|\boldsymbol{x}_k - \boldsymbol{x}_k^\star\|^2_{\mathbf{Q}_k} + \sum_{k=0}^{N_p-1} \|\boldsymbol{u}_k - \boldsymbol{u}_k^\star\|^2_{\mathbf{R}_k} \tag{2a}$$

subject to:

$$\boldsymbol{x}_{k+1} - \boldsymbol{f}\left(\boldsymbol{x}_k, \boldsymbol{u}_k\right) = 0 \quad \forall k = 0, \cdots, N_p - 1 \tag{2b}$$

$$\boldsymbol{c}_x\left(\boldsymbol{x}_k\right) \leq 0 \quad \forall k = 1, \cdots, N_p \tag{2c}$$

$$\boldsymbol{c}_u\left(\boldsymbol{u}_k\right) \leq 0 \quad \forall k = 0, \cdots, N_p - 1 \tag{2d}$$

wherein N_p is known as *prediction horizon*, while $\boldsymbol{x}_k^\star$ and $\boldsymbol{u}_k^\star$ represent the desired state and control inputs at the discrete time step k. $\|\boldsymbol{v}\|_{\mathbf{A}}$ is used to denote the norm of vector $\boldsymbol{v}$ weighted by the positive semidefinite matrix $\mathbf{A}$, such that $\|\boldsymbol{v}\|^2_{\mathbf{A}} = \boldsymbol{v}^T\mathbf{A}\boldsymbol{v}$. The constraints (2b) are introduced to ensure that the sequence of states $\underline{\boldsymbol{x}} = \boldsymbol{x}_1, \cdots, \boldsymbol{x}_{N_p}$ is coherent with the sequence of control inputs $\underline{\boldsymbol{u}} = \boldsymbol{u}_0, \cdots, \boldsymbol{u}_{N_p-1}$ according to the discrete time evolution model of the system. Constraints (2c) and (2d) can optionally be added to enforce explicitly, by the functions $\boldsymbol{c}_x$ and $\boldsymbol{c}_u$ respectively, feasibility of the states and controls contained in $\underline{\boldsymbol{x}}$ and $\underline{\boldsymbol{u}}$. Finally, it must be noted that in the problem above the value $\boldsymbol{x}_0$ is assumed to be a known constant, corresponding to the actual state of the system when performing the optimization.

Model constraints (2b) can be enforced in two ways. The first one is to use both $\underline{\boldsymbol{u}}$ and $\underline{\boldsymbol{x}}$ as decision variables and to explicitly deal with these constraints during the optimization. This leads to a sub-family of direct methods known as *multiple-shooting* techniques [13], [14]. The second option consists in using a *single-shooting* method, in which only the control input sequence $\underline{\boldsymbol{u}}$ is used as free variables for the search. Predicted future states can then be evaluated iteratively starting from $\boldsymbol{x}_0$ and the selected control inputs. Model constraints are thus implicitly dealt with by this evaluation process. In this paper, we have focused our attention on single-shooting strategies and therefore (2b) are assumed to be always satisfied during the optimization.

To obtain a control sequence given the current state $\boldsymbol{x}_0$, a fairly popular choice is to solve the optimization using Sequential Quadratic Programming with Gauss-Newton approximations [15] of the Hessian of the objective in (2a). This iterative solution technique works by firstly approximating the original problem around the current guess of $\underline{\boldsymbol{u}}$ with a linearly constrained quadratic minimization. Solving the quadratic sub-problem allows to obtain a variation $\boldsymbol{\delta}_u$ for the control input sequence, and a line search is then performed to form a new guess $\underline{\boldsymbol{u}}'$ for the original nonlinear optimization. The new solution takes the form $\underline{\boldsymbol{u}}' = \underline{\boldsymbol{u}} + s\boldsymbol{\delta}_u$, with $s \in [0, 1]$ chosen so that it causes a sufficient decrease in a given merit function that penalizes constraints violation and favors decreases in the original objective [16], [17]. The whole process (solution of the sub-problem and evaluation of a new candidate) is repeated until termination criteria are met – typically, until the step size is small enough or a maximum period of time has elapsed.

B. Complexity Reduction via Parameterization

In the optimization scheme presented above, performances highly depend on the prediction horizon. Ideally, one would wish to sample the system at a high frequency to reduce the inaccuracies introduced by discretization. At the same time, predicting over a longer period of time should lead to better results since it enables the algorithm to intelligently consider maneuvers that are globally optimal and steer the system to the desired state. It is thereby beneficial to increase the value of N_p in both cases. However, the number of decision

variables (and therefore the complexity of the nonlinear optimization) grows with the prediction horizon as well.

Since the optimization has to be performed at each iteration, it is fundamental to be able to converge to a solution quickly enough to meet real-time feasibility. This is particularly important when the controlled system is characterized by a fast dynamics, presents instability or features limited computational power. In order to reduce the complexity of the problem to be solved, different strategies can be employed that aims at decreasing the number of decision variables.

One common practice consists in allowing only the first $N_c \leq N_p$ control samples to be freely allocated in the optimization [18], [19]. The remaining $N_p - N_c$ samples are instead set to the same value of $\boldsymbol{u}_{N_c-1}$, as illustrated in fig. 1(a). This allows the number of decision variables to be substantially reduced, while still being able to foresee a sufficiently large amount of future samples. Nonetheless, concentrating all the "degrees of freedom" of the optimization in the beginning of the prediction interval can lead to sub-optimal performances. Since the last value of the control sequence is repeated a large number of times, its influence on the final state reached by the system is intuitively much larger than that of the first samples. Furthermore, if the system presents instabilities, constant signals can lead to significant displacements. The control action in the beginning of the prediction horizon might thus be required to preemptively compensate for such large displacement, possibly leading to unnecessarily large motions and less regular control actions.

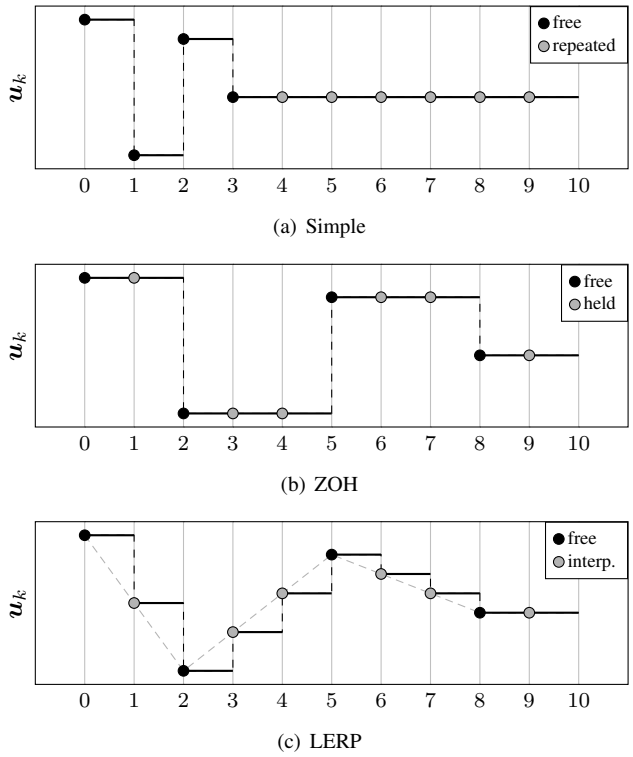
For the reasons above, it would be desirable to distribute the degrees of freedom of the optimization along the prediction horizon. One possible way to do so could consist in selecting, as "free samples", values from $\underline{\boldsymbol{u}}$ that are not all at the beginning of the prediction horizon, but rather uniformly spread along it. Remaining values could instead be obtained by "holding" the free samples constant until the next one is reached (as shown in fig. 1(b)) or alternatively using linear interpolation (fig. 1(c)). These strategies are known in the literature as *Move Blocking* [20].

A further, more general, option can be considered to reduce the number of optimization variables, which relies on the choice of a set of N_b basis functions $\phi_i : \mathbb{R}^+ \to \mathbb{R}$ $(i = 1, \cdots, N_b)$. The control sequence can then be generated from these functions by linear combination as:

$$\boldsymbol{u}_k = \sum_{i=1}^{N_b} \phi_i(k\Delta t)\boldsymbol{\eta}_i \tag{3}$$

where the combination coefficients $\boldsymbol{\eta}_i$ are to be determined by the optimization and Δt is the sampling time used in the prediction model. Some examples in this sense are damped Laguerre polynomials [8] and Haar wavelets [21]. In this paper, we propose to use a set of exponentially damped polynomials in the form:

$$\phi_i(t) = \left(\frac{et}{i\tau}\right)^i e^{-t/\tau} \tag{4}$$



(a) Simple

(b) ZOH

(c) LERP

Fig. 1. Example of control sequences $\underline{\boldsymbol{u}}$ with a reduced number of degrees of freedom. Even if the prediction horizon is $N_p = 10$, only 4 variables are freely assigned, while the remaining ones are defined in function of the free samples.

All functions from this basis have a unique maximum in $\mathbb{R}^+$ located at $t = i\tau$, such that $\phi_i(i\tau) = 1$. It is thus reasonably easy to tune the shape constant τ, which simply controls the locations of the maxima. To the best of our knowledge, this is the first time such basis is employed in a control parameterization.

The interest in using a basis of functions is that the overall control sequence can "inherit" some desirable properties from its generating functions. As an example, if all ϕ_i are smooth (such as in (4)) then the generated sequences naturally feature reduced variations between consecutive samples. In addition, changes in a single combination parameter usually affects a whole portion of the control sequence, rather than a specific one. From a practical point of view, this implies that even with a reduced number of variables it is possible to generate a rather wide variety of control profiles.

All the strategies described so far can be unified under the same concept of *control parameterization*, which is a mapping from a given vector of parameters $\boldsymbol{\eta} \in \mathbb{R}^{N_\eta}$ to the control sequence $\underline{\boldsymbol{u}}$. Furthermore, all the parameterizations discussed above can be shown to be linear in $\boldsymbol{\eta}$, *i.e.*,

$$\underline{\boldsymbol{u}} = \begin{bmatrix} \boldsymbol{u}_0 \\ \cdots \\ \boldsymbol{u}_{N_p-1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\Pi}_0 \\ \cdots \\ \boldsymbol{\Pi}_{N_p-1} \end{bmatrix} \boldsymbol{\eta} = \boldsymbol{\Pi}\boldsymbol{\eta} \tag{5}$$

In particular, it can be shown that for the first three parameterizations, which will be referred to as "Simple", "ZOH" (*Zero-Order-Holder*) and "LERP" (*linear interpolation*), the parameters $\boldsymbol{\eta}$ correspond to the values of the free control

samples (see fig. 1) and that their combination matrices $\mathbf{\Pi}$ have a special banded form. On the contrary, function basis parameterizations feature dense combination matrices:

$$\mathbf{\Pi} = \begin{bmatrix} \phi_1(0) & \cdots & \phi_{N_b}(0) \\ \phi_1(\Delta t) & \cdots & \phi_{N_b}(\Delta t) \\ \vdots & \ddots & \vdots \\ \phi_1(\Delta t(N_p-1)) & \cdots & \phi_{N_b}(\Delta t(N_p-1)) \end{bmatrix} \tag{6}$$

with the parameters $\boldsymbol{\eta}$ being the stack of vectors $\boldsymbol{\eta}_i$ from (3).

C. SQP Solver for Parameterized MPC

In this section we detail a Sequential Quadratic Programming solver that can be employed for the solution of the MPC optimization problem (2) when using parameterizations. The proposed algorithm works by formulating a quadratic sub-problem, whose solution allows to update the current guess of the parameters $\boldsymbol{\eta}$. These two steps (solution of the sub-problem and update of the parameters) is repeated until termination conditions are met. As mentioned before, we consider in this paper a single-shooting strategy in which (2b) is directly satisfied, while concerning other constraints we pose limits on the magnitude of the control actions and its variation, *i.e.*,

$$\boldsymbol{u}_{min} \leq \boldsymbol{u}_k \leq \boldsymbol{u}_{max} \quad \forall k = 0, \cdots, N_p - 1 \tag{7a}$$
$$-\Delta\boldsymbol{u} \leq \boldsymbol{u}_k - \boldsymbol{u}_{k-1} \leq \Delta\boldsymbol{u} \quad \forall k = 0, \cdots, N_p - 1 \tag{7b}$$

Concerning the definition of the quadratic sub-problem, we firstly consider a current guess $\boldsymbol{\eta}$ of the parameters, from which the control sequence $\underline{\boldsymbol{u}}$ is computed using (5). Afterwards, predicted states $\underline{\boldsymbol{x}}$ are evaluated recursively as function of the initial state $\boldsymbol{x}_0$ and $\underline{\boldsymbol{u}}$ using the discrete-time model of the system. States and controls in the objective are then linearized around $\boldsymbol{\eta}+\boldsymbol{\delta}_\eta$, with $\boldsymbol{\delta}_\eta$ representing variation of the current parameters. Injecting the linearized states into the original objective (2a) leads to

$$\min \frac{1}{2}\boldsymbol{\delta}_\eta^T \mathbf{H} \boldsymbol{\delta}_\eta + \boldsymbol{g}^T \boldsymbol{\delta}_\eta \tag{8a}$$

with

$$\mathbf{H} = \sum_{k=1}^{N_p} \frac{\partial \boldsymbol{x}_k}{\partial \boldsymbol{\eta}}^T \mathbf{Q}_k \frac{\partial \boldsymbol{x}_k}{\partial \boldsymbol{\eta}} + \sum_{k=0}^{N_p-1} \mathbf{\Pi}_k^T \mathbf{R}_k \mathbf{\Pi}_k \tag{8b}$$

$$\boldsymbol{g}^T = \sum_{k=1}^{N_p} (\boldsymbol{x}_k - \boldsymbol{x}_k^\star)^T \mathbf{Q}_k \frac{\partial \boldsymbol{x}_k}{\partial \boldsymbol{\eta}} + \sum_{k=0}^{N_p-1} (\boldsymbol{u}_k - \boldsymbol{u}_k^\star)^T \mathbf{R}_k \mathbf{\Pi}_k \tag{8c}$$

Note that $\mathbf{H}$ corresponds to a Newton-approximation of the Hessian of the original objective function. In the relations above, the Jacobians of the states with respect to the parameters can be evaluated using the recursive relation

$$\frac{\partial \boldsymbol{x}_{k+1}}{\partial \boldsymbol{\eta}} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} \frac{\partial \boldsymbol{x}_k}{\partial \boldsymbol{\eta}} + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}} \mathbf{\Pi}_k \tag{9}$$

wherein the derivatives of $\boldsymbol{f}$ are computed at the k-th state and control samples.

Constraints (7) can be easily translated into constraints on the parameters variation $\boldsymbol{\delta}_\eta$ thanks to (5). However, insights on the used parameterizations can lead to more efficient solutions. First of all, in simple, ZOH and LERP parameterizations the parameters represent a set of control samples, and it is easy to understand that other values cannot exceed (be less than) the largest (smallest) free samples. As a consequence, control bounds can be readily translated into parameter limits:

$$\boldsymbol{\eta}_{min} \leq \boldsymbol{\eta} + \boldsymbol{\delta}_\eta \leq \boldsymbol{\eta}_{max} \tag{10}$$

Similarly, variation constraints need not to be enforced at each discrete sample k, but rather at the free samples j only, giving:

$$-\Delta\boldsymbol{u} \leq (\mathbf{\Pi}_j - \mathbf{\Pi}_{j-1})(\boldsymbol{\eta} + \boldsymbol{\delta}_\eta) \leq \Delta\boldsymbol{u} \tag{11}$$

When function bases are used to parameterize the control inputs, in the general case there is no way to reduce the number of constraints (7). However, if the chosen functions are smooth, it is reasonable to assume that values of consecutive samples will not be too dissimilar. In particular, if a control sample located at a given discrete instant j is within bounds, chances are that neighboring samples will either be within bounds or exceed control limits by a small margin. Furthermore, as the inputs are generated by differentiable functions, control variations $\boldsymbol{u}_k - \boldsymbol{u}_{k-1}$ can be approximated as $\dot{\boldsymbol{u}}_k \Delta t$. Once again, under smoothness assumption, it is reasonable to reduce the number of points at which variation bounds are imposed. To summarize, the set of valid parameters can be approximated by

$$\begin{bmatrix} \boldsymbol{u}_{min} \\ -\Delta\boldsymbol{u} \end{bmatrix} \leq \begin{bmatrix} \mathbf{\Pi}_j \\ \Delta t \mathbf{\Pi}_j^d \end{bmatrix} (\boldsymbol{\eta} + \boldsymbol{\delta}_\eta) \leq \begin{bmatrix} \boldsymbol{u}_{max} \\ \Delta\boldsymbol{u} \end{bmatrix} \tag{12a}$$

which are applied only at some instants j selected by the user. In the relations above, matrix $\mathbf{\Pi}_j^d$ is defined in terms of the derivatives of the basis functions as:

$$\mathbf{\Pi}_j^d = \begin{bmatrix} \dot{\phi}_1(j\Delta t) & \cdots & \dot{\phi}_{N_b}(j\Delta t) \end{bmatrix} \tag{13}$$

The sub-problem defined by (8a) and the constraints (7) is a dense QP optimization that can be solved by state-of-the-art solvers, such as qpOASES [22], HPIPM [23] or OSQP [24]. Once the solution is computed, a line search algorithm can be used to update the current guess of the parameters as $\boldsymbol{\eta}' = \boldsymbol{\eta} + s\boldsymbol{\delta}_\eta$ with $s \in [0, 1]$. The algorithm can then continue by repeating the previous steps over and over until convergence.

It must be noted that, in a classical direct single-shooting formulation, the number of decision variables, control bounds and control variation constraints are all proportional to the prediction horizon N_p. Instead, in the parameterized approach proposed here, they are all proportional to N_η. As shown in the next section, the use of a reduced number of parameters allows for considerable speed-ups in the solution process, without degrading the control performance.

III. Simulations

In this section we benchmark an implementation of the proposed solver in a simulated environment. The parameterization and algorithms detailed in previous sections were implemented as a C++ library[1], using Eigen [25] for linear algebra and qpOASES [22] to solve the dense QP sub-problems. Different parameterizations with increasing number of parameters have been tested to determine the performances of the solver and a comparison with state of the art methods from the acados framework [26] is included.

The system to be controlled is an inverted pendulum with moving base (sometimes referred to as cart-pole system), which is often used to investigate new approaches in nonlinear control theory. The objective here is not to propose a new control strategy for this type of machine but simply to compare the efficiency of the proposed approach with the state of the art in MPC on a simple system. The state is given by the quadruplet (p, θ, v, ω), representing respectively the position of the cart, the angle of the pendulum ($\theta = 0$ meaning that the pendulum points downward), the linear velocity of the base and the angular rate of the pendulum. The control input of the system is a horizontal force pushing the cart. The continuous-time model of the system can be obtained using a Lagrangian approach, which gives the linear and angular accelerations as function of the state and input:

$$\ddot{p} = \frac{I_0 \left(F + \mu\dot{\theta}^2 \sin\theta\right) + g\mu^2 \sin\theta\cos\theta}{m_t I_0 - \mu^2 \cos^2\theta} \tag{14a}$$

$$\ddot{\theta} = \frac{-\left(F + \mu\dot{\theta}^2 \sin\theta\right)\mu\cos\theta - m_t g\mu\sin\theta}{m_t I_0 - \mu^2 \cos^2\theta} \tag{14b}$$

wherein g is the gravity acceleration constant, m_t the total mass of the system (base and pendulum), I_0 the moment of inertia of the pendulum with respect to the pivot point and μ the product of the mass of the pendulum times the distance between its center of mass and the pivot. Numerical values used in the simulations are: $g = 9.806\,\mathrm{m\,s^{-2}}$, $m_t = 0.3678\,\mathrm{kg}$, $I_0 = 0.011\,852\,\mathrm{kg\,m^2}$ and $\mu = 0.035\,34\,\mathrm{kg\,m}$.

Different sets of simulations are presented in the sequel: first, we focus on a task in which the pendulum has to be kept in balance while also tracking a time-varying trajectory with the base. In this first set of simulations, all parameterizations presented in previous sections are tested multiple times with different values of N_η. Afterwards, the results obtained with two parameterizations with $N_\eta = 5$ are compared to four algorithms from acados. Finally, to test the effectiveness of the parameterized approach in a more challenging scenario, a swing-up task is considered.

In all simulations, the time horizon used for the predictions was set to $1\,\mathrm{s}$, and a very high control frequency was employed to test the algorithms under challenging conditions. In particular, all presented simulations were performed with both $\Delta t = 10\,\mathrm{ms}$ and $\Delta t = 2\,\mathrm{ms}$, leading to a prediction horizon of $N_p = 100$ and $N_p = 500$ samples respectively. Tests were run on two different platforms: (1) a Laptop computer featuring an Intel Core i7-8750H @$2.2\,\mathrm{GHz}$ CPU and 16GB of RAM; (2) a Raspberry Pi 400 featuring a Broadcom BCM2711 Cortex-A72 (ARM v8) @$1.8\,\mathrm{GHz}$ processor and 4GB of RAM. In this way, it was possible to get some insights on the performances of the implemented solver both on a workstation and an embedded device.

[1]We are polishing the code and releasing it on GitHub in the final version.

A. Performance benchmarking

In this first set of tests, the objective was to gather insight about the performances of parameterizations for an increasing number of parameters. Each simulation lasts $30\,\mathrm{s}$, with the pendulum starting in the state $(p, \theta, v, \omega) = (0, \pi, 0, 0)$ and having to track the desired states $(p^\star(t), \pi, 0, 0)$, with the position defined as:

$$p^\star(t) = \begin{cases} -0.5 & t < 7.5 \\ 0.5 & 7.5 \leq t < 15 \\ 0.2\sin\left(0.4\pi t\right) & t \geq 15 \end{cases} \tag{15}$$

Weighting matrices in (8) were defined as $\mathbf{Q}_k = \mathrm{diag}(200, 50, 7, 2)/N_p$ and $\mathbf{R}_k = 1/N_p$, and only control bounds (7a) were included, with $\boldsymbol{u}_{max} = -\boldsymbol{u}_{min} = 0.1\,\mathrm{N}$.

Concerning the tested parameterizations, the simple, ZOH and LERP ones were tested for $N_\eta = 2, \cdots, 20$, with free samples spread uniformly along the prediction horizon in the case of ZOH and LERP parameterizations. In addition, a further parameterization using a basis of exponentially damped polynomials was tested (later referred to as "Poly", with $N_b = N_\eta = 2, \cdots, 10$ parameters. In each of these cases, the parameter τ was set to $1/(N_\eta - 2)$, so that the maxima of the basis functions are regularly spread along the prediction horizon. In addition, control bounds (7a) were imposed on $2N_\eta$ samples, uniformly distributed along the prediction horizon.

For each parameterization, prediction horizon and number of parameters, the same simulation was repeated 10 times to be able to obtain statistical data on the amount of time required to solve the nonlinear optimization problems. Results are shown in figs. 2 and 3 for experiments run on a laptop and a Raspberry Pi respectively. For each parameterization, two types of graphs are included: average run-times (per iteration) in microseconds vs the number of parameters (figs. 2(a), 2(c), 3(a) and 3(c)) and average run-times vs average position tracking errors (figs. 2(b), 2(d), 3(b) and 3(d)).

The results show that the simple parameterization consistently requires less computation time than other strategies, but that it fails to converge to a good solution, with tracking errors that remain more or less the same independently from the number of parameters used. This can be justified by the fact that parameters are able to affect only a small portion of the prediction horizon, with most of the control inputs being handled by a single parameter. ZOH and LERP parameterizations have very similar performances in terms of optimization times, with ZOH usually taking slightly more than LERP. However, the former features worse tracking performances for an equal number of parameters. Finally, the

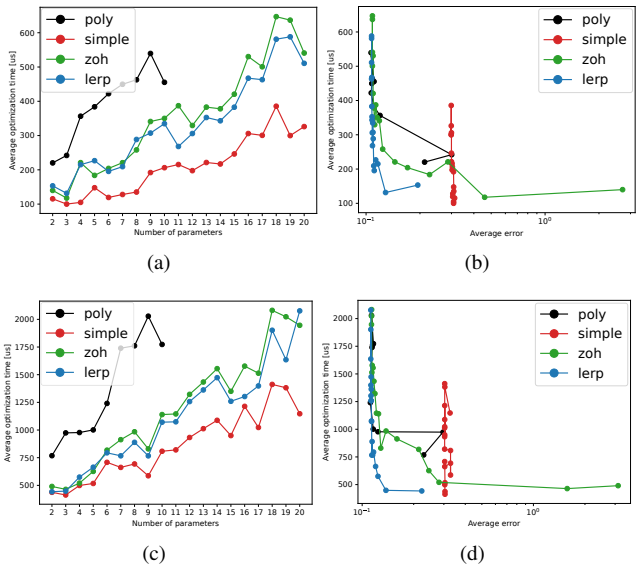


Fig. 2. Performances (on a Laptop) of the proposed parameterizations with increasing number of parameters. In (a) and (c) the average optimization times are reported for increasing numbers of parameters, while (b) and (d) show the relation between optimization times and tracking errors (each curve corresponds to a parameterization, and each data-point to a specific number of parameters). The prediction horizon for (a) and (b) is $N_p = 100$, while for (c) and (d) it is $N_p = 500$.

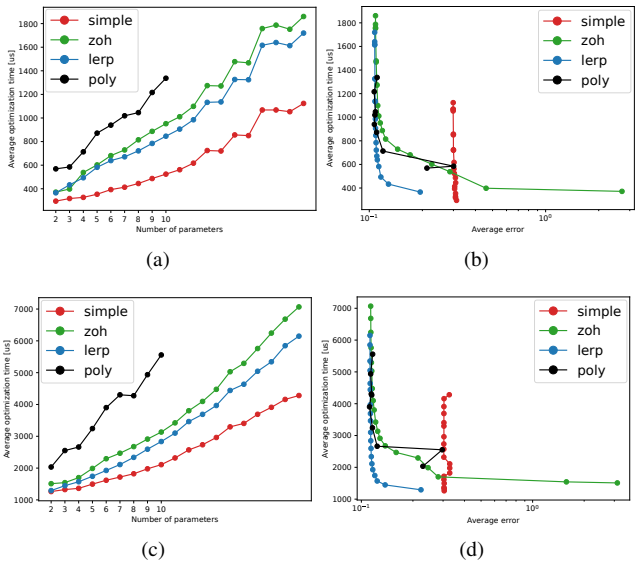


Fig. 3. Performances of the proposed parameterizations on a Raspberry Pi board. The meaning of each plot is analogous to those depicted in fig. 2.

Poly parameterization takes the longest computation times, which can be explained by the fact that computing the control sequence from parameters requires more flops and that the number of constraints is larger than in other cases. Nonetheless, it generally reaches good tracking performances with less computation times than the ZOH parameterization.

It is interesting to notice that in multiple cases the average optimization times are less than a millisecond. In particular, the results show that almost all the parameterizations could have been employed in real-time experiments in the case of a laptop. The average optimization times on the Raspberry board are larger, but still compatible with real-time requirements in the case of $N_p = 100$.

B. Comparison against existing solvers: trajectory tracking

To better place the obtained results within the state of the art, the same simulations performed in the previous section were implemented also using the tools contained in acados. All tuning parameters of the problem, such as the weights in the objective function, were taken as in the previous section to be as fair as possible in the comparison. Termination conditions were also chosen to ensure that the results from the different strategies had the same level of optimality. Four multiple-shooting solvers were tested: one based on qpOASES and three HPIPM variants featuring full, partial and no condensing. As before, the simulations were repeated 10 times per solver, with N_p set to 100 or 500.

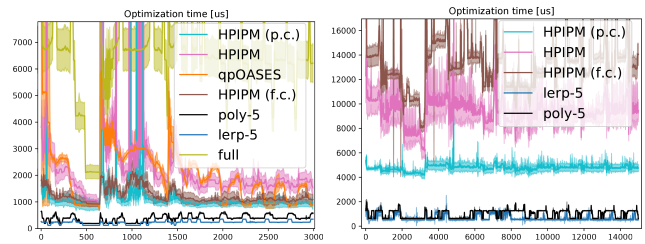
Average time performances (with 95 % confidence intervals) at each iteration of the simulation are plotted in fig. 4 for all acados' algorithms and for LERP and Poly parameterizations[2] (with $N_\eta = 5$). In addition, for the case $N_p = 100$, we include the results obtained using a simple parameterization with $N_\eta = N_p = 100$. This corresponds to a MPC problem with full degrees of freedom and is therefore labeled as "full" in the results. In addition, a summary of numerical values is included in the top-half of table I. In particular, for each algorithm we report the average optimization time over all iterations and all simulations in the different conditions. In addition, we evaluated a numerical index that quantifies how faster an algorithm is with respect to a given "baseline". This index is evaluated as:

$$speed\,gain = \frac{\sum_{i=1}^{N_{iters}} \frac{T_{opt}^{bl}[i]}{T_{opt}^{solver}[i]}}{N_{iters}} \tag{16}$$

where N_{iters} is the number of iterations per simulation and $T_{opt}^{solver}[i]$ and $T_{opt}^{bl}[i]$ are the average optimization times required at the i-th iteration to solve the SQP problem using respectively the given $solver$ and the baseline (therefore, it does not necessarily coincide with the overall average time of the baseline divided by that of the solver).

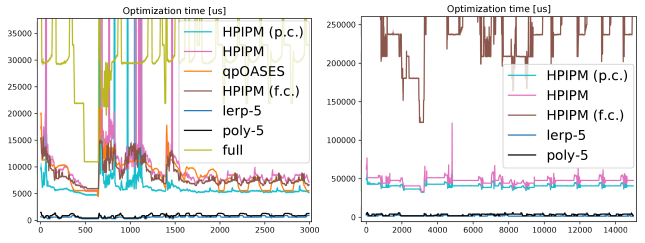
Two observations are due. First of all, the two parameterized approaches consistently take shorter computation times than acados' solvers (they are always at the bottom of the plots shown in fig. 4). The Poly and in particular the LERP parameterizations both converge to a good solution faster than every solver from acados – especially on the Raspberry board, where they run more than 10 times faster on average. Furthermore, even though the values have been omitted for brevity, the average tracking errors obtained with these two parameterizations are slightly smaller than those of acados' algorithms, showing that shorter optimization times have not been obtained by sacrificing optimality.

[2]Simple and ZOH were not considered here since with 5 parameters they did not converge to satisfactory solutions in terms of position error.



(a) Laptop, $N_p = 100$

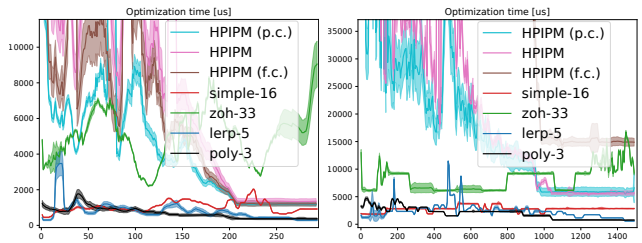
(b) Laptop, $N_p = 500$



(c) Raspberry Pi, $N_p = 100$

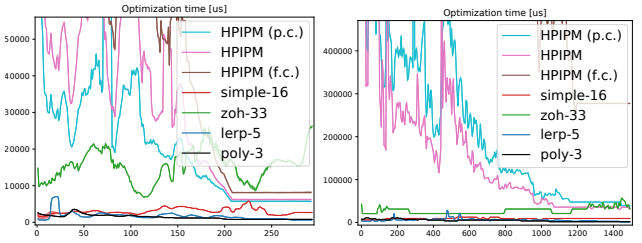
(d) Raspberry Pi, $N_p = 500$

Fig. 4. Optimization times for the tracking task. HPIPM (f.c.), HPIPM (p.c), HPIPM: acados solvers based on HPIPM (with full, partial and no condensing); qpOASES: acados solver based on qpOASES; lerp-5, poly-5: proposed solver with $N_\eta = 5$. full: proposed solver with $N_\eta = N_p$.



(a) Laptop, $N_p = 100$

(b) Laptop, $N_p = 500$



(c) Raspberry Pi, $N_p = 100$

(d) Raspberry Pi, $N_p = 500$

Fig. 5. Optimization times for the swing-up task.

The second important remark is that the full parameterization takes much longer than any algorithm in acados. Since its working principles are close to those of the qpOASES-based solver in acados, this suggests that there is possibly room for further improvement in the efficiency of our software implementation.

C. Comparison against existing solvers: swing-up

The last set of simulations considered here involves a swing-up task, with the pendulum starting from the origin of its state space and having to reach $(0, \pi, 0, 0)$ as desired configuration. Weighting matrices were changed to $\mathbf{Q}_k = \mathrm{diag}(10^3, 10^3, 10^{-2}, 10^{-2})/N_p$ and $\mathbf{R}_k = 10^{-2}/N_p$, and the control bounds relaxed to $\boldsymbol{u}_{max} = -\boldsymbol{u}_{min} = 12\,\mathrm{N}$. To make the problem more challenging, we also added variation constraints[3] (7b) with $\Delta\boldsymbol{u} = 3\Delta t \boldsymbol{u}_{max}$.

Considering the proposed solver, we decided to find for each parameterization the minimum number of N_η that was sufficient to complete the task within 3 s. It turns out that the simple and ZOH parameterizations require at least 16 and 33 parameters respectively (see fig. 1). Instead, LERP and Poly parameterizations were successful with as few as 5 and 3 parameters each. Moreover, the full parameterization and the qpOASES-based solver in acados both failed to converge in a reasonable amount of time and were thus not included in the analysis.

Similarly to what done in the previous section, we show plots of the average optimization times during the simulations in fig. 5, while numerical results are included in the bottom-half of table I. Once again, the parameterized methods LERP and Poly are confirmed to be faster than existing solvers, by quite a large margin. In particular, it is worth remarking that on the Raspberry Pi board with $N_p = 100$, only the parameterized approaches Simple, LERP and Poly converged within the sampling period of $\Delta t = 10\,\mathrm{ms}$ for more than half of the simulation.

[3]Since in acados this is not directly possible, we added the force as a new state variable and used its derivative as the new control. In this way, both the actuation force (as a state) and its variation (being the new input) can be bounded.

IV. Conclusions

In this work we reviewed the benefits of using input parameterizations in Model Predictive Control to achieve fast and reliable performances. The advantages of these strategies are twofold. On one hand, they allow to reduce the number of optimization variables and of the constraints, making the problem solvable even on embedded hardware with very limited computational resources. On the other, they improve the performances of the control with respect to receding horizon techniques characterized by a small control horizon. We implemented a single-shooting SQP algorithm to solve the nonlinear MPC optimization and showed that, coupled with LERP and function basis parameterizations, it can consistently outperform state-of-the-art solvers.

The results obtained so far are very promising, and we believe that it is worth investigating parameterizations more in depth. In particular, we would like to focus our attention on different types of basis functions and extend the analysis to nonlinear parameterizations. The stability study of these parameterized problems is also an interesting track, that could be addressed by extending the stability results already obtained in the framework of the full parameterization [6].

Several works in the field suggest that multiple-shooting methods can provide more stable and fast solutions when formulating and solving the optimization instances. Indeed, ad-hoc condensing strategies for ZOH parameterizations have already been proposed [27]. We are confident that other parameterized approaches would also benefit from the use of lifted techniques. Furthermore, we would like to

TABLE I

Summary of performance comparison between state-of-the-art solvers included in acados and the proposed parameterized approach.

n.a.: 'not available' (the computation time was long and the test was not performed) bl.: 'baseline' (this is the algorithm used to evaluate the relative speed gain).

		$N_p = 100$		$N_p = 500$		$N_p = 100$		$N_p = 500$	
	Solver	**Opt.time** [ms]	**Speed gain**	**Opt.time** [ms]	**Speed gain**	**Opt.time** [ms]	**Speed gain**	**Opt.time** [ms]	**Speed gain**
trajectory tracking	HPIPM (f.c.)	1.231	bl.	14.447	0.368	8.438	bl.	275.161	0.167
	HPIPM (p.c.)	1.439	1.172	4.783	bl.	7.470	1.421	40.641	bl.
	HPIPM	3.090	0.616	9.494	0.511	12.176	0.917	47.535	0.859
	qpOASES	2.101	0.709	n.a.	n.a.	8.678	1.008	n.a.	n.a.
	full	6.698	0.204	n.a.	n.a.	31.050	0.295	n.a.	n.a.
	lerp-5	**0.227**	6.214	**0.664**	7.806	**0.582**	16.673	**1.743**	25.082
	poly-5	0.384	3.802	1.001	5.384	0.858	11.458	3.429	13.312
swing-up	HPIPM (f.c.)	6.645	0.879	183.052	0.254	60.106	0.541	4320.603	0.083
	HPIPM (p.c.)	4.545	bl.	22.190	bl.	22.762	bl.	274.899	0.771
	HPIPM	8.225	0.675	31.334	0.970	31.484	0.801	189.600	bl.
	simple-16	1.031	5.584	2.784	9.326	3.014	9.194	9.201	24.277
	zoh-33	5.090	1.267	8.249	3.140	15.595	2.055	27.812	8.052
	lerp-5	0.783	7.362	2.339	11.590	1.640	17.525	5.718	41.544
	poly-3	**0.688**	5.900	**1.981**	10.026	**1.411**	14.279	**4.733**	34.055
		Laptop				*Raspberry Pi*			

optimize our software library and add new features such as automatic differentiation and code generation, and possibly incorporating it into existing frameworks [28], [26].

REFERENCES

[1] D. Q. Mayne, "Model predictive control: Recent developments and future promise," *Automatica*, vol. 50, pp. 2967–2986, 2014.

[2] B. Kouvaritakis, M. Cannon, and J. A. Rossiter, "Who needs QP for linear mpc anyway?" *Automatica*, vol. 27, pp. 879–884, 2002.

[3] ——, "Efficient robust predictive control," *IEEE Transactions on Automatic Control*, vol. 45, pp. 1545–1549, 2002.

[4] H. Ferreau, H. Bock, and M. Diehl, "An online active set strategy to overcome the limitations of explicit MPC," *International Journal of Robust and Nonlinear Control*, vol. 518, pp. 816–830, 2008.

[5] A. Bemporad, F. Borrelli, and M. Morari, "Model predictive control based on linear programming - the explicit solution," *IEEE Transactions on Automatic Control*, vol. 47, pp. 1974–1985, 2003.

[6] M. Alamir, *Stabilization of Nonlinear Systems Using Receding-Horizon Control Schemes: a Parametrized Approach for Fast Systems*, ser. Lecture Notes in Control and Information Sciences. Springer, 2006.

[7] E. T. Van Donkelaar, O. H. Bosgra, and P. M. J. Van den Hof, "Model predictive control with generalized input parametrization," in *European Control Conference*, 1999, pp. 1693–1698.

[8] M. Muehlebach and R. D'Andrea, "Parametrized infinite-horizon model predictive control for linear time-invariant systems with input and state constraints," in *American Control Conference*, 2016, pp. 2669–2674.

[9] J. O. A. Limaverde Filho, T. S. Lourenco, E. Fortaleza, A. Murilo, and R. Lopes, "Trajectory tracking for a quadrotor system: A flatness-based nonlinear predictive control approach," in *2016 IEEE Conference on Control Applications (CCA)*. IEEE, 2016, pp. 1380–1385.

[10] T. Binder, L. Blank, H. G. Bock, R. Bulirsch, W. Dahmen, M. Diehl, T. Kronseder, W. Marquardt, J. P. Schlöder, and O. von Stryk, "Introduction to model based optimization of chemical processes on moving horizons," in *Online optimization of large scale systems*. Springer, 2001, pp. 295–339.

[11] M. Diehl, H. G. Bock, H. Diedam, and P.-B. Wieber, "Fast direct multiple shooting algorithms for optimal robot control," in *Fast motions in biomechanics and robotics*. Springer, 2006, pp. 65–93.

[12] R. Findeisen and F. Allgower, "An introduction to nonlinear model predictive control," in *21st Benelux meeting on systems and control*, vol. 11. Technische Universiteit Eindhoven Veldhoven Eindhoven, The Netherlands, 2002, pp. 119–141.

[13] H. G. Bock and K.-J. Plitt, "A multiple shooting algorithm for direct solution of optimal control problems," *IFAC Proceedings Volumes*, vol. 17, no. 2, pp. 1603–1608, 1984.

[14] J. Albersmeyer and M. Diehl, "The lifted newton method and its application in optimization," *SIAM Journal on Optimization*, vol. 20, no. 3, pp. 1655–1684, 2010.

[15] M. Giftthaler, M. Neunert, M. Stauble, J. Buchli, and M. Diehl, "A family of iterative gauss-newton shooting methods for nonlinear optimal control," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 1–9.

[16] S.-P. Han, "A globally convergent method for nonlinear programming," *Journal of optimization theory and applications*, vol. 22, no. 3, pp. 297–309, 1977.

[17] M. J. D. Powell, "A fast algorithm for nonlinearly constrained optimization calculations," in *Numerical analysis*. Springer, 1978, pp. 144–157.

[18] G. Allibert, E. Courtial, and F. Chaumette, "Predictive control for constrained image-based visual servoing," *IEEE Transactions on Robotics*, vol. 26, no. 5, pp. 933–939, 2010.

[19] D. Perez-Morales, O. Kermorgant, S. Dominguez-Quijada, and P. Martinet, "Multi-sensor-based predictive control for autonomous parking in presence of pedestrians," in *16th International Conference on Control, Automation, Robotics and Vision (ICARCV)*. IEEE, 2020, pp. 406–413.

[20] R. Cagienard, P. Grieder, E. C. Kerrigan, and M. Morari, "Move blocking strategies in receding horizon control," *Journal of Process Control*, vol. 17, no. 6, pp. 563–570, 2007.

[21] J. H. Lee, Y. Chikkula, Z. Yu, and J. C. Kantor, "Improving computational efficiency of model predictive control algorithm using wavelet transformation," *International Journal of Control*, vol. 61, no. 4, pp. 859–883, 1995.

[22] H. J. Ferreau, C. Kirches, A. Potschka, H. G. Bock, and M. Diehl, "qpoases: A parametric active-set algorithm for quadratic programming," *Mathematical Programming Computation*, vol. 6, no. 4, pp. 327–363, 2014.

[23] G. Frison and M. Diehl, "HPIPM: a high-performance quadratic programming framework for model predictive control," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 6563–6569, 2020.

[24] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: an operator splitting solver for quadratic programs," *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, 2020.

[25] G. Guennebaud, B. Jacob *et al.*, "Eigen v3," http://eigen.tuxfamily.org, 2010.

[26] R. Verschueren, G. Frison, D. Kouzoupis, J. Frey, N. v. Duijkeren, A. Zanelli, B. Novoselnik, T. Albin, R. Quirynen, and M. Diehl, "acados—a modular open-source framework for fast embedded optimal control," *Mathematical Programming Computation*, pp. 1–37, 2021.

[27] Y. Chen, N. Scarabottolo, M. Bruschetta, and A. Beghi, "Efficient move blocking strategy for multiple shooting-based non-linear model predictive control," *IET Control Theory & Applications*, vol. 14, no. 2, pp. 343–351, 2019.

[28] M. Giftthaler, M. Neunert, M. Stauble, and J. Buchli, "The control toolbox – an open-source C++ library for robotics, optimal and model predictive control," *2018 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, pp. 123–129, 2018.