Benchmarking Nonlinear Model Predictive Control with Input Parameterizations

Franco Fusco, Guillaume Allibert, Olivier Kermorgant, Philippe Martinet

Benchmarking Nonlinear Model Predictive Control with Input Parameterizations

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

***Abstract*— Model Predictive Control (MPC) while being a very effective control technique can become computationally demanding when a large prediction horizon is selected. To make the problem more tractable, 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 benchmark the performances of the solver in simulation and compare them with state-of-the-art solvers. Results show that parameterizations allow to attain good performances with (significantly) lower computation times.**

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. 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 like [2], [3], [4], [5] to cite a few.

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]. 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.

[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

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 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] .

Direct methods work 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 optimal control problem can be written as:

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

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 $\mathbf{A}$. $\boldsymbol{x}_0$ 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 $\underline{\boldsymbol{x}}$ (respectively the sequence of controls and states) 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 [12], [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 states can be evaluated iteratively starting from $\boldsymbol{x}_0$ and the selected control inputs using (1). In this paper, we have focused our attention on the later.

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.

One way to simplify the optimization would be to allow only the first $N_c \leq N_p$ control samples to be freely allocated in the optimization [15], [16], with the remaining $N_p - N_c$ samples set to the same value of $\boldsymbol{u}_{N_c-1}$ (fig. 1(a)). More generally, the "free samples" from $\underline{\boldsymbol{u}}$ could be spread uniformly along the prediction horizon rather than at the beginning, with remaining values obtained by "holding" the free samples constant (fig. 1(b)) or alternatively using linear interpolation (fig. 1(c)). These strategies are known in the literature as *Move Blocking*. For clarity, they will be referred to as "simple", "ZOH" and "LERP" in the sequel.

A more general option 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 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 [17]. In this paper, we propose to use a set of exponentially damped polynomials in the form:

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

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.

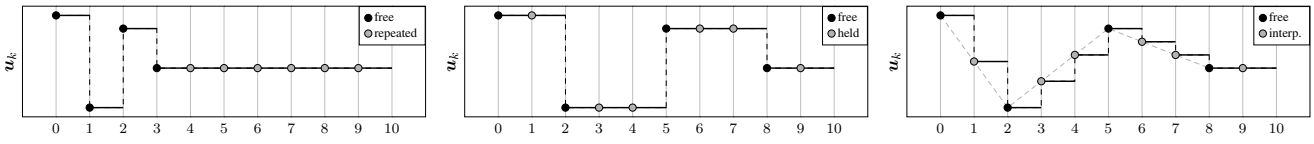
The interest in using a basis of functions is that the overall control sequence can "inherit" some desirable properties from its generating functions. As an example, if all ϕ_i are smooth (such as in (4)) then the generated sequences naturally feature reduced variations between consecutive samples. In 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 linear mapping from a given vector of parameters $\boldsymbol{\eta} \in \mathbb{R}^{N_\eta}$ to the control sequence $\underline{\boldsymbol{u}}$,

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

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 subproblem, whose solution allows to update the current guess of the parameters $\boldsymbol{\eta}$. These two steps (solution of the subproblem 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. Concerning the definition of the quadratic subproblem, 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 (1). States and controls in the objective are then linearized around $\boldsymbol{\eta} + \boldsymbol{\delta}_\eta$,



(a) Simple (b) ZOH (*Zero-Order-Holder*) (c) LERP (*linear interpolation*)

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

with $\boldsymbol{\delta}_\eta$ representing variation of the current parameters. Injecting the linearized states into (2a) leads to

$$\min \frac{1}{2}\boldsymbol{\delta}_\eta^T \mathbf{H} \boldsymbol{\delta}_\eta + \boldsymbol{g}^T \boldsymbol{\delta}_\eta \tag{6a}$$

i.e., a QP with Gauss-Newton Hessian approximation, with

$$\mathbf{H} = \sum_{k=1}^{N_p} \frac{\partial \boldsymbol{x}_k}{\partial \boldsymbol{\eta}}^T \mathbf{Q}_k \frac{\partial \boldsymbol{x}_k}{\partial \boldsymbol{\eta}} + \sum_{k=0}^{N_p-1} \boldsymbol{\Pi}_k^T \mathbf{R}_k \boldsymbol{\Pi}_k \tag{6b}$$

$$\boldsymbol{g}^T = \sum_{k=1}^{N_p} (\boldsymbol{x}_k - \boldsymbol{x}_k^\star)^T \mathbf{Q}_k \frac{\partial \boldsymbol{x}_k}{\partial \boldsymbol{\eta}} + \sum_{k=0}^{N_p-1} (\boldsymbol{u}_k - \boldsymbol{u}_k^\star)^T \mathbf{R}_k \boldsymbol{\Pi}_k \tag{6c}$$

The Jacobians of the states with respect to the parameters can be evaluated using the recursive relation

$$\frac{\partial \boldsymbol{x}_{k+1}}{\partial \boldsymbol{\eta}} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} \frac{\partial \boldsymbol{x}_k}{\partial \boldsymbol{\eta}} + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}} \boldsymbol{\Pi}_k \tag{7}$$

Constraints on the magnitude of the control actions and its variation can be easily translated into constraints on the parameters variation $\boldsymbol{\delta}_\eta$ thanks to (5). Nevertheless, it is worth noting that 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. Similarly, variation constraints need not to be enforced at each discrete sample k, but rather at the free samples only.

When function bases are used to parameterize the control inputs, in the general case there is no way to reduce the number of control constraints. 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 and a similar argument holds for control variations. Therefore, only a subset of the original constraints can be considered at the cost of (small) constraints violations, granting a considerable simplification of the problem.

The sub-problem defined by (6a) and the constraints is a dense QP optimization that can be solved by state-of-the-art solvers, such as qpOASES [18], HPIPM [19] or OSQP [20]. Once the solution is computed, a line search algorithm can 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 formulation the number of decision variables and constraints are 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, using Eigen [21] for linear algebra and qpOASES [18] 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 [22] 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) (position of the cart, angle of the pendulum, linear and angular velocities). The reader is referred to [22] for the equations adopted in our simulations.

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 GHz CPU

and 16GB of RAM; (2) a Raspberry Pi 400 featuring a Broadcom BCM2711 Cortex-A72 (ARM v8) @1.8 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.

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 s, with the pendulum already starting upward and tracking a time-varying position while keeping balance. Weighting matrices in (6) 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 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 were imposed on $2N_\eta$ samples 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 ZOH. However, the former features worse tracking performances for an equal number of parameters. Finally, the 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 $N_p = 100$.

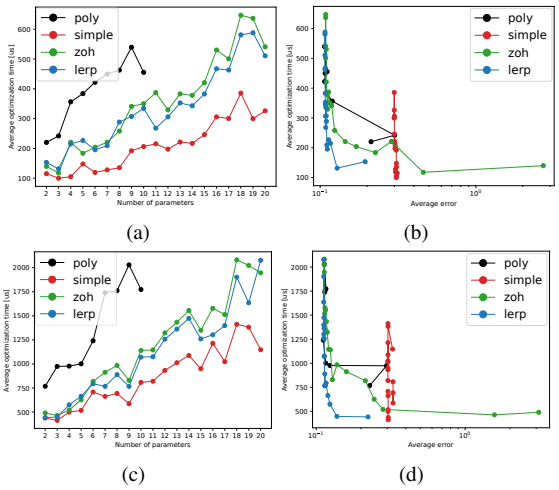


Fig. 2. Performances (on a Laptop) of the proposed parameterizations with increasing number of parameters. In (a) and (c) the average optimization times are reported for increasing numbers of parameters, while (b) and (d) show the relation between optimization times and tracking errors (each curve corresponds 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 $N_p = 500$.

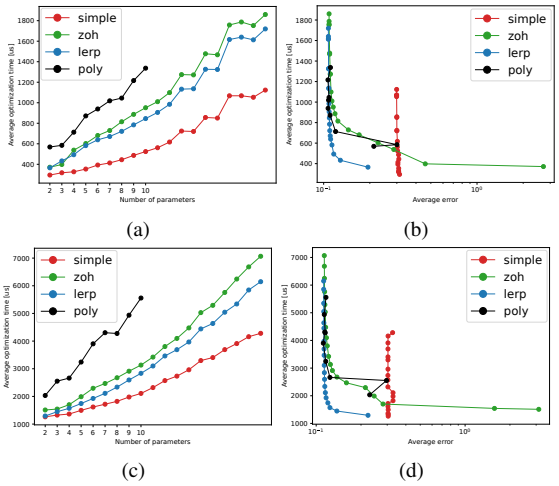


Fig. 3. Performances of the proposed parameterizations on a Raspberry Pi board. The meaning of each plot is analogous to those depicted in fig. 2.

B. Comparison against existing solvers: trajectory tracking

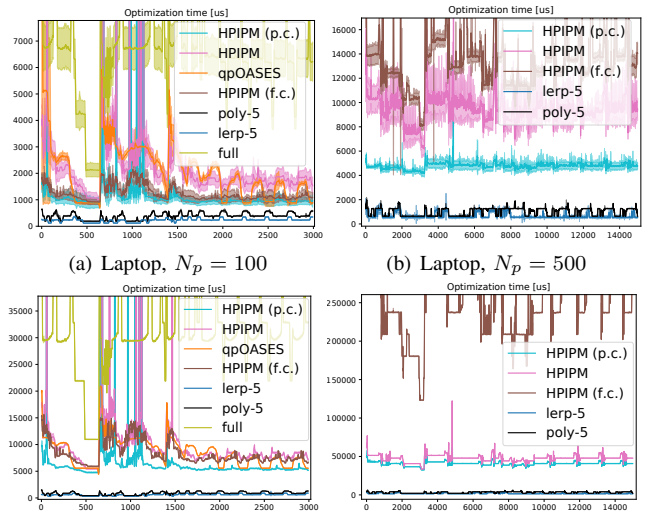
To better place the obtained results within the state of the art, the same simulations performed in the previous section were implemented also using the tools contained in acados. All tuning parameters of the problem, such as the weights in the objective function, were taken as in the previous section 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 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[1] (with $N_\eta = 5$). In addition, for the case $N_p = 100$, we include the results obtained using a 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:

$$speed\,gain = \frac{1}{N_{iters}} \sum_{i=1}^{N_{iters}} \frac{T_{opt}^{bl}[i]}{T_{opt}^{solver}[i]} \tag{8}$$

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.



(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 $N_\eta = N_p$.

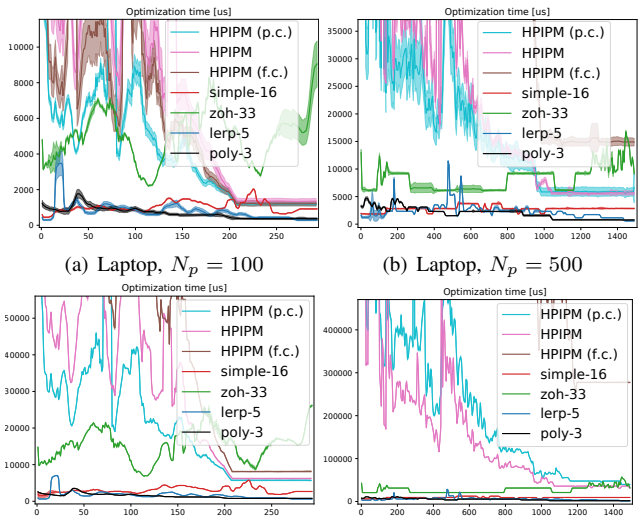
The two parameterized approaches consistently take shorter computation times than acados' solvers (they are always at the bottom of the plots shown in fig. 4). The Poly and in particular the LERP parameterizations both converge to a good solution faster than every solver from acados – especially on the 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. Also note 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 room for improvement in the efficiency of our software implementation.

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

C. Comparison against existing solvers: swing-up

The last set of simulations considered here involves a swing-up task, with the pendulum starting downward and having to reach the upward position. 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 control variation constraints.

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. 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 took too long to converge and were thus not included in the analysis.



(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.

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 converged within the sampling period of $\Delta t = 10\,\mathrm{ms}$ consistently.

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*			

IV. Conclusions

In this work we reviewed the benefits of using input parameterizations in Model Predictive Control to achieve fast and reliable performances. 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. We implemented a single-shooting SQP algorithm to solve the nonlinear MPC optimization and showed that 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. We would like to focus our attention on different types of basis functions and extend the analysis to nonlinear parameterizations. Furthermore, several works in the field suggest that multiple-shooting methods can provide more stable and fast solutions and ad-hoc condensing strategies for ZOH parameterizations have already been proposed [23]. We are confident that other parameterized approaches would also benefit from the use of lifted techniques. Furthermore, we would like to optimize our software library, add new features and possibly incorporate it into existing frameworks.

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] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

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

[22] 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.

[23] 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.