Derivation of Certification-Based Admissibility Dashboard of NMPC Implementation Settings: Framework and Associated `Python` Package

Mazen Alamir

***Abstract*— This brief presents a framework that delivers a certification-oriented dashboard of admissible nonlinear model predictive control (NMPC) implementation settings. This differs from the commonly adopted performance-centered tuning approaches by providing a dashboard of admissible setting options for which the optimal choice might be context-dependent. Some of the considered parameters are scarcely tuned in the literature on model predictive control (MPC)-parameter tuning such as the control updating period and the precision of the internal prediction. Moreover, a freely available `Python`-based implementation is also proposed, and typical results on an illustrative example are discussed highlighting the relevance of the contribution.**

***Index Terms*— Certification, nonlinear model predictive, nonlinear systems, Python package, real-time implementation.**

I. Introduction

THE theoretical foundations of nonlinear model predictive control (NMPC) [1] are now quite established and frameworks [2], [3] embedding efficient and trustworthy dedicated solvers [4], [5] are available.

Nevertheless, the concrete implementation of NMPC involves many choices such as: the auxiliary weights enforcing constraints/stability-related requirements, the prediction horizon, and the control horizon. Some additional choices impact the time-related aspects such as the control updating period, the sampling period in the internal prediction process, and the maximum number of iterations per updating period.

The impacts of all these choices are strongly coupled and the outcome in terms of admissibility (real-time implementability, constraint satisfaction, and stability as detailed in Section III-C) is very difficult to guess making manual tuning a tedious task, especially when the number of underlying free design/implementation parameters grows.

Performing these choices rationally and systematically is referred to as the model predictive control (MPC)-parameter tuning paradigm (see [6] for a recent survey). Shortly speaking, this problem is commonly addressed by focusing on optimizing some measure of the resulting closed-loop performance. To cite but few examples, this can be done using reinforcement learning (RL) [7], [8], genetic algorithms [9] or Bayesian optimization (BO) [10], [11], [12] which aims at replacing the heavy-to-compute cost function by a surrogate model. In these methods, the use of a performance index, which is generally linked to a measure of the closed-loop cost, is instrumental.

Received 13 March 2024; revised 16 August 2024 and 29 October 2024; accepted 6 November 2024. Date of publication 26 November 2024; date of current version 25 February 2025. Recommended by Associate Editor K. Worthmann.

The author is with CNRS, Grenoble INP, GIPSA-Lab, Université Grenoble Alpes, 38000 Grenoble, France (e-mail: mazen.alamir@grenoble-inp.fr).

Digital Object Identifier 10.1109/TCST.2024.3499835

The starting point of the present contribution is to acknowledge that among all *admissible* candidate settings, the selection might not be only based on the closed-loop performance measure. For instance, provided that some performance level is achieved, the control updating rate might be a key quantity when the available computation power is shared by other tasks. The prediction horizon length might also be an important parameter to examine depending on the presence of long-term causalities and the level of uncertainties. Moreover, these context-dependent tradeoffs might be time-varying making relevant the online switches between different admissible settings.

From the above discussion, it appears relevant to come out with a heuristic that does not focus on a specific target criterion but rather delivers a dashboard containing a list of admissible implementation settings for which the different key quantities are computed to allow multiobjective considerations to be easily and dynamically handled.

The novelty of this contribution lies in the following aspects.

1) The MPC setting parameters include relevant implementation options (subsampling for prediction, control updating period, control parameterization, soft constraints penalty, soft/terminal constraints penalty, prediction horizon) that are rarely addressed together in the literature while being frequently used on a trial-and-error basis.
2) The problem formulation that mixes certified feasibility criteria (real-time, constraints satisfaction) and certified stability/performance criterion, both included in a certification framework leading to a nonconventional outcome which consists of providing a list of feasible settings among which the designer can choose depending on its own, potentially multiobjective criterion (short updating periods when high uncertainties are expected, large ones when `cpu`-load is shared among multiple tasks, closed-loop performance when this is the main goal, or through a context-dependent time-varying multiobjective-based tradeoff).
3) A dedicated heuristic of the specificity of the formulated problem, namely: the presence of mixed-integer decision variables and the need for a certification-related heavy computation renders the problem hardly tractable without a dedicated algorithm's design. Moreover, the brief briefly describes a freely available[1] `Python`-based package that implements the proposed solution.

The brief is organized as follows: the problem under consideration is first clearly described in Section II. The

[1] `https://github.com/mazenalamir/MPC_tuner`

implementation of the optimization algorithm that *tunes* the parameters of the NMPC setting is discussed in Section III. A brief description of the *Python* package that implements the algorithm is proposed in Section IV and a concrete example showing its use is depicted in Section V. Finally, the brief ends with a conclusion and a brief discussion regarding possible extensions of the package scope and utilities.

II. Problem Statement

To clearly state the problem under consideration, we need to define the items to be fed as inputs to the algorithm and those NMPC parameters that need to be *tuned* by the algorithm and hence represent its delivered output.

A. Input Items to the Algorithm

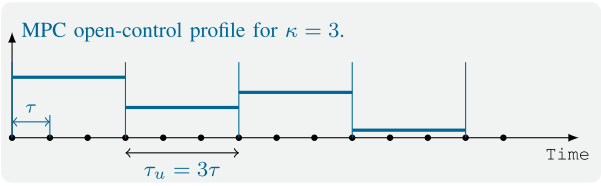
The input items to be provided to the algorithm is defined by the following items:

1) A set of ordinary differential equations (ODEs) governing the dynamics of the system to be controlled which takes the form $\dot{x} = f(x, u, p)$, where $x \in \mathbb{X} \subset \mathbb{R}^{n_x}$, $u \in \mathbb{U} \subset \mathbb{R}^{n_u}$, and $p \in \mathbb{P} \in \mathbb{R}^{n_p}$.
2) A basic sampling period $\tau > 0$ is tailored such that the associated Runge–Kutta (RK) integration scheme yields accurate forward prediction should the expression of f be perfectly known.
3) A stage cost and a terminal penalty function are denoted, respectively, by $\ell(x, u, p, q)$ and $\Psi(x, p, q)$ in which $q \in \mathbb{R}^{n_q}$ is a vector of task-related parameters.
4) A constraint map of the form $c(x, u, p, q) \leq 0 \in \mathbb{R}^{n_c}$. Notice that all the constraints other than the input saturation-related ones are treated as **soft constraints** through appropriate highly weighted exact penalty (to be tuned as shown later on).
5) Bounds on the search domain that are detailed later on.
6) A random sampling function that can be used to generate a representative cloud of initial states x_0, model's parameters vector p, and context parameter vector q.

B. NMPC Parameters to be Tuned (Algorithm's Output)

The list of parameters to be tuned by the algorithms contains the following items:

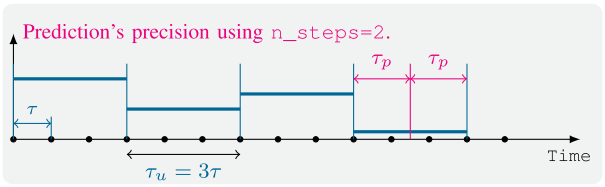
1) The control updating period $\tau_u = \kappa\tau$ expressed as an integer multiple κ of the basic sampling period τ invoked above. This explicitly means that the computation rounds that update the control profiles and hence the implemented closed-loop control are separated by $\kappa\tau$ time units. The next figure illustrates this parameter for $\kappa = 3$.



2) The prediction's precision parameter $\mu_d \in [0, 1]$ determines the precision of the integration used in the MPC-related computation. More precisely, if no precision drop is used, then κ steps of RK(τ) are performed with a sampling period of τ each to predict the τ_u-prediction step. Now MPC practice suggests that the closed-loop might be successful while using `n_steps` ($\leq \kappa$) inner steps of RK($\frac{\tau_u}{\texttt{n\_steps}}$) to get a step prediction over τ_u, this is expressed by ($\lceil r \rceil$ denotes the smallest integer greater than r)

$$\texttt{n\_steps} := \lceil 1 + \mu_d(\kappa - 1) \rceil \tag{1}$$

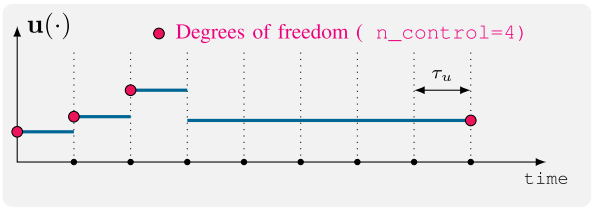
namely $\mu_d = 0$ yields a single RK(τ_u) *large* step (low precision, faster computation) while $\mu_d = 1$ yields κ *small* steps of RK(τ) (high precision, longer computation). This is illustrated in the figure for the choices $\kappa = 3$ and $\mu_d = 0.5$ (hence leading to `n_steps=2`).



3) The prediction horizon's length is defined as a multiple of the control updating period $\tau_u = \kappa\tau$, namely,

$$T = N_{\text{pred}} \times \kappa \times \tau = N_{\text{pred}} \times \tau_u.$$

4) The control horizon's length n_{contr}. This is the number of updating instants over the prediction horizon before the control is frozen to the last value leading to the so-called control horizon that is shorter than the prediction horizon.



5) The weighting penalties ρ_f and ρ_{constr} are used to enforce the terminal penalty and the soft constraints, respectively.
6) The maximum number of iterations `max_iter` used in the optimization process.

To summarize, the vector of NMPC design parameters that the algorithm is intended to tune is defined by

The vector of design parameters

$$\pi := \begin{bmatrix} \kappa \\ \mu_d \\ N_{\text{pred}} \\ n_{\text{contr}} \\ \rho_f \\ \rho_{\text{constr}} \\ \texttt{max\_iter} \end{bmatrix} \in \Pi := [\underline{\pi}, \bar{\pi}] \subset \mathbb{R}^{n_\pi} \tag{2}$$

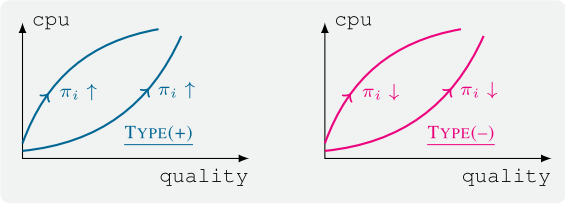


Fig. 1. Any component π_i of the design vector π is either of Type(+) [left] or of Type(−) [right].

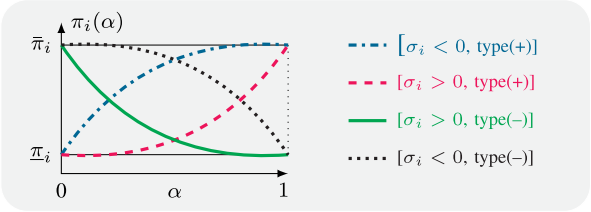


Fig. 2. Typical shapes of $\pi_i(\alpha)$ depending on the sign of σ_i and the type of the component π_i.

in which $\underline{\pi}$ and $\bar{\pi}$ are lower and upper bounds on the components of the design vector π that are to be given as inputs to the tuning algorithm. This corresponds to a quite rich and nonconvex set of possibilities. For each candidate setting π, the corresponding NMPC has to be evaluated for the targeted implementation (device and algorithm) over a high number of relevant scenarios needed for the certification. This cannot be done exhaustively nor is it possible to consider an outer-loop optimizing π while an inner-loop performs a probabilistic certification using a high number of scenarios. Some different heuristics should be derived which is explained in Section III. The problem can be stated as follows.

Problem Statement:

Given the above-defined input elements, an admissible domain $\Pi = [\underline{\pi}, \bar{\pi}]$ and an implementation target, derive a tractable heuristic that followed.

1) Either yields a list of admissible rational choices of the vector of parameters π addressing stability, constraints satisfaction, and real-time implementability concerns.
2) Or it suggests that these requirements cannot be met given the data of the problem.

Such a solution is proposed in Section III below.

III. Proposed Tuning Algorithm

A. Parameterization of the Set of NMPC Design Parameters

To break the complexity of the search over the domain Π defined in (2), a first observation is worth making that can be stated as follows.

Each component π_i of π is of one of the two types shown in Fig. 1, namely, either the pairs (`cpu/quality`) are increasing functions of π_i [TYPE(+)] or they are decreasing functions of π_i [TYPE(−)].

Given the definition (2) of π, it can be rather easily checked that all the components are of Type(+) except $\pi_1 = \kappa$ which is of Type(−). Notice, however, that for a given component, the shape of the monotonicity mentioned above for a given NMPC problem is difficult to know a priori. That is the reason why the shapes of the curves shown in Fig. 1 are determined hereafter via a set of random sampling of a shaping parameter vector [the appropriate shape may not be the same for all the NMPC design parameters (components of π)]

$$\sigma := [\sigma_1, \ldots, \sigma_{n_\pi}] \in \mathcal{S}^{n_\pi}$$

where $n_\pi := \text{card}(\pi) (= 7$ in the current setting) while $\mathcal{S}$ is a set of allowed integers bounded by $\bar{\sigma}$ of the form

$$\mathcal{S} := \{-\bar{\sigma}, \ldots, -1, +1, \ldots \bar{\sigma}\}.$$

For each sampled choice of the shaping parameter vector σ, the shape of the curve representing the monotonicity of π_i is defined via a function $\phi_{\sigma_i}(\alpha) : [0, 1] \to [0, 1]$ which depends on a scalar parameter $\alpha \in [0, 1]$

$$\phi_{\sigma_i}(\alpha) := \begin{cases} \alpha^{\sigma_i} & \text{if } \sigma_i > 0 \quad \text{(positive curvature)} \\ \alpha^{\frac{1}{|\sigma_i|}} & \text{if } \sigma_i < 0 \quad \text{(negative curvature)} \end{cases} \tag{3}$$

and this curve is used to define the value of π_i according to

$$\pi_i(\alpha) := \begin{cases} (1 - \phi_{\sigma_i}(\alpha))\underline{\pi}_i + \phi_{\sigma_i}(\alpha)(\bar{\pi}_i - \underline{\pi}_i) & \text{Type(+)} \\ (1 - \phi_{\sigma_i}(\alpha))\bar{\pi}_i + \phi_{\sigma_i}(\alpha)(\underline{\pi}_i - \bar{\pi}_i) & \text{Type(−).} \end{cases} \tag{4}$$

Consequently, the vector of shaping parameters σ is to be sampled in the set $\mathcal{S}^{n_\pi}$. Fig. 2 illustrates the above definitions for four different configurations of the pair (σ, type).

It is worth noticing that the parameterization defined by (4) is such that, regardless of the type of the component being considered, low values of $\alpha \in [0, 1]$ correspond to low computational complexity and low performance levels while high values induce high computation times and better performance levels should the latter computations be possible within the updating period.

B. Ideal Computation Architecture at a Glance

The remaining task is to find a set of *best* $(\sigma, \alpha) \in \mathcal{S}^{n_\pi} \times [0, 1]$ in terms of the control objective and implementability. Notice the two components of the design, namely, σ and α are to be handled differently. Indeed, it seems reasonably easy to select the scalar α for a given σ as the underlying *indicators* vary monotonically in α. On the contrary, it is unrealistic to attempt a complete and rigorous optimization of $\sigma \in \mathcal{S}^{n_\pi}$ because of the highly combinatorial problem and the involved computational burden for each candidate value.

That is the reason why, the two-layer architecture shown in Fig. 3 is adopted in which `N_trials` values of the shaping parameter vectors $\{\sigma^{(j)} \in \mathcal{S}^{n_\pi}\}_{j=1}^{\texttt{N\_trials}}$ are sampled in $\mathcal{S}^{n_\pi}$ and for each sampled $\sigma^{(j)}$, a scalar constrained optimization on $\alpha \in [0, 1]$ is performed to check whether there is at least one value that meets the requirements (over a predefined set $\mathcal{A}$ of scenarios as explained later on) and if any, find the *optimal* one, denoted by $\hat{\alpha}(\sigma^{(j)}|\mathcal{A})$. These requirements and the associated scalar optimization problem are explained in Section III-C. The computation yields the best cost $\mathcal{J}(\sigma^{(j)}|\mathcal{A})$ given $\sigma^{(j)}$. In case the performance is the main focus, the *best* NMPC design setting pair $(\sigma^\star, \alpha^\star)$ is, therefore, obtained by

$$\sigma^\star \leftarrow \arg\min \left\{ \mathcal{J}(\sigma^{(j)}|\mathcal{A}) \quad j \in \{1, \ldots, \texttt{N\_trials}\} \right\} \tag{5}$$

$$\alpha^\star \leftarrow \hat{\alpha}(\sigma^\star|\mathcal{A}). \tag{6}$$

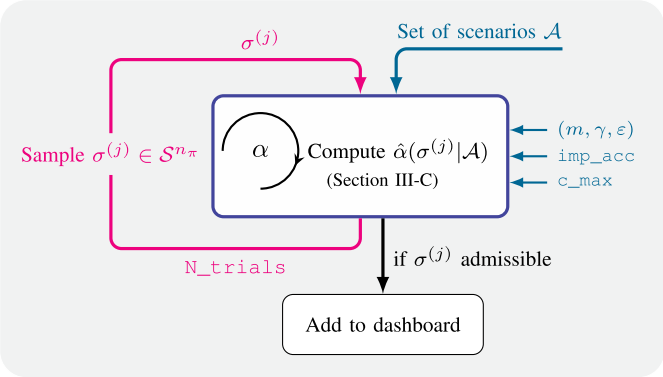


Fig. 3. Architecture of the two-layer algorithm for the determination of the dashboard of admissible implementation settings for a given set of representative scenarios $\mathcal{A}$.

Otherwise, the designer can pick up the more convenient setting from the returned list of feasible settings or he/she can perform a multiobjective optimization within the set of admissible settings. This principle is depicted in Fig. 3 while the details of the boxes' content are explained in the following sections.[2] Let us first focus on the inner optimization loop that computes $\hat{\alpha}(\sigma^{(j)}|\mathcal{A})$. This is the aim of the following section.

C. Admissibility Criteria: Computing $\hat{\alpha}(\sigma|\mathcal{A})$

For a given shaping vector candidate value σ, admissible values of α are those that make the NMPC setting defined by (σ, α) compatible with the requirements regarding the real-time implementation, the constraints satisfaction, and the stability.

More precisely, considering m updating steps over a closed-loop simulation scenario $\mathtt{sc} \in \mathcal{A}$ corresponding to a given initial state, a given model's parameters p and an exogenous task-related parameter vector q (see Section II), there are three concerns when it comes to evaluating the successful (or not) use of the NMPC setting on this specific scenario, namely the following.

1) The real-time feasibility that can be stated by the following constraint ($\lfloor \xi \rfloor_+ := \max(0, \xi)$):

$$\text{RT-feasibility}$$
$$C_{\text{RT}}(\alpha, \sigma|\mathtt{sc}) := \max_{k=1,\ldots,m} \left\lfloor \frac{\tau_{\text{solver}}}{\tau_u}(k) - 1 \right\rfloor_+ = 0 \qquad (7)$$

where $\tau_{\text{solver}}(k)$ denotes the time needed to iterate on the kth optimization problem encountered in the scenario and given the chosen maximum number of iterations $\pi_7(\alpha) = \phi_{\sigma_7}(\alpha)$ corresponding to the currently evaluated σ. Indeed, if the above expression is equal to 0, this means that for all k, one has $\tau_{\text{solver}}(k) \leq \tau_u$. Notice that by replacing τ_u in (7) by $\mathtt{imp\_acc} \times \tau_u$, real-time implementability can be checked for a targeted implementation that is (certifiably) imp_acc faster than the one on which the evaluation is done (or imp_acc times slower for $\mathtt{imp\_acc} < 1$). This can result from the use of different underlying solvers for instance.

[2]In particular, the significance of m, γ, ε, imp_acc, and c_max is given in Section III-C.

2) The contraction property that implicitly assumes that the MPC formulation is such that a decrease of the cost function is expected over the closed-loop trajectory although not necessarily at each step (as in contraction-based formulation for instance [13]). Therefore, using m-step contraction horizon, the associated requirement writes

$$\gamma\text{-Contraction}$$
$$C_\gamma(\alpha, \sigma|\mathtt{sc}) := \lfloor J_{\text{ol}}(m) - \gamma J_{\text{ol}}(1) \rfloor_+ = 0 \qquad (8)$$

where $J_{\text{ol}}(k)$ is the *best open-loop cost value* returned by the solver at the kth updating computation (given the allowed maximum number of iterations corresponding to the design parameter $\pi_7 = \mathtt{max\_iter}$ [see (2)] while $\gamma \in (0, 1)$ is some predefined contraction rate).

3) The constraints satisfaction which applies mainly to soft exact penalty constraints since it is reasonably assumed that the input hard constraints are structurally enforced by the optimization algorithm

$$\text{Constraints satisfaction}$$
$$C_{\text{cstr}}(\alpha, \sigma|\mathtt{sc}) := \max_{k=1,\ldots,m} \lfloor \max_{i \leq n_c} c_i(k) \rfloor_+ = 0. \qquad (9)$$

Notice that the above-mentioned *success conditions*, namely, (7)–(9), concern a specific scenario sc. Now, the overall assessment of a candidate NMPC setting pair $(\sigma, \alpha) \in \mathcal{S}^{n_\pi} \times [0, 1]$ has to be scenario-independent. This is the reason why a set $\mathcal{A}$ of *representative scenarios* is considered. As it is discussed later, the cardinality of this set can be determined following the probabilistic certification formulas [14]. For the remainder of the presentation, it is hence assumed that one disposes of a generator of $\mathcal{A} \leftarrow \mathtt{Generate\_A}(n_{\text{sc}})$ that generates n_{sc} representative scenarios. The set $\mathcal{A}$ is used to compute the *best* value $\hat{\alpha}(\sigma|\mathcal{A})$ for a given σ by solving the following constrained scalar optimization problem:

$$\mathcal{P}(\sigma, \mathcal{A})$$
$$\hat{\alpha}(\sigma|\mathcal{A}) \leftarrow \max_{\alpha \in [0,1]} [\alpha] \qquad (10a)$$
$$\text{under} \left| \begin{array}{l} \max\limits_{\mathtt{sc} \in \mathcal{A}} C_{RT}(\alpha, \sigma|\mathtt{sc}) = 0 \\ \max\limits_{\mathtt{sc} \in \mathcal{A}} C_\gamma(\alpha, \sigma|\mathtt{sc}) = 0 \\ \max\limits_{\mathtt{sc} \in \mathcal{A}} C_{\text{cstr}}(\alpha, \sigma|\mathtt{sc}) \leq \mathtt{c\_max} \end{array} \right. \qquad (10b)$$

where c_max is a threshold on the level of possible violation of the soft constraints. Although one can view (10a) and (10b) as a general constrained optimization problem, the specificity of the problem enables a simple method to derive quite good suboptimal solutions. This is even mandatory given the computation effort needed to evaluate the terms involved in (10a) and (10b) for any candidate value of α (since all the scenarios included in $\mathcal{A}$ are involved). The specific process which is mainly based on a binary search is described in detail by the algorithm of Fig. 4.

This algorithm delivers for each candidate shaping parameter vector σ and for a given set of scenarios $\mathcal{A}$, a success-related Boolean and in the case of success an associated $\hat{\alpha}(\sigma|\mathcal{A})$ together with the associated optimal cost $\mathcal{J}(\sigma|\mathcal{A})$.

Computing $\hat{\alpha}(\sigma|\mathcal{A})$

1) Start with $\alpha = 0$. If (10) is not feasible because of the RT constraint, namely $\max_{\mathtt{sc}} C_{RT}(0, \sigma|\mathtt{sc}) > 0$ return a non feasibility flag for σ, namely $\hat{\alpha}(\sigma|\mathcal{A}) \leftarrow$ `None` and stop. Otherwise proceed into Step 2)

2) Evaluate RT-feasibility for $\alpha = 1$. If the problem is feasible then return $\hat{\alpha}(\sigma|\mathcal{A}) = 1$ otherwise, perform a **binary** search, starting with 0 and 1 as initial extreme values, in order to determine the largest value $\alpha_{\max}(\sigma)$ for which RT-feasibility holds.

3) if at least one of the remaining constraints is violated at $\alpha = \alpha_{\max}(\sigma)$, namely $\max_{\mathtt{sc}} C_\gamma(0, \sigma|\mathtt{sc}) > 0$ or $C_{\mathrm{cstr}}(\alpha, \sigma|\mathtt{sc}) >$ `c_max`, return a non feasibility, namely $\hat{\alpha}(\sigma|\mathcal{A}) \leftarrow$ `None` and stop. Otherwise, returns $\hat{\alpha}(\sigma) = \alpha_{\max}(\sigma)$ and $\mathcal{J}(\sigma|\mathcal{A}) \leftarrow$ the sum of closed-loop costs corresponding to the use of $\alpha_{\max}(\sigma)$ in the scenarios contained in $\mathcal{A}$.

Fig. 4. Algorithm exploiting the specificity of the constrained scalar optimization problem (10a) and (10b) to compute the optimal $\hat{\alpha}(\sigma|\mathcal{A})$ for a given shaping parameter vector σ and a set of scenarios $\mathcal{A}$.

TABLE I

REQUIRED CARD ($\mathcal{A}$) AS A FUNCTION OF THE PRECISION PARAMETER η FOR A CONFIDENCE PARAMETER OF $\delta = 10^{-3}$ AND A NUMBER OF ADMITTED FAILURE = 1

`N_trials`	$\eta = 0.1$	$\eta = 0.05$	$\eta = 0.01$	$\eta = 0.001$
1	132	264	1317	13164
5	154	308	1536	15354
10	163	326	1628	16280
100	193	386	1930	19299
1000	223	445	2225	22249

Unfortunately, this process is computationally very expensive as explained in the next section where an alternative suboptimal but far more tractable is proposed. Notice that the particular role played by C_{RT} in the algorithm is because the underlying monotonicity property needed in the binary research is more solid for this criterion than for the contraction-related one.

D. Suboptimal Tractable Algorithm

The previous formulation requires a huge amount of computation. Indeed for each σ a binary search has to be performed to compute the optimal $\hat{\alpha}(\sigma|\mathcal{A})$ (if any) in which the expressions involved in the constraints (10b) has to be evaluated through m-steps closed-loop simulations and this, for all the scenarios included in the high cardinality set $\mathcal{A}$ (see Table I). Assuming a precision of ε on α, this induces a *worst case* number of optimal control problems that compares to

$$\mathtt{N\_trials} \times \mathrm{card}(\mathcal{A}) \times m \times \log(1/\varepsilon). \tag{11}$$

Note that $\mathrm{card}(\mathcal{A})$ is determined by the precision ($\eta \in (0, 1)$) and the confidence ($\delta \in (0, 1)$) parameters required for the certification of the success when using the corresponding setting [14] (see Table I).

That is the reason why a suboptimal version of the formulation is adopted in which, the process is split into two subprocesses, namely the following.

1) First, a randomly generated set $\mathcal{A}_0$ containing a reduced number of scenarios, namely $n_0 := \mathrm{card}(\mathcal{A}_0) \ll \mathrm{card}(\mathcal{A})$ is considered for which the previously described optimal formulation is applied to determine $\hat{\alpha}(\sigma^{(j)}|\mathcal{A}_0)$ for $j = 1, \ldots,$ `N_trials`.

2) The so obtained $\hat{\alpha}(\sigma^{(j)}|\mathcal{A}_0)$ are now frozen and the $\sigma^{(j)}$ are individually checked (which means that we can use the first line of Table I) over a sequence of subsets $\mathcal{A}^{[\ell]}$ that forms a partition of the original set $\mathcal{A}$, namely: $\mathcal{A} := \bigcup_{\ell=1}^{n_s} \mathcal{A}^{[\ell]}$ and the associated optimal costs are summed up over the subsets $\mathcal{A}^{[\ell]}$ to ultimately form the corresponding cost $\mathcal{J}(\sigma^{[j]}|\mathcal{A})$.

During this process, as soon as a failure is detected for some $\sigma^{[j]}$ for one of the scenarios of a subset $\mathcal{A}^{[\ell]}$, this $\sigma^{[j]}$ is removed from the set of candidate values and is no more considered for the remaining subsets $\mathcal{A}^{[\ell+\cdot]}$.

This solution hugely reduces the number of evaluations since binary search is restricted to $\mathcal{A}_0$, on the one hand, and the number of uselessly visited evaluations is drastically reduced by progressively removing unsuccessful $\sigma^{[j]}$ after failure on intermediate low cardinality $\mathcal{A}^{[\ell]}$, on the other hand. The counterpart of this simplification is that the frozen values of $\hat{\alpha}(\sigma^{(j)})$ do not take into account all the scenarios of $\mathcal{A}$ and might hence be wrongly biased by the small number n_0 of initial scenarios contained in $\mathcal{A}_0$. This bias might lead to discarding some σ's that would otherwise be admissible.

IV. BRIEF DESCRIPTION OF THE PYTHON-PACKAGE

In this section, a brief description of the `Python` package `mpc_tuner` is provided while a complete description can be found at the author's `GitHub` account (`https://github.com/mazenalamir/MPC_tuner`). To use the package, the user needs to provide an instance `pb` of the `Container` class with the attributes and the methods shown in Fig. 5. This object `pb` gathers the problem-dependent items and is used in the package's utilities. The package contains the following main classes and their main methods (Fig. 5):

(`Class: Sigma`): An instance of this class corresponds to a specific value of the σ vector. This class exports the maps $\kappa(\alpha)$, `N_pred`(α), ..., and so on, which are the maps $\pi_i(\alpha)$ described in (4).

(`Class: MPC`): An instance of this class is a specific MPC setting defined by the triplet (`pb`, σ, α). The main methods of this class are

```
MPC.feedback(x,p,q,z0)
MPC.sim_cl(sc, z0, optim_par)
```

where the `feedback` function delivers the feedback input vector while `sim_cl` simulates the closed-loop associated with the scenario with initial guess z_0 at the initial instant followed by warm start initialization. The computation of the success condition uses the object `optim_par:=(`γ, ε`, imp_acc,c_max)` as shown in Fig. 3. Notice that in the current version of the package, the optimization uses the framework `CasADi` [2] with the solver option `IPOPT` [4] in single-shooting mode. Besides the above classes, the package contains the following main global methods.

(`Function: Generate_A(pb,nb,nsb)`): Which generates a list of `nb` subsets $\mathcal{A}^{[\ell]}$, $\ell = 1, \ldots,$ `nb` of cardinality `nsb` each, leading to a total set of scenarios

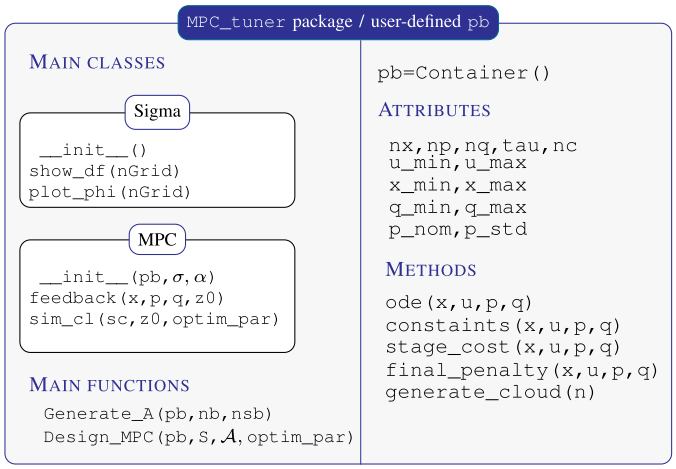


Fig. 5. Summary of the main classes and functions exported by the package and the user-defined object to prepare for a specific control problem.

$\mathcal{A} := \bigcup_{\ell=1}^{\mathtt{nb}} \mathcal{A}^{[\ell]}$ of cardinality $\mathrm{card}(\mathcal{A}) = \mathtt{nb} \times \mathtt{nsb}$. The first of these sets plays the role of the initial set $\mathcal{A}_0 := \mathcal{A}^{[1]}$ invoked above and used to determine the values of $\hat{\alpha}(\sigma^{(j)}|\mathcal{A}_0)$ to be certified using the remaining subsets $\mathcal{A}^{[\ell]}$, for $\ell = 2, \ldots, \mathtt{nb}$. The function `Generate_A(pb,nb,nsb)` calls the method `pb.generate_cloud` of the user-defined object `pb`.

(`Function: Design_MPC(...)`): Which is the main function that enables one to look for a list of admissible NMPC settings following the algorithm described in Section III-D. The call of this main function admits the following input arguments: 1) the user-defined object `pb`; 2) a list `S` of possible values of the shaping parameter σ generated by successive calls (`N_trials`) of instance generation of the class `Sigma` described above; 3) the list of sets $\mathcal{A} := \bigcup_{\ell=1}^{n_s} \mathcal{A}^{[\ell]}$; and 4) the object `optim_par` described above. This function returns a `pandas` data frame representing the set of admissible settings (σ, α) with their associated cumulated closed-loop costs.

V. Illustrative Example

To illustrate the framework and the use of the associated `Python`-based package, let us consider the control of the PVTOL that obeys the following normalized dynamics:

$$\ddot{y} = -u_1 \sin\theta + p_1 u_2 \cos\theta \tag{12a}$$

$$\ddot{z} = u_1 \cos\theta + p_1 u_2 \sin\theta - 1 \tag{12b}$$

$$\ddot{\theta} = p_2 u_2 \tag{12c}$$

which shows a state vector $x := (y, z, \theta, \dot{y}, \dot{z}, \dot{\theta})$ of dimension $n_x = 6$, a control vector of dimension $n_u = 2$. The model depends on the 2-D parameter vector p ($n_p = 2$). The control objective is to regulate around reference values of y and z that are denoted hereafter by q_1 and q_2, respectively (these are the first two components of the context vector q mentioned in the previous sections). This leads to the desired targeted state $x_d := (q_1, q_2, 0, \ldots, 0)$ corresponding to the steady control $u_d := (1, 0)$. Therefore, the economic stage cost is given by $\ell(x, u, p, q) := \|x - x_d\|_Q^2 + \|u - u_d\|_R^2$ with predefined Q and R weighting matrices[3] which are not to be tuned. On the contrary, the terminal penalty function defined by $\Psi(x, p, q) := \rho_f \|x - x_d\|_Q$ does involve the tunable parameter $\pi_5 := \rho_f$ [see (2)]. All the scenarios considered hereafter last a duration of 0.5 time units which induces a different number of optimization instances depending on the values of the parameters that induce different values of τ_u. The contraction rate $\gamma = 0.98$ is used. The precision on α in the binary search is fixed to `optim_par.eps` = 0.15. As for the constraints, besides the input saturation defined by $u \in [-50, +50]^2$, the following two constraints are imposed: $|\dot{\theta}| \leq q_3$ and $|\theta| \leq q_4$ which defines two other components of the context vector q (hence $n_q = 4$) the constraint admissible violation parameter $c_{\max} = 0.1$ is used on the normalized constraints. This defines the constraints map $c(x, u, p, q)$ that encodes $n_c = 4$ constraints. Interested readers can refer to the arXiv version (http://arxiv.org/abs/2309.17238) of the brief to examine the Python file that defines the user-defined items needed for the NMPC design.

TABLE II
Bounds Used on the Design Parameters

Design parameter	min-value	max-value
`N_pred`	5	25
κ	1	10
ρ_f	1	10^3
`rho_cstr`	10^3	10^7
`max_iter`	5	20

The problem-dependent object defined in the user-defined script, namely `pvtol` is imported in the main script shown in Fig. 6. This script starts by using the function `generate_A` to create a set $\mathcal{A}$ of scenarios with cardinality 300 that is decomposed into `nb` = 30 batches of `nsb` = 10 scenarios each. Then, the script creates a set of `N_trials` = 100 candidate shaping parameter vectors $\sigma^{[j]}$, $j = 1, \ldots,$ `N_trials`.

Notice that the chosen values of `nb` and `nsb` induce an initial set of scenarios $\mathcal{A}_0$ with cardinality $n_0 = 10$ and a number of certification scenarios equal to 290 which, according to the first line of Table I, is sufficient to certify the successful design for a precision level of $\eta = 0.05$ and a confidence level of $\delta = 10^{-3}$.

VI. Discussion

Table II shows the default bounds used in the random sampling of the shaping parameter σ when using the instantiation call `Sigma ()`. Fig. 7 shows the tuning results for two different target implementations corresponding to `imp_acc=1` (top) and `imp_acc=2` (bottom) for the same randomly generated set of scenarios $\mathcal{A}$ and candidate set of parameters $\sigma^{(j)}$, $j = 1, \ldots,$ `N_trials`. For each case, two results are shown, namely, the data frame showing the list of admissible settings together with their individual parameters as well as the cumulative closed-loop cost over the scenarios contained in the set $\mathcal{A}$ of 300 scenarios. Below this data frame, the evolution of the number of excluded candidate values of σ among the

[3]In the numerical investigation, $Q = \mathrm{diag}(10^3, 10^3, 10^3, 1, 1, 1)$ and $R = \mathrm{diag}(0.1, 0.1)$.

```
import numpy as np
from MPC_tuner import Design_MPC, Sigma, generate_A, OptimPar
from user_defined_pvtol import pvtol

# Generate the set of scenarios
#------------------------------------------------
# Here, a small number of batch of small size is
# used for the illustration.

nb, nsb = 30, 10
A = generate_A(pvtol, nb, nsb)

# Generate the set of candidate sigma's
#------------------------------------------------

N_trials = 100
S = [Sigma() for _ in range(N_trials)]

# Set the Design meta-parameters and run the Design
#------------------------------------------------
optim_par = OptimPar(gam=0.98, c_max=0.1,
                     imp_acc=1.0, T=0.5)

R_design_log = Design_MPC(pvtol, S, A, optim_par)
```

Fig. 6. Script that runs the MPC design procedure using `N_trials` = 100 candidates σ and a set of scenarios of cardinality card($\mathcal{A}$) = 300 decomposed into batches $\mathcal{A}^{[\ell]}$, $\ell = 1, \ldots, 30$ of cardinality 10 each. Note that each experiment uses a different initial set of 100 candidate configurations.

`imp_acc = 1`

	kappa	N_pred	n_steps	n_ctr	rho_cstr	rho_final	tau	tau_u	max_iter	cost	alpha
0	9	7	3	8	9.646822e+06	178.800537	0.02	0.18	11	7859.487689	0.750
1	4	6	2	4	9.102911e+06	24.259781	0.02	0.08	16	4404.438416	0.625
2	2	5	2	2	7.937212e+06	1.243896	0.02	0.04	8	2437.847089	0.250
3	8	7	5	2	1.250875e+06	2.951172	0.02	0.16	18	3401.306977	0.500
4	9	6	7	7	9.170123e+06	8.804688	0.02	0.18	6	3755.852208	0.500

`imp_acc = 2`

	kappa	N_pred	n_steps	n_ctr	rho_cstr	rho_final	tau	tau_u	max_iter	cost	alpha
0	7	17	4	12	8.409123e+06	16.609375	0.02	0.14	5	3210.056159	0.250
1	9	7	3	8	9.646822e+06	178.800537	0.02	0.18	11	7859.487689	0.750
2	9	20	2	19	8.409123e+06	820.515021	0.02	0.18	8	3766.654430	0.250
3	6	17	3	16	6.124112e+06	3.778133	0.02	0.12	5	3284.618189	0.375
4	4	6	2	4	9.102911e+06	24.259781	0.02	0.08	16	4404.438416	0.625
5	9	5	8	4	6.259375e+05	2.951172	0.02	0.18	18	3126.158301	0.500
6	8	12	2	11	7.515030e+04	1.054939	0.02	0.16	17	2891.563097	0.375
7	7	9	5	2	2.442162e+06	15.537363	0.02	0.14	19	3583.879784	0.625
8	9	5	8	6	9.350676e+06	38.215650	0.02	0.18	18	4611.172897	0.625
9	2	12	2	2	8.846257e+06	1.146503	0.02	0.04	7	3408.564392	0.375
10	6	6	6	2	5.282910e+05	1.146503	0.02	0.12	18	3355.846137	0.375
11	9	18	2	13	3.536180e+06	125.875000	0.02	0.18	12	3250.814430	0.125
12	6	19	2	2	7.430228e+06	2.951172	0.02	0.12	14	3360.845813	0.125

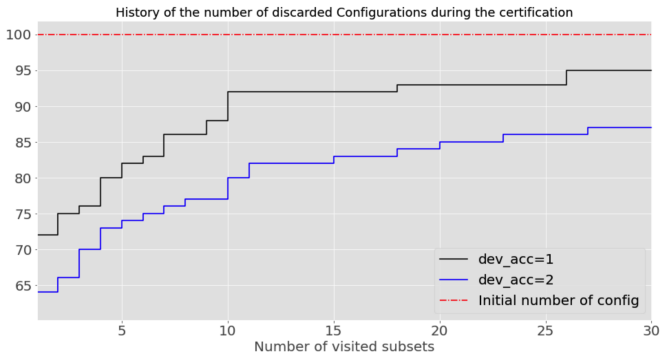


Fig. 7. Two instances of results given by the function `Design_MPC` for different values of the `imp_acc` input for the algorithm, namely, `imp_acc` = 1 (upper items) and `imp_acc` = 2 (lower items). In both cases, the configurations are sampled inside the domains defined by the bounds shown in Table II.

`N_trials` = 100 randomly sampled design configurations is shown as a function of the number of already executed batches corresponding to the subsets of scenarios $\mathcal{A}^{[\ell]}$. Regarding the results, the following observations are worth making.

1) *Only a Small Portion of Configurations Is Eligible:* Despite the quite reasonable bounds given in Table II, it is quite remarkable that only a small portion of the randomly sampled settings are admissible (5% for a computation target given by `imp_acc=1`) and (13% for a computation target given by `imp_acc=2`). This simple fact suggests that the problem addressed here is quite relevant.
2) *High Values of ρ_f Lead to Infeasibility:* The results suggest that high values of ρ_f lead to inadmissible settings and/or high values of the cost. Recall that the reported closed-loop costs do not incorporate the terminal penalty as the latter is generally used for stability reasons. However, this cost includes the possible nonvanishing terms coming from the constraint being violated by less than the authorized threshold $c_{\max} \neq 0$. The results suggest that too high a terminal penalty might lead to bad performance when the number of iterations is limited. This is intuitively sound because in this case, the problem is stiffer and the step size is consequently small.
3) *High Values of `rho_cstr` Are Needed:* On the contrary, the high majority of admissible values of soft constraints penalty `rho_cstr` seem to lie exclusively close to the upper bound of the admissible interval $[10^3, 10^7]$. This is intuitively quite tricky to guess and enforces if still needed, the relevance of the problem addressed in this contribution. This also suggests that it might be interesting to re-run the algorithm with the bounds of `rho_cstr` shifted toward higher values, for instance, $[10^5, 10^9]$ following the conjecture according to which, given the random sampling, too many sampled settings fail in meeting the constraints with low values of this parameter which penalizes the randomly generated set of configurations.
4) *Scenarios Are Constraints-Challenging:* The previous fact also suggests that the set of scenarios used in the certification does involve constraint-violation-risky initial conditions that have been managed using high penalty on the soft constraints; otherwise, the random sampling would have provided successful settings with lower values of ρ_{constr} since the latter would have not affected the success/failure status.
5) *Large Eligible Possibilities for the Prediction Horizon:* Notice that among the set of admissible settings, the prediction horizon lengths take values from 0.2 up to 1.26 when `imp_acc`= 1 and from 0.48 to 3.24 when `imp_acc` = 2. This can be explained by the definition of $\hat{\alpha}(\sigma|\mathcal{A})$ being the maximum allowable value since this definition enhances longer prediction horizons `N_pred` × `tau_u`. The final choice of the NMPC

design might favor not too short prediction horizons for obvious reasons even if this corresponds to slightly higher closed-loop since the connection between the truly obtained closed-loop performances and the ones predicted on the short-term simulation used in the algorithm is not so strong.

The higher the cardinality of the set of candidates σ is, the more exhaustive the resulting list of admissible configurations. The computation times for the design examples are, respectively, 62 and 97 min on a `MacBook Pro, 2.4 GHz Intel Core i9`. Keep in mind, however, that the computation time depends on the randomly sampled design settings. The call of `Ipopt` uses the default setting.

VII. Conclusion and Ongoing Investigation

In this brief, a systematic approach and an associated freely available (`MPC_tuner`) `Python` package are proposed for the design of the implementation parameters of an NMPC controller. Despite encouraging preliminary results, it might be conjectured that additional/different tricks might be used to accelerate the computation and or reduce the level of suboptimality. One option would be to use machine learning tools to derive preliminary feasibility predictors that can be trained over a cloud of (σ, α) without the binary search used in the first step which greatly impacts the computation time, the resulting model can then be used to make better guesses reducing hence the number of useless randomly generated samples.

On the other hand, future inclusion of solvers in the `CasADi` framework can immediately be included in the search instead of the only `IPopT` [4] that is currently used by default. As possible options, it is possible to add different packages of modeling framework for optimization problems (like `Acados` [5], `Gekko` [15], and `Pyomo` [3]) even if unavoidable compatibility issues would force the user to switch between different syntax in the definition of the problem-dependent file. For the time being, recall that any other implementation conditions (device, algorithm) can be examined through the current `mpc_tuner` through the parameter `imp_acc`.

References

[1] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model Predictive Control: Theory, Computation, and Design*, vol. 2. Madison, WI, USA: Nob Hill Publishing, 2017.

[2] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi: A software framework for nonlinear optimization and optimal control," *Math. Program. Comput.*, vol. 11, no. 1, pp. 1–36, Mar. 2019.

[3] W. E. Hart et al., *Pyomo—Optimization Modeling in Python*, vol. 67. Springer, 2017.

[4] L. T. Biegler and V. M. Zavala, "Large-scale nonlinear programming using IPOPT: An integrating framework for enterprise-wide dynamic optimization," *Comput. Chem. Eng.*, vol. 33, no. 3, pp. 575–582, Mar. 2009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0098135408001646

[5] B. Houska, H. J. Ferreau, and M. Diehl, "ACADO toolkit—An open-source framework for automatic control and dynamic optimization," *Optim. Control Appl. Methods*, vol. 32, no. 3, pp. 298–312, May 2011.

[6] V. Ramasamy et al., "A comprehensive review on advanced process control of cement kiln process with the focus on MPC tuning strategies," *J. Process Control*, vol. 121, pp. 85–102, Jan. 2023.

[7] E. Bøhn, S. Gros, S. Moe, and T. A. Johansen, "Optimization of the model predictive control meta-parameters through reinforcement learning," *Eng. Appl. Artif. Intell.*, vol. 123, Aug. 2023, Art. no. 106211.

[8] F. Sorourifar, G. Makrygirgos, A. Mesbah, and J. A. Paulson, "A data-driven automatic tuning method for MPC under uncertainty using constrained Bayesian optimization," *IFAC-PapersOnLine*, vol. 54, no. 3, pp. 243–250, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896321010223

[9] A. Mohammadi, H. Asadi, S. Mohamed, K. Nelson, and S. Nahavandi, "Multiobjective and interactive genetic algorithms for weight tuning of a model predictive control-based motion cueing algorithm," *IEEE Trans. Cybern.*, vol. 49, no. 9, pp. 3471–3481, Sep. 2019.

[10] M. Neumann-Brosig, A. Marco, D. Schwarzmann, and S. Trimpe, "Data-efficient autotuning with Bayesian optimization: An industrial control study," *IEEE Trans. Control Syst. Technol.*, vol. 28, no. 3, pp. 730–740, May 2020.

[11] Q. Lu, R. Kumar, and V. M. Zavala, "MPC controller tuning using Bayesian optimization techniques," 2020, *arXiv:2009.14175*.

[12] D. Stenger, M. Ay, and D. Abel, "Robust parametrization of a model predictive controller for a CNC machining center using Bayesian optimization," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 10388–10394, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896320335412

[13] M. Alamir, "Contraction-based nonlinear model predictive control formulation without stability-related terminal constraints," *Automatica*, vol. 75, pp. 288–292, Jan. 2017.

[14] T. Alamo, R. Tempo, and E. F. Camacho, "Randomized strategies for probabilistic solutions of uncertain feasibility and optimization problems," *IEEE Trans. Autom. Control*, vol. 54, no. 11, pp. 2545–2559, Nov. 2009.

[15] L. Beal, D. Hill, R. Martin, and J. Hedengren, "GEKKO optimization suite," *Processes*, vol. 6, no. 8, p. 106, 2018.