Efficient Calibration of Embedded MPC

Marco Forgione[1], Dario Piga[1], and Alberto Bemporad[2]

[1]IDSIA Dalle Molle Institute for Artificial Intelligence SUPSI-USI, Manno, Switzerland
[2]IMT School for Advanced Studies Lucca, Lucca, Italy

April 16, 2021

Abstract

Model Predictive Control (MPC) is a powerful and flexible design tool of high-performance controllers for physical systems in the presence of input and output constraints. A challenge for the practitioner applying MPC is the need of tuning a large number of parameters such as prediction and control horizons, weight matrices of the MPC cost function, and observer gains, according to different trade-offs. The MPC design task is even more involved when the control law has to be deployed to an embedded hardware unit endowed with limited computational resources. In this case, real-time system requirements limit the complexity of the applicable MPC configuration, engendering additional design tradeoffs and requiring to tune further parameters, such as the sampling time and the tolerances used in the on-line numerical solver. To take into account closed-loop performance and real-time requirements, in this paper we tackle the embedded MPC design problem using a global, data-driven, optimization approach We showcase the potential of this approach by tuning an MPC controller on two hardware platforms characterized by largely different computational capabilities.

To cite this work, please use the following bibtex entry:

```
@inproceedings{forgione2020efficient,
  title={{E}fficient {C}alibration of {E}mbedded {MPC}},
  author={Forgione, Marco and Piga, Dario and Bemporad, Alberto},
  booktitle={Proc. of the 21st IFAC World Congress, Berlin, Germany},
  year={2020}
}
```

Using the plain bibtex style, the bibliographic entry should look like:

M. Forgione, D. Piga, and A. Bemporad. Efficient Calibration of Embedded MPC. In *Proc. of the 21st IFAC World Congress, Berlin, Germany*, 2020.

1 Introduction

Model Prediction Control (MPC) is an advanced control technology that is getting widely popular in different application domains (Borrelli et al., 2017). The main technical reason of its success is the ability to optimally coordinate inputs and outputs of multivariable systems in the presence of input/output constraints. Besides, the intuitive interpretation of MPC as an optimal controller with respect to a given objective function makes it accessible even to practitioners with limited control background.

Nonetheless, calibrating a high-performance MPC controller taking advantage of all the available tuning knobs may still require substantial effort. The challenges normally encountered are: (i) to choose parameters such as prediction and horizon, cost function weight matrices, and observer gains in order to meet desired closed-loop requirements; (ii) to implement the MPC control law on the target hardware platform, ensuring that all computations are performed in real time.

Regarding challenge (i), the final MPC control law is determined by the prediction model, the specified cost function, and input/output constraints. Leaving aside the constraints, which may be considered direct problem specifications, the practitioner has yet to define the plant model for prediction and the cost function. The model is typically obtained from first-principle laws or estimated from measured data. However, when deriving such a model, a tradeoff emerges between accuracy and complexity, and, most of the times, it is difficult to decide *a priori* how accurate the model should be in order to achieve satisfactory closed-loop performance (Formentin et al., 2016; Piga et al., 2018).

As for the MPC cost function, it should represent the underlying engineering or economic objective and — in some cases — it could also be a direct specification. However, the MPC cost function is often constrained to have a specific structure, typically a quadratic form of predicted input/output values, to allow the use of very efficient quadratic programming (QP) numerical optimization algorithms. Conversely, the true underlying objective resulting from a combination of time- or frequency-domain specifications (or economical considerations) may be formulated more naturally in a different form than a purely quadratic objective. In these cases, the MPC cost function has to be considered as a tuning knob available to achieve the actual closed-loop goal, rather than an exact quantification of the goal itself.

Challenge (ii) is particularly relevant for fast systems controlled by embedded hardware platforms, such as mobile robots, automotive and aerospace systems, *etc.* Indeed, in embedded systems the on-board computational power is usually limited, *e.g.*, by cost, weight, power consumption, and battery life constraints. Therefore, real-time requirements must be taken into account, fur-

ther complicating the overall MPC design task. For instance, due to throughput and memory limitations, the MPC sampling time cannot be chosen arbitrarily small and the control horizon cannot be chosen arbitrarily large. Moreover, the low-level design choices of the MPC implementation become crucial and should be taken into account in the design phase. For instance, the engineer should carefully decide whether to go for an explicit MPC approach where the solution is pre-computed offline for all states in a given range (Bemporad et al., 2002), or solve the MPC problem on-line by numerical optimization. The first approach is potentially very fast, but requires storing a large lookup table whose size increases with the MPC problem complexity. The applicability of this method is thus limited by the available system memory. Conversely, the second approach has generally a smaller memory footprint, but requires solving a numerical optimization problem on-line. Thus, in the latter case, the MPC problem complexity and the hardware's computational power limit the maximum controller update frequency.

When the MPC law is computed by numerical optimization (a QP solver in case of linear MPC), the hyperparameters of the optimizer are also MPC tuning knobs, in that they affect solution accuracy and required computation time. The overall design of a high-performance MPC must therefore take into account simultaneously high-level aspects related to control systems (model, weights, constraints, prediction horizon, sample time, *etc.*) and low-level aspects of numerical optimization (problem size and solver-related hyperparametes).

In recent years, data-driven approaches for solving complex engineering tuning problems based on derivative-free global optimization are gaining increasing attention (Shahriari et al., 2016). The idea behind these approaches is rather intuitive: the user specifies a search space for the design parameters and the optimization algorithm, based on performance data previously observed, sequentially suggests the new configurations to be tested, aiming to optimize a user-given performance index. The procedure is iterated until a configuration achieving satisfactory performance is found or the maximum number of available tests has been reached. This approach has also been popularized as Design and Analysis of Computer Experiments (DACE) (Sacks et al., 1989). Specialized global optimization algorithms for this task such as Bayesian Optimization (BO) have been proposed (Brochu et al., 2010). The key feature of these algorithms is their ability to optimize the objective function with a small number of (possibly noisy) evaluations, without relying on derivative information. Recently, optimization-based tuning has been successfully applied to control system design (Roveda et al., 2019; Drieß et al., 2017) and to choose the MPC prediction model (Piga et al., 2019; Bansal et al., 2017).

In this paper, we demonstrate the potential of the optimization-based data-driven approach for the joint tuning of high- and low-level MPC parameters in order to optimize a certain closed-loop performance objective, while ensuring that the control law can be computed in real-time on the hardware platform at hand. We apply a derivative-free global optimization algorithm recently developed by one of the authors (Bemporad, 2019), which has been shown to be very efficient in terms of number of function evaluations required to solve

the global optimization problem. We present the results of our MPC tuning procedure for a simulated cart-pole system on two hardware platforms with very different computational capability, namely a high-end x86-64 workstation and a low-performance ARM-based board (specifically, a Raspberry PI 3, Model B). We show that our tuning procedure can find an MPC configuration that squeezes the maximum performance out of the two architectures for the given control task.

The rest of the paper is organized as follows. The MPC problem formulation and its design parameters are introduced in Section 2. Next, the data-driven MPC calibration strategy based on global optimization is described in Section 3 and numerical examples are presented in Section 4. Finally, conclusions and directions for future research are discussed in Section 5

2 Problem formulation

Let us consider the following nonlinear continuous-time multi-input multi-output dynamical system in state-space form

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

$$y = g(x, u), \tag{1b}$$

where $u \in \mathbb{R}^{n_u}$ is the vector of control inputs, $x \in \mathbb{R}^{n_x}$ the state vector, $y \in \mathbb{R}^{n_y}$ the controlled outputs, $\dot{x}$ denotes the time derivative of the state x, and $f : \mathbb{R}^{n_x+n_u} \to \mathbb{R}^{n_x}$ and $g : \mathbb{R}^{n_x+n_u} \to \mathbb{R}^{n_y}$ are the state and output mappings, respectively. Output variables can be collected and used for real-time control at a sampling time $T_\mathrm{s} \geq T_{\mathrm{s,min}}$, where $T_{\mathrm{s,min}}$ is the minimum sampling time achievable by the measurement system.

We aim at synthesizing an MPC (with a state estimator) for (1) such that the resulting closed-loop system minimizes a certain closed-loop performance index J^cl. This performance index is defined as a continuous-time functional $J^\mathrm{cl}(y_{[0,\, T_\mathrm{exp}]}, u_{[0,\, T_\mathrm{exp}]})$ in an experiment of fixed duration T_exp, where $y_{[0,\, T_\mathrm{exp}]}$ (resp. $u_{[0,\, T_\mathrm{exp}]}$) denotes the output signal $y(t)$ (resp. input signal $u(t)$) within the time interval $[0,\, T_\mathrm{exp}]$. As an additional requirement, we must ensure that the control law can be computed in real-time on a given hardware platform. This requirement is translated into the constraint $T_\mathrm{calc}^\mathrm{MPC} \leq T_\mathrm{s}^\mathrm{MPC}$, where $T_\mathrm{s}^\mathrm{MPC}$ denotes the MPC sampling time and $T_\mathrm{calc}^\mathrm{MPC}$ the worst-case time required to compute the MPC control law on the given platform. It is worth remarking that in general the closed-loop performance index J^cl is a nonconvex function of the MPC design parameters, that we will defined later. For the sake of generality, J^cl has been denoted above as a continuous-time functional over the duration T_exp of the experiment. Thus, J^cl does not necessarily correspond to the cost function minimized on-line by MPC. Indeed, the latter is generally defined as a discrete-time quadratic function over a prediction horizon generally shorter than T_exp.

In the following paragraphs, we define the MPC and state estimator design problem, along with their tuning parameters.

2.1 MPC controller

We assume that a continuous-time (possibly parametrized) model $M(\theta^m)$ of (1) is available, where θ^m represents a vector of adjustable model parameters. For a given choice of $T_\mathrm{s}^\mathrm{MPC}$, $M(\theta^m)$ can be linearized and discretized in time, yielding the discrete-time state-space model

$$x_{t+1} = A(T_\mathrm{s}^\mathrm{MPC}, \theta^m)x_t + B(T_\mathrm{s}^\mathrm{MPC}, \theta^m)u_t \tag{2a}$$

$$y_t = C(\theta^m)x_t + D(\theta^m)u_t \tag{2b}$$

that is used as prediction model for MPC.

At each time t that is an integer multiple of the MPC sampling time $T_\mathrm{s}^\mathrm{MPC}$, MPC solves the minimization problem

$$\begin{aligned}
\min_{\{u_{t+k|t}\}_{k=0}^{N_u-1},\epsilon} & \sum_{k=0}^{N_p-1} \left(y_{t+k|t} - y_{t+k}^\mathrm{ref}\right)^\top Q_y \left(y_{t+k|t} - y_{t+k}^\mathrm{ref}\right) + \\
& + \sum_{k=0}^{N_p-1} \left(u_{t+k|t} - u_{t+k}^\mathrm{ref}\right)^\top Q_u \left(u_{t+k|t} - u_{t+k}^\mathrm{ref}\right) + \\
& + \sum_{k=0}^{N_p-1} \Delta u_{t+k|t}^\top Q_{\Delta u} \Delta u_{t+k|t} + Q_\epsilon \epsilon^2 && \text{(3a)} \\
\text{s.t. model equations (2a), (2b)} & && \text{(3b)} \\
y_\mathrm{min} - V_y\epsilon \le y_{t+k|t} \le y_\mathrm{max} + V_y\epsilon, \ k = 1, \dots, N_p & && \text{(3c)} \\
u_\mathrm{min} - V_u\epsilon \le u_{t+k|t} \le u_\mathrm{max} + V_u\epsilon, \ k = 1, \dots, N_p & && \text{(3d)} \\
\Delta u_\mathrm{min} - V_{\Delta u}\epsilon \le \Delta u_{t+k|t}, \quad k = 1, \dots, N_p & && \text{(3e)} \\
\Delta u_{t+k|t} \le \Delta u_\mathrm{max} + V_{\Delta u}\epsilon, \quad k = 1, \dots, N_p & && \text{(3f)} \\
u_{t+N_u+j|t} = u_{t+N_u|t}, \quad j = 1, \dots, N_p - N_u, & && \text{(3g)}
\end{aligned}$$

where $\Delta u_{t+k|t} = u_{t+k|t} - u_{t+k-1|t}$; N_p and N_u are the prediction and control horizon, respectively; Q_y, Q_u, and $Q_{\Delta u}$ are positive semidefinite weight matrices specifying the MPC cost function; u_ref and y_ref are the input and output references, respectively; Q_ϵ, V_y, V_u, $V_{\Delta u}$ are positive constants used to soften the input and output constraints, ensuring that the optimization problem (3) is always feasible. An MPC calibrator would typically adjust $N_p, N_u, Q_y, Q_u, Q_{\Delta u}$ using a mix of experience and trial-and-error until the desired closed-loop goals are achieved, fixing the remaining parameters to their default value. Such a process, in particular in the absence of a deep knowledge of MPC, can be very time consuming and therefore costly.

Several parameterizations may be used to simplify the calibration task. For instance, weight matrices Q_y, Q_u, and $Q_{\Delta u}$ may be constrained to be diagonal (one of the weights may also be chosen equal to one without loss of generality). For notation convenience, we denote by θ^c the set of all tuning parameters of the MPC problem introduced above.

The solution of the QP problem (3) is computed through numerical optimization. We denote by θ^s the QP solver settings, that we assume can also be adjusted by the calibrator. For instance, important solver parameters are the QP's relative and absolute feasibility/optimality tolerances for termination. Note that the parameters θ^s influence both the accuracy of the numerical solution (thus, the performance index J^{cl}) and the computation time (thus, $T_{\mathrm{calc}}^{\mathrm{MPC}}$).

2.2 State estimator

An estimate of the system state x_t is required to solve the MPC optimization problem (3). In this paper, we use a Luenberger observer for state estimation:

$$\hat{x}_{t+1|t} = A\hat{x}_{t|t-1} + Bu_t + L(y_t - C\hat{x}_{t|t-1}) \tag{4a}$$

$$\hat{y}_{t+1|t} = C\hat{x}_{t+1|t}, \tag{4b}$$

where $\hat{x}_{t|t-1}$ is the state estimate at time t based on observations up to time $t-1$. We compute the gain L as the standard stationary Kalman filter gain, based on the (linearized) model (2), assuming positive definite covariance matrices W_w and W_v for the additive process and measurement noise, respectively. As for the MPC tuning weights, different parametrizations/structures may be used to define such covariance matrices. The corresponding parameters are the tuning knobs of the state estimator and are denoted as θ^f.

3 Performance-driven parameter tuning

For notation convenience, the design parameters θ^m, θ^c, θ^s, θ^f introduced in the previous section are collected in the single vector $\theta \in \mathbb{R}^{n_\theta}$.

In this section, we describe how to tune θ through an experiment-driven approach in order to optimize the closed-loop performance index $J^{\mathrm{cl}}(y_{[0,T_{\mathrm{exp}}]}, u_{[0,T_{\mathrm{exp}}]}; \theta)$, under the real-time constraint $T_{\mathrm{calc}}^{\mathrm{MPC}}(\theta) \leq T_{\mathrm{s}}^{\mathrm{MPC}}$. The overall MPC design task can be formalized as the following constrained global optimization problem

$$\min_{\theta \in \Theta} J^{\mathrm{cl}}(y_{[0,T_{\mathrm{exp}}]}, u_{[0,T_{\mathrm{exp}}]}; \theta) \tag{5a}$$

$$\text{s.t. } T_{\mathrm{calc}}^{\mathrm{MPC}}(\theta) \leq \eta T_{\mathrm{s}}^{\mathrm{MPC}}. \tag{5b}$$

In (5a), $\Theta \subseteq \mathbb{R}^{n_\theta}$ is the set of admissible values of the design vector θ. Specifically, in this work, Θ is a box-shaped region delimited by lower and upper bounds for each individual parameter. The constant η, $0 < \eta < 1$ in (5b) takes into account that, in a practical implementation, a fraction of the controller's computation time should be left available for other tasks.

It is important to stress that neither for $J^{\mathrm{cl}}(\theta)$ nor for $T_{\mathrm{calc}}^{\mathrm{MPC}}(\theta)$ a closed-form expression is available. Nevertheless, these functions can be evaluated through real experiments or simulations[1]. In the following, we describe the optimization algorithm used to solve problem (5) based on function evaluations of J^{cl} and

[1] In order to evaluate the worst-case computation time $T_{\mathrm{calc}}^{\mathrm{MPC}}$ for a given parameter θ, the

$T_{\text{calc}}^{\text{MPC}}$. One of main strengths of this algorithm is its efficiency in terms of number of required function evaluations.

3.1 Global optimization for parameter selection

First, the constrained optimization problem (5) is approximated by the following box-constrained problem

$$\min_{\theta \in \Theta} \tilde{J}^{\text{cl}}(\theta), \tag{6a}$$

where $\tilde{J}^{\text{cl}}(\theta)$ is defined as the original cost $J^{\text{cl}}(\theta)$ plus a continuous barrier function $\ell : \mathbb{R} \to \mathbb{R}^+$ penalizing the violation of the constraint $T_{\text{calc}}^{\text{MPC}}(\theta) \leq \eta T_{\text{s}}^{\text{MPC}}$, *i.e.*,

$$\tilde{J}^{\text{cl}}(\theta) = J^{\text{cl}}(\theta) + \ell\left(T_{\text{calc}}^{\text{MPC}}(\theta) - \eta T_{\text{s}}^{\text{MPC}}\right). \tag{6b}$$

We solve the optimization problem (6) by using the approach described in (Bemporad, 2019), called GLIS (GLobal optimization based on Inverse distance weighting and radial basis function Surrogates).

The algorithm first runs $n_{\text{in}} \geq 1$ closed-loop experiments for n_{in} different values of the vector of controller parameters θ_n, $n = 1, \ldots, n_{\text{in}}$, generated randomly using Latin Hypercube Sampling (LHS) (McKay et al., 1979) within Θ, measuring the corresponding performance index $\tilde{J}_n^{\text{cl}}$. Next, at iteration $n \geq n_{\text{in}}$ a radial basis function $\hat{f}_{\text{RBF}}$ is fit to the available samples $\{(\theta_1, \tilde{J}_1^{\text{cl}}), \ldots, (\theta_n, \tilde{J}_n^{\text{cl}})\}$. This function $\hat{f}_{\text{RBF}}$ is a surrogate of the (non-quantified) performance index $\tilde{J}^{\text{cl}}$. Another function $z : \Theta \to \mathbb{R}$, that promotes the exploration of the set Θ in areas that have not been sampled yet and where the empirical variance of $\tilde{J}^{\text{cl}} - \hat{f}_{\text{RBF}}$ is large, is summed to $\hat{f}_{\text{RBF}}$ to define an acquisition function $a : \Theta \to \mathbb{R}$. The function a (which is very easy to evaluate and even differentiate) is then minimized over Θ by using a global optimization algorithm, such as Particle Swarm Optimization (PSO) (Eberhart and Kennedy, 1995; Vaz and Vicente, 2009), to get a new configuration θ_{n+1} of MPC parameters to test. A new closed-loop experiment is performed with controller parameters $\theta = \theta_{n+1}$ and the performance index $\tilde{J}_{n+1}^{\text{cl}}$ is measured. The algorithm iterates until a stopping criterion is met or a maximum number $n_{\max}$ of iterations is reached.

The main advantage of using the global optimization algorithm GLIS described above for solving the calibration problem is twofold. First, it is a derivative-free algorithm, and thus it is particularly convenient since a closed-form expression of the cost $\tilde{J}^{\text{cl}}$ as a function of the design parameters θ is not available. Second, it allows us to tune the controller parameters θ with a smaller number of experiments compared to other existing global optimization methods, such as PSO, DIRECT (DIvide a hyper-RECTangle) (Jones, 2009), Multilevel Coordinate Search (MCS) (Huyer and Neumaier, 1999), Genetic Algorithms

control law should be computed on the target hardware directly. In the absence of a platform emulator, this can be done, for instance, with an *hardware-in-the-loop* setup where the target hardware closes the control loop on a simulated system.

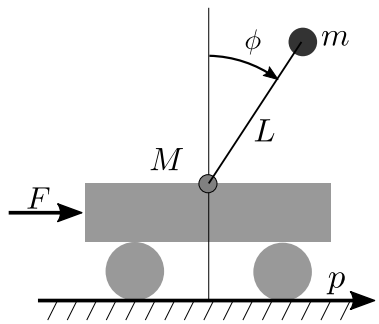

Figure 1: Schematics of the cart-pole system.

(GA) (Hansen, 2006), and usually even than BO as reported in (Bemporad, 2019).

4 Numerical Example

As a case study, we consider the problem of controlling the cart-pole system depicted in Fig. 1. We aim at designing an MPC controller that minimizes a given closed-loop performance index J^{cl}, while satisfying the constraint (5b) on the worst-case MPC execution time $T_{\text{calc}}^{\text{MPC}}$ for real-time implementation. The MPC law is computed using a custom-made Python library that transforms problem (3) into a standard QP form, which is subsequently solved using the ADMM-based QP solver OSQP (Stellato et al., 2018). The MPC parameters are tuned via global optimization using the solver GLIS recalled in Section 3.1 and retrieved from `http://cse.lab.imtlucca.it/~bemporad/glis`. In particular, package version 1.1 was used with default settings.

The source code generating the results in this paper can be found at `https://github.com/forgi86/efficient-calibration-embedded-MPC`. A standalone installation version of the MPC library is also available at `https://github.com/forgi86/pyMPC` for convenient integration in other projects.

The MPC tuning is performed for two different hardware platforms with remarkably different speed performance:

- an x86-64 PC equipped with an Intel i5-7300U 2.60 GHz CPU and 32 GB of RAM;
- a Raspberry PI 3 rev. B board equipped with a 1.2 GHz ARM Cortex-A53 CPU and 1 GB of RAM.

The Raspberry PI 3 is roughly 10 times slower than the PC in computing the MPC law. This leads to different constraints on the maximum controller complexity and thus on the achievable closed-loop performance.

4.1 System description

The cart-pole dynamics are governed by the following continuous-time differential equations which are used to simulate the behavior of the system:

$$\begin{aligned}(M+m)\ddot{p} + mL\ddot{\phi}\cos\phi - mL\dot{\phi}^2\sin\phi + b\dot{p} &= F \\ L\ddot{\phi} + \ddot{p}\cos\phi - g\sin\phi + f_\phi\dot{\phi} &= 0,\end{aligned}$$

where ϕ (rad) is the angle of the pendulum with respect to the upright vertical position, p (m) is the cart position, and F (N) is the input force on the cart. The following values of the physical parameters are used: $M = 0.5\,\text{kg}$ (cart mass), $m = 0.2\,\text{kg}$ (pendulum mass), $L = 0.3\,\text{m}$ (rod length), $g = 9.81\,\text{m/s}^2$ (gravitational acceleration), $b = 0.1\,\text{N/m/s}$, and $f_\phi = 0.1\,\text{m/s}$ (friction terms). Measurements of p and ϕ are supposed to be collected at a minimum sampling time $T_{\text{s,min}} = 1\,\text{ms}$, and are corrupted by additive zero-mean white Gaussian noise sources with standard deviation $0.02\,\text{m}$ and $0.01\,\text{rad}$, respectively. The input force F is perturbed by an additive zero-mean colored Gaussian noise with standard deviation $0.1\,\text{N}$ and bandwidth $5\,\text{rad/s}$. The force F is bounded to the interval $[-10,\ 10]\,\text{N}$, which is due to actuator saturation, while the cart position p is limited to the interval $[-1.2,\ 1.2]\,\text{m}$, representing the length of the track where the cart moves.

The system is initialized at $[p(0)\ \dot{p}(0)\ \phi(0)\ \dot{\phi}(0)] = [0\ 0\ \frac{\pi}{18}\ 0]$. In the MPC calibration experiment, the objective is to track a piecewise linear position reference p^{ref} for the cart passing through the time-position points $\{(0,0)$, $(5,0)$, $(10,0.8)$, $(20,0.8)$, $(25,0)$, $(30,0)$, $(40,0.8)\}$ over an experiment of duration $T_{\text{exp}} = 40$ s, while stabilizing the angle ϕ to the upright vertical position, *i.e.,* at $\phi = 0$.

The controller is disabled at time $T_{\text{stop}} < T_{\text{exp}}$ whenever one of the following early termination condition occurs:

- pendulum falling ($|\phi| > \frac{\pi}{6}$)
- cart approaching end of the track ($|p| \geq 1.1$ m)
- numerical errors in the MPC law computation

Similar conditions may be required to ensure safety in the case of real experiments performed on a physical setup. In our simulation setting, they are still useful to reduce the computational time as they speed up the test of configurations that are definitely not optimal. Furthermore, early termination is explicitly penalized in our closed-loop performance index (see next paragraph), and thus provides useful information for MPC calibration to the global optimization algorithm.

4.2 Performance index

The closed-loop performance index $\tilde{J}^{\mathrm{cl}}$ is defined as

$$\tilde{J}^{\mathrm{cl}} = \ln\left(\int_{t=0}^{T_{\mathrm{exp}}} 10|p^{\mathrm{ref}}(t) - p(t)| + 30|\phi(t)|\, dt\right) + \\ + \ell\left(T_{\mathrm{calc}}^{\mathrm{MPC}} - \eta T_{\mathrm{s}}^{\mathrm{MPC}}\right) + \ell'(T_{\mathrm{exp}} - T_{\mathrm{stop}}), \quad (7)$$

where the penalty term ℓ for real-time implementation of the control law is

$$\ell = \begin{cases} \ln\left(1 + 10^3 \frac{T_{\mathrm{calc}}^{\mathrm{MPC}} - \eta T_{\mathrm{s}}^{\mathrm{MPC}}}{\eta T_{\mathrm{s}}^{\mathrm{MPC}}}\right) & \text{if } T_{\mathrm{calc}}^{\mathrm{MPC}} > \eta T_{\mathrm{s}}^{\mathrm{MPC}} \\ 0 & \text{otherwise,} \end{cases} \quad (8)$$

with $\eta = 0.8$. Another term ℓ' is used to penalize early termination conditions, as previously discussed:

$$\ell' = \begin{cases} \ln\left(1 + 10^3 \frac{T_{\mathrm{exp}} - T_{\mathrm{stop}}}{T_{\mathrm{exp}}}\right) & \text{if } T_{\mathrm{stop}} < T_{\mathrm{exp}} \\ 0 & \text{otherwise.} \end{cases} \quad (9)$$

The integral in (7) is approximated using samples collected at the fastest sampling time $T_{\mathrm{s,min}}$.

4.3 Control design parameters

We have different MPC design parameters to tune in order to minimize the performance-driven objective (7).

As for the MPC cost function, the positive definite weight matrix Q_y is diagonally parameterized as $\left(\begin{smallmatrix} q_{y_{11}} & 0 \\ 0 & q_{y_{22}} \end{smallmatrix}\right)$, where $q_{y_{11}}$, and $q_{y_{22}}$ are design parameters taking real values in the interval $[10^{-16},\ 1]$. Since in the example $n_u = 1$, the weights Q_u and $Q_{\Delta u}$ are scalars. $Q_{\Delta u}$ is taken as a decision variable and bounded in the range $[10^{-16},\ 1]$, while Q_u is set to 0. The prediction horizon N_p is an integer parameter in the range $[5,\ 300]$, while the control horizon N_u is a fraction ϵ_c of N_p rounded to the closest integer, with design parameter ϵ_c restricted to the range $[0.3,\ 1]$. Lastly, the MPC sampling time $T_{\mathrm{s}}^{\mathrm{MPC}}$ is a design parameter restricted to the range $[1,\ 50]$ ms.

In the QP solver, the relative and absolute feasibility/optimality tolerances are tuned. Specifically, the log of two tolerances are parameters in the range $[-7,\ -1]$.

As for the state estimator, the 4x4 process noise covariance matrix W_w and the 2x2 output noise covariance matrix W_v are diagonally parametrized, similarly to Q_y.

The system dynamics are assumed to be known. Therefore, there is no tunable model parameter in our design problem.

Finally, MPC is configured with fixed constraints $F_{\max} = -F_{\min} = 10$ N and $p_{\max} = -p_{\min} = 1$ m on the input force F and on the output position p,

respectively, while standard values are used for all other MPC settings in (3) not mentioned here.

The design parameter θ has thus dimension $n_\theta = 14$.

4.4 Results

The global optimizer GLIS is run for $n_{\max} = 500$ iterations. The performance cost $\tilde{J}^{\text{cl}}$ (7) *vs.* the iteration index n is shown in Fig. 2 for the PC (top) and the Raspberry PI 3 (bottom). It can be observed that, after about 100 iterations, the majority of the controller parameter configurations proposed by the optimization algorithm are concentrated in regions with low cost $\tilde{J}^{\text{cl}}$. This is more evident on the Raspberry PI 3 platform, where the set of parameters satisfying the real-time constraint (5b) is smaller. Persistent high values $\tilde{J}^{\text{cl}}$ are due (correctly) to the exploration of the parameter space Θ promoted by the GLIS algorithm.

The obtained optimal performance index $\tilde{J}^{\text{cl}}$ after 500 iterations is slightly better on the PC (-0.44) than on the Raspberry PI (0.02), as expected. Indeed, certain MPC configurations characterized, *e.g.*, by small sampling time and long prediction horizon may be feasible on the PC, but not on the Raspberry PI.

Fig. 3 shows the time trajectories of position p, angle ϕ, and force F for the optimal MPC controller on the PC (top panel) and on the Raspberry PI (bottom panel), over a validation reference trajectory different from the one used for calibration. A slightly better performance for the MPC implementation on the PC can be appreciated both in terms of a tighter cart position tracking and a lower variance in the angle signal.

Analyzing the two final MPC designs, we noticed that on the PC we have $T_{\text{s}}^{\text{MPC}} = 6$ ms, while on the Raspberry PI we have $T_{\text{s}}^{\text{MPC}} = 22$ ms. The optimal solution found for the PC platform allows a faster loop update, and thus achieves superior trajectory tracking and noise rejection capabilities. On the other hand, a larger MPC time is required on the Raspberry PI to guarantee real-time implementation.

5 Conclusions and follow-up

We have presented an automated method to calibrate MPC parameters with a limited number of experiments. Real-time implementation constraints are explicitly taken into account in the design in order to allow embedded implementation of the resulting controller.

Future research will be devoted to find a parameterized solution of the optimal MPC tuning knobs with respect to the reference trajectories, and to the analysis of the generalization properties of the designed controllers against control objectives not considered in the calibration phase.

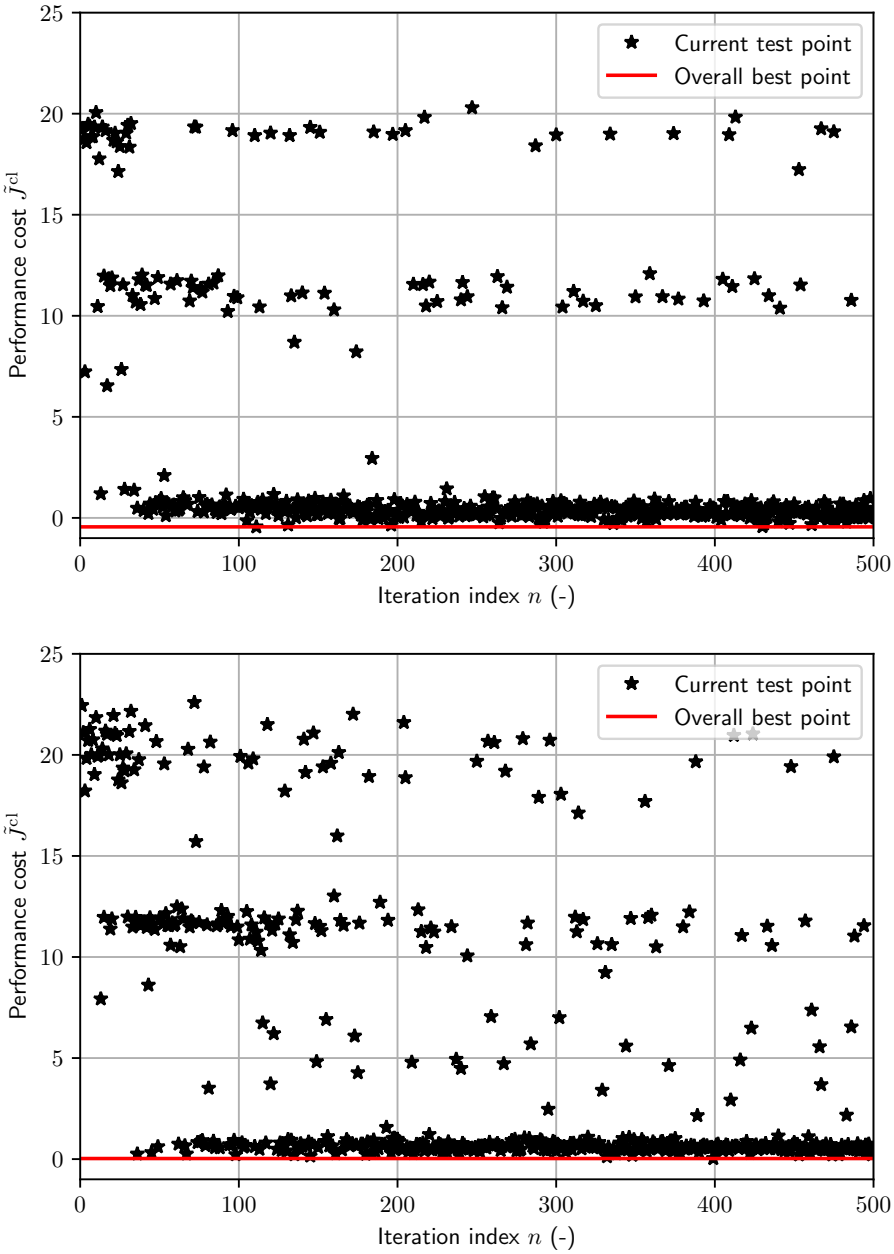

Figure 2: Performance cost $\tilde{J}^{\mathrm{cl}}$ vs. iteration index n of GLIS for experiments run on the PC (top) and on the Raspberry PI (bottom).

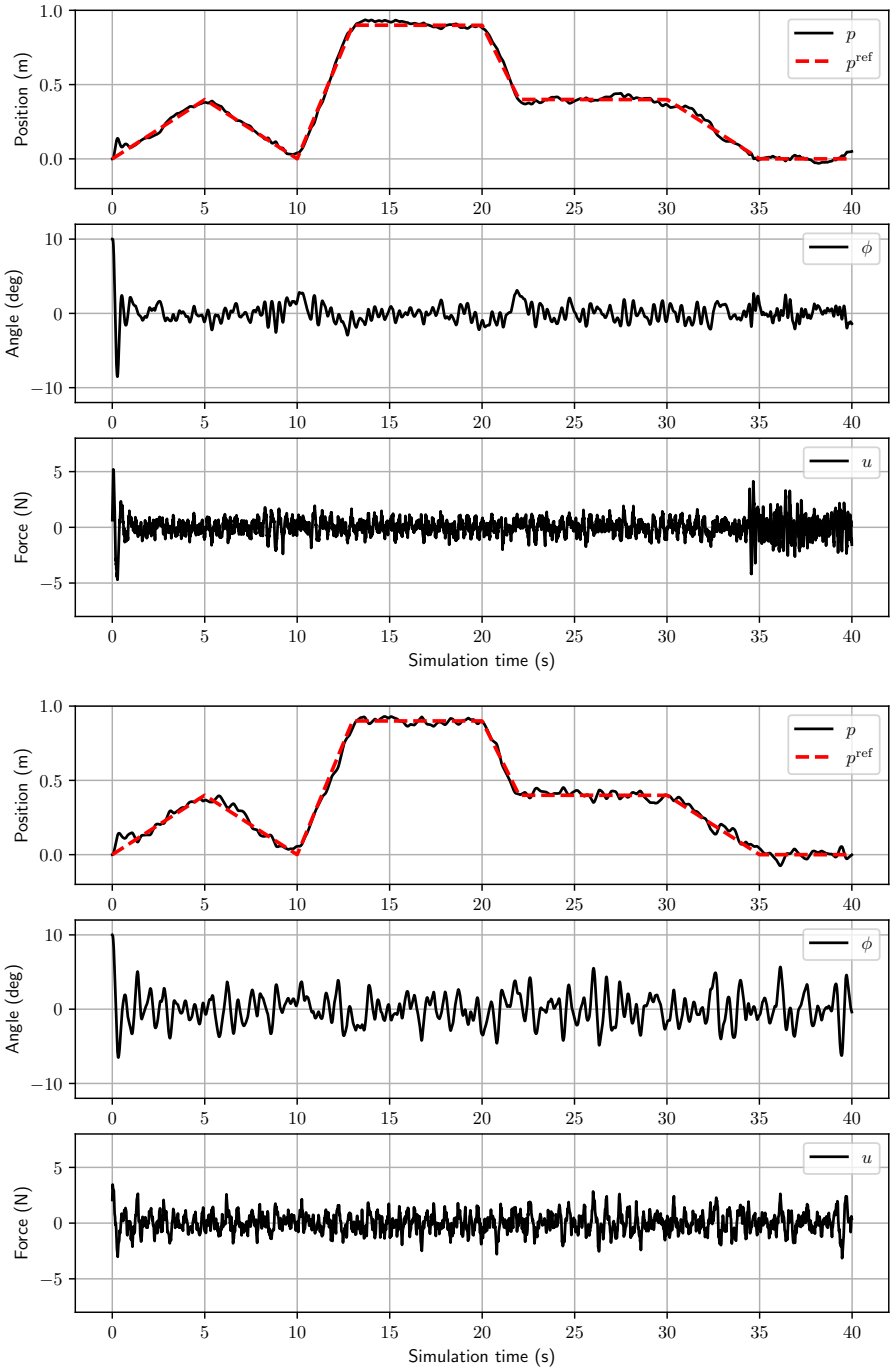

Figure 3: Performance achieved by the designed MPC for experiments run on the PC (top panel) and on the Raspberry PI (bottom panel).

References

S. Bansal, R. Calandra, T. Xiao, S. Levine, and C. J. Tomlin. Goal-driven dynamics learning via bayesian optimization. In *IEEE 56th Annual Conference on Decision and Control*, pages 5168–5173, 2017.

A. Bemporad. Global optimization via inverse distance weighting. *arXiv preprint arXiv:1906.06498*, 2019. `http://cse.lab.imtlucca.it/~bemporad/glis/`.

A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos. The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1):3–20, 2002.

F. Borrelli, A. Bemporad, and M. Morari. *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.

E. Brochu, V. M. Cora, and N. De Freitas. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010.

D. Drieß, P. Englert, and M. Toussaint. Constrained bayesian optimization of combined interaction force/task space controllers for manipulations. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 902–907, 2017.

R. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pages 39–43, Nagoya, 1995.

S. Formentin, D. Piga, R. Tóth, and S. M. Saveresi. Direct learning of LPV controllers from data. *Automatica*, 65:98–110, 2016.

N. Hansen. The CMA evolution strategy: a comparing review. In *Towards a new evolutionary computation*, pages 75–102. 2006.

W. Huyer and A. Neumaier. Global optimization by multilevel coordinate search. *Journal of Global Optimization*, 14(4):331–355, 1999.

D. Jones. DIRECT global optimization algorithm. *Encyclopedia of Optimization*, pages 725–735, 2009.

M. D. McKay, R. J. Beckman, and W. J. Conover. Comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2):239–245, 1979.

D. Piga, S. Formentin, and A. Bemporad. Direct data-driven control of constrained systems. *IEEE Transactions on Control Systems Technology*, 25(4): 331–351, 2018.

D. Piga, M. Forgione, S. Formentin, and A. Bemporad. Performance-oriented model learning for data-driven MPC design. *IEEE Control Systems Letters*, 3(3):577–582, 2019.

L. Roveda, M. Forgione, and D. Piga. Control parameters tuning based on bayesian optimization for robot trajectory tracking. In *2019 Workshop on Learning for Industry 4.0 (ICRA)*, pages 902–907, 2019.

J. Sacks, W. Welch, T. Mitchell, and H. Wynn. Design and analysis of computer experiments. *Statistical Science*, pages 409–423, 1989.

B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd. OSQP: An operator splitting solver for quadratic programs. In *2018 UKACC 12th International Conference on Control (CONTROL)*, pages 339–339, 2018. `https://osqp.org`.

A. Vaz and L. Vicente. PSwarm: A hybrid solver for linearly constrained global derivative-free optimization. *Optimization Methods and Software*, 24:669–685, 2009. `http://www.norg.uminho.pt/aivaz/pswarm/`.