Fast NMPC: A Reality-Steered Paradigm: Key Properties of Fast NMPC Algorithms

Mazen Alamir[1]

Abstract— **In this paper, the paradigm of fast Nonlinear Model Predictive Control is recalled. Then a fundamental inequality that conditions the closed-loop stability is derived. Based on this inequality, it is shown that the comparison between different algorithms in the context of Fast NMPC must be based not only on the efficiency per iteration but also on the time needed to perform a single iteration. An illustrative example is used to assess the fact that under some circumstances, it is worth using less efficient algorithms (in conventional sense) that correspond to less amount of computation per iteration and this even when perfect model is used and in a disturbance-free context.**

I. INTRODUCTION

Fast Nonlinear Model Predictive Control [10], [4], [6], [5], [1] emerged from the need to extend the use of NMPC to systems that show fast dynamics. These are systems that can not be left in open-loop more than some maximum updating period τ_u^{max} which is too short to completely solve the underlying nonlinear programming problem (NLP). The use of sub-optimal solutions invalidates one of the main assumptions that assess the stability of the closed-loop system, namely the one stating that:

> After shifting the horizon, there is an easy computable hot-start that decreases the cost function.

When this property is not satisfied, each time the horizon recedes, there is a potential increase in the value of the cost function. Moreover, this positive increment increases with the amount of horizon shift (the control updating period). It results that the evolution of the cost function is the consequence of two opposite effects: a potentially positive increment that is due to the horizon's shift (under non optimal current solution) and the negative increment that is induced by the optimization process.

In the present paper, this trade-off is studied more rigorously by underlying two key properties of optimization algorithms, namely the efficiency map and the time needed for a single iteration. It is then shown, first intuitively using generic illustrations and then by a specific numerical example that it is sometimes better to use a less efficient optimizer provided that it offers significantly less expensive iterations.

Note that unlike the papers [2], [3] where updating mechanisms are suggested to monitor the updating period for a given solver, the present work involves a comparison between candidate algorithms that show different iteration complexity and analyzes the impact of their use on the resulting closed-loop performance.

The paper is organized as follows: After some definitions and notation are introduced in section II, section III discusses the conditions under which a decrease in the cost function is obtained. This analysis is then used in section III to establish the main claim of the paper according to which, it is sometimes better to use less efficient (by iteration) algorithms provided that a significant gain in the computation time of the iteration is obtained. A complexity-scalable algorithm is proposed in section V that can yield different iteration-related computational burdens according to the choice of the algorithms parameters. Two instantiations of the algorithms are then used in section VI to illustrate the main result of the paper on a concrete example.

II. DEFINITIONS AND NOTATION

Consider a general nonlinear system that is governed by the following equation:

$$x(t+\tau) := F(\tau, x(t), \tilde{u}(\cdot)) \tag{1}$$

where $x \in \mathbb{R}^n$ and $u \in \mathbb{R}^m$ stand for the state and the control vectors respectively. $\tilde{u}(\cdot)$ is a control profile that is defined over the time interval $[0, \tau]$. When the control profile is parametrized by some parameter vector p such that:

$$u(\sigma) := \mathcal{U}(\sigma, p(t)) \tag{2}$$

the following notation is also used overloading the definition of the map F without ambiguity:

$$x(t+\tau) = F(\tau, x(t), p(t)) \tag{3}$$

It is assumed that an MPC strategy is used through the following open-loop optimization problem:

$$\mathcal{P}(x(t)) : \min_{p\in\mathbb{P}} J(x(t), p) \quad \text{under } g(x(t), p) \le 0 \tag{4}$$

where $\mathbb{P} \in \mathbb{R}^{n_p}$ is a hypercube while J and g are scalar functions denoting the cost and the constraint functions respectively. Note that a scalar function g can represent a set of constraints through the $\max$ operator.

In order to solve the nonlinear programming problem (4), an iterative solver is used that can be denoted by:

$$p^{(q)} = \mathcal{S}^{(q)}\big(p^{(0)}, x(t)\big) \tag{5}$$

[1]Mazen Alamir is CNRS Research Director at Gipsa-lab, Control Systems Department, University of Grenoble. 11 rue des Mathmatiques, Domaine Universitaire, Saint-Martin d'Hres, France.

where $p^{(0)}$ is the initial guess while $p^{(q)}$ is the iterate that is obtained after q successive iterations of the solver. In this paper, we mean by *an iteration* the unbreakable set of operations that would be needed to deliver an update of the decision variable value. It is also assumed that the time $\tau_1^{\mathcal{S}}$ needed to perform a single iteration is constant for a given hardware. This means for instance that a computation time lower than $\tau_1^{\mathcal{S}}$ does not enable an update of the decision variable. That is the reason why, a typical real-time MPC implementation is defined by a control updating period which is a multiple of $\tau_1^{\mathcal{S}}$, namely $\tau_u := q\tau_1^{\mathcal{S}}$. This results in an implementation that can be described as follows:
✓ The control updating instants t_k are defined by

$$t_k = t_{k+1} + q\tau_1^{\mathcal{S}} \tag{6}$$

✓ The control profile that is applied over $[t_{k-1}, t_k]$ is the one given by $u(t_{k-1}+\sigma) := \mathcal{U}(\sigma, p(t_{k-1}))$ for $\sigma \in [0, t_k - t_{k-1}]$.
✓ During the sampling period $[t_{k-1}, t_k]$, q iterations are computed starting from the initial guess $p^*(t_{k-1})$ which is the hot start corresponding to the previous value $p(t_{k-1})$. Note that in a standard piecewise constant parametrization, one has:

$$p(t_{k-1}) := \big(u^{(0)}, u^{(1)}, \dots, u^{(N-2)}, u^{(N-1)}\big)$$

a typical hot start is defined by

$$p^*(t_{k-1}) := (u^{(1)}, u^{(2)}, \dots, u^{(N-1)}, u^{(N)}) \tag{7}$$

where $u^{(N)}$ is appropriately chosen (see [9] for more details). Based on the discussion above, the updating law for the control parameter p is therefore given by:

$$p(t_k) := \mathcal{S}^{(q)}\big(p^*(t_{k-1}), x(t_k)\big) \tag{8}$$

Consequently, there is an extended system with the state vector $z := (x, p)$ that is governed by the following dynamic equations:

$$\begin{aligned} x(t_{k+1}) &= F(\sigma_k, x(t_k), p(t_k)) && (9)\\ p(t_{k+1}) &= \mathcal{S}^{(q)}(p^*(t_k), x(t_{k+1})) && (10) \end{aligned}$$

where $\sigma_k := t_{k+1} - t_k$.

III. Stability Analysis

The stability of the extended system (9)-(10) depends on the decrease of the cost function J, namely the sign of the difference:

$$\Delta J(t_k) \;:=\; J(x(t_k), p(t_k)) - J(x(t_{k-1}), p(t_{k-1}))$$

Indeed, if it can be guaranteed that $\Delta J(t_k)$ is always negative then stability results under mild conditions, otherwise there is risk of instability. Now the main argument that enables stability to be proved in classical NMPC schemes (see [9]) comes from the inequality stating that the hot start decreases the cost function, namely:

$$\begin{aligned} D_k(\tau_u) &:= J(x(t_k), p^*(t_{k-1})) - J(x(t_{k-1}), p(t_{k-1}))\\ &\le 0 \end{aligned} \tag{11}$$

Note that the term $D_k(\tau_u)$ depends on the current instant t_k and the value of the control updating period τ_u as it is the amount of shift that is applied to the prediction horizon before a hot start is used as initial guess for the new optimization problem. Note also that one obviously has $D_k(0) = 0$.

This term appears when the expression of $\Delta J(t_k)$ is decomposed according to:

$$\begin{aligned} \Delta J(t_k) &= J(x(t_k), p(t_k)) - J(x(t_k), p^*(t_{k-1})) + \dots\\ &+ J(x(t_k), p^*(t_{k-1})) - J(x(t_{k-1}), p(t_{k-1})\\ &= -E_k^{\mathcal{S}}(\tau_u) + D_k(\tau_u) \end{aligned}$$

where $E_k^{\mathcal{S}}(\tau_u)$ is the drop in the cost function between the initial guess and after $q := \lfloor \tau_u/\tau_1^{\mathcal{S}} \rfloor$ iterations in the available control updating period τ_u.
namely:

$$E_k^{\mathcal{S}}(\tau_u) := J(x(t_k), p^*(t_{k-1})) - J(x(t_k), p(t_k)) \ge 0 \tag{12}$$

This is because as mentioned earlier, $p^*(t_{k-1})$ is used as initial guess in the optimization problem leading to the updating value $p(t_k)$ after q iterations:

$$p(t_k) := p^{(q)} := \mathcal{S}^{(q)}(p^*(t_{k-1}), x(t_k)) \tag{13}$$

which implies that the final value $J(x(t_k), p(t_k))$ is always lower than the initial value $J(x(t_k), p^*(t_{k-1}))$.

The assumption (11) that underlines the stability proof in the ideal case requires the satisfaction of two conditions [9]:

1) The inclusion of a final constraint on the state trajectory. This final constraint imposes that the state at the end of the prediction horizon lies in a subset $\mathcal{X}$ of the state space that is both invariant under some local controller and such that the penalty that defines the cost function is a control Lyapunov function w.r.t the local control law.

2) The parameter value $p(t_k)$ is sufficiently good to induce a cost decreasing hot start.

The stability issue in Fast NMPC implementations comes from two distinct reasons that correspond each to a violation of one of the two above mentioned conditions:

✓ Indeed the computation of the final invariant set is not a straightforward task for realistic nonlinear systems. Moreover, even when it is possible, practitioners do not always include the corresponding constraint in the NMPC formulation because it leads to more involved NLP problems and can induce a potentially useless drop in performance.

✓ Even when the constraint is included in the NMPC formulation, the fact that only a limited number of iterations q is available during the control updating period results in a current value $p(t_k)$ that does not necessarily lead to a state trajectory that meets this constraint.

The consequence of the above discussion can be summarized as follows:

Fact *1:* In realistic fast NMPC implementations, there is no guarantee that the term $D_k(\tau_u)$ is negative. Therefore, the decrease of the cost function depends on the sign of the term:

$$\Delta J(t_k) := -E_k^{\mathcal{S}}(\tau_u) + D_k(\tau_u) \tag{14}$$

that depends on both the control updating period τ_u and the solver $\mathcal{S}$ through the time $\tau_1^{\mathcal{S}}$ needed for a single iteration.

IV. Qualitative Discussion

The key expression (14) involves the two quantities $D(\tau_u)$ and $E^{\mathcal{S}}(\tau_u)$ that show the following properties:

- ✓ $D(0) = E^{\mathcal{S}}(0) = 0$.
- ✓ $E^{\mathcal{S}}(\tau_u)$ is necessarily ≥ 0
- ✓ $D(\tau_u)$ can incidently be positive (which is the case we are interested in).
- ✓ $D(\tau_u)$ is defined continuously in τ_u and is independent of the solver. It depends however on the problem formulation (control parametrization, length of prediction horizon, cost function, constraints, definition of the hot start, etc.)
- ✓ $E^{\mathcal{S}}(\tau_u)$ is defined only on the quantized set defined by:

$$\mathcal{T} := \{1, 2, \dots, \} \times \tau_1^{\mathcal{S}} \tag{15}$$

since $\tau_1^{\mathcal{S}}$ is supposed to be the incompressible time needed to perform a single iteration of the optimization method $\mathcal{S}$.

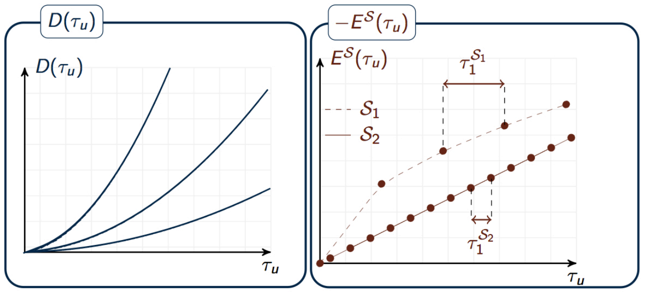

Fig. 1. Schematic view of potential allures of the maps $D(\tau_u)$ and $E^{\mathcal{S}}(\tau_u)$ in realistic fast NMPC implementations. The right figure shows the efficiency map for two different solvers corresponding to two different computation times per iteration $\tau_1^{\mathcal{S}_1}$ and $\tau_1^{\mathcal{S}_2}$

Figure 1 shows typical allures of these two maps. As far as the efficiency map is concerned, two optimization methods are considered, namely $\mathcal{S}_1$ and $\mathcal{S}_2$. The figure suggests that $\mathcal{S}_1$ is more efficient per iteration (the first single iteration of $\mathcal{S}_1$ performs a decrease in the cost function that would need 8 iterations of $\mathcal{S}_2$). However this efficient iteration needs more time $\tau_1^{\mathcal{S}_1}$ to be performed than the one needed to perform a single iteration of $\mathcal{S}_2$, namely $\tau_1^{\mathcal{S}_2}$.

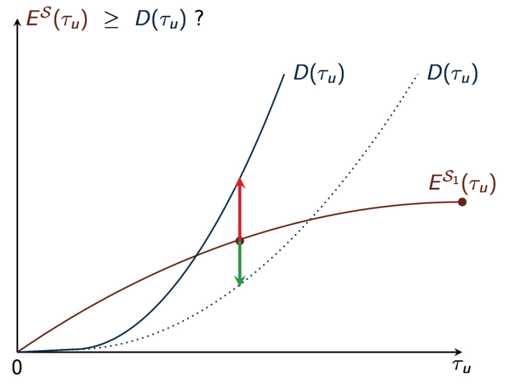

Fig. 2. Analysis of the decrease of the cost function when $\mathcal{S}_1$ is used for two different instantiations of the map $D(\tau_u)$. Note that when the highest instantiation occurs, there is no way to decrease the cost function using the optimization method $\mathcal{S}_1$.

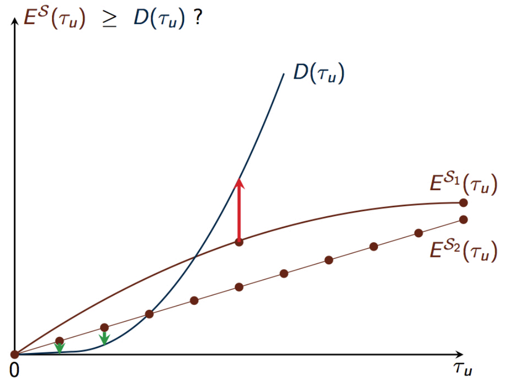

Fig. 3. Analysis of the decrease of the cost function for two different optimization methods. Note that there is no way to decrease the cost function using the optimization method $\mathcal{S}_1$ while the use of 1 or 2 iterations of $\mathcal{S}_2$ would decrease the cost function.

Figure 2 depicts the situation that may be encountered when the optimization method $\mathcal{S}_1$ is used. Two instantiations of the map $D(\tau_u)$ are considered. Note that when the highest instantiation is encountered, there is no way to decrease the cost function using the optimization method $\mathcal{S}_1$.

Figure 3 shows that in this case, the use of one or two iterations of the optimization method $\mathcal{S}_2$ would decrease the cost function. This qualitatively proves the following fact:

Fact *2:* Under certain circumstances, it is worth using less efficient solver provided that it corresponds to shorter computation time by iteration.

This fact motivates the emergence of (fast) gradient-based implementation of MPC even in the linear system framework (see [8] and the references therein). The present paper gives a deeper understanding of the question.

In section VI, a concrete illustrative example showing the relevance of the qualitative discussion of the present section is given. Before this example is given, a complexity-scalable optimization method is first described in section V. The resulting method defines a family of solvers each of which can be obtained for a specific choice of the method's parameters. Two specific settings are then used to illustrate Fact 2 which is the main message of the present paper.

V. A Successive Partial Gradient-Based Scalar Quadratic Programming Algorithm For Constrained Multivariable NLP Problems

Let us consider a general multi-variable constrained optimization problem of the following form:

$$\min_{p\in\mathbb{P}} J(p) \quad \text{under } g(p)\le 0 \tag{16}$$

where J and g are two scalar functions of the decision variable $p\in\mathbb{P}$ that belongs to a hypercube of the form $\mathbb{P} := \Pi_{i=1}^{n_p}[p_i^{min}, p_i^{max}]$. Note that this problem takes the same form as (4) except that the state has been omitted for simplicity. Recall also that the scalar inequality $g(p)\le 0$ can express a set of inequalities through the max operator.

A. Overview of The Algorithm

The optimization algorithm can be summarized by the following steps where $p^{(i)}$ stands for the current update:

1. Compute a search direction d [see section V-B]
2. Compute **scalar** quadratic approximations of J and g over the line defined by d, namely:
$$\hat{J}(s)\approx J(p^{(i)}+s\cdot d)\ ;\ \hat{g}(s)\approx g(p^{(i)}+s\cdot d) \tag{17}$$
this is done by performing two simulations of the system for the arguments $p=p^{(i)}+\dfrac{\alpha}{2}d$ and $p=p^{(i)}+\alpha d$ where $\alpha\in\mathbb{R}_+$ is a trust region size.
3. Solve the **scalar** QP defined by:
$$s^{cand} := \arg min_{s\in[0,\alpha]}\hat{J}(s) \quad \text{under } \hat{g}(s)\le 0 \tag{18}$$
This problem is solved in standard way when it is feasible (that is $\min_{s\in[0,\alpha]}\hat{g}(s)\le 0$). Otherwise, s^{cand} is chosen so as to minimize $\hat{g}(s)$ over $[0,\alpha]$.
4. At the candidate value $p^{cand} := p^{(i)}+s^{cand}\cdot d$, compute $J(p^{cand})$ and $g(p^{cand})$ by simulating the system once again.
5. If p^{cand} is successful then $p^{(i+1)}=p^{cand}$ and the trust region size is increase ($\alpha\leftarrow\beta^{(+)}\alpha$ with $\beta^{(+)}>1$). Otherwise, $p^{(i+1)}\leftarrow p^{(i)}$ and the trust region size is reduced ($\alpha\leftarrow\beta^{(-)}\alpha$ with $\beta^{(-)}<1$)
6. Goto step 1.

B. Definition of The Search Direction

In what follows, the concept of partial gradients of a scalar function $f:\mathbb{R}^{n_p}\to\mathbb{R}$ is defined as follows: Given any subset of indices $I\subset\{1,\dots,n_p\}$, the partial gradient of f over I is defined by:

$$G^{(f,I)}:\qquad \mathbb{R}^{n_p}\to\mathbb{R}^{n_p}\quad \text{s.t.}\quad \forall i\in\{1,\dots,n_p\}$$
$$G_i^{(f,I)}(p) := \begin{cases} \dfrac{\partial f}{\partial p_i}(p) & \text{if } i\in I\\ 0 & \text{otherwise}\end{cases} \tag{19}$$

Note that when $I=\{1,\dots,n_p\}$, the standard gradient definition is obtained. Otherwise, the partial gradient can be invoked when only increments on the components of p with indices in I are investigated.

Remark *1:* In what follows, the search direction is defined using the standard gradient notation in order to simplify the expressions. It is worth to keep in mind that everywhere the notation $G^{(J)}$ and $G^{(g)}$ is used, one can also use $G^{(J,I)}$ and $G^{(g,I)}$ as soon as a subset of indices is defined.

The rational behind the definition of the search direction d involved in the algorithm's description (section V-A) can be simply described as follows:

- ✓ If $g(p)>0$, then the search direction is simply $-G^{(g)}(p)$.
- ✓ If $(g(p)<-\varepsilon)$ then the search direction is simply $-G^{(J)}(p)$.
- ✓ Otherwise, the search direction is the one that decreases the cost without increasing the constraint function g.

This results in the following definition of the search direction:

$$d(p) := \begin{cases} -G^{(J)}(p) & \text{if } g(p)<-\varepsilon\\ -G^{(g)}(p) & \text{if } g(p)>0\\ -G^{(J)}(p)+\phi(p)G^{(g)}(p) & \text{otherwise}\end{cases} \tag{20}$$

where

$$\phi(p) := \frac{\min\{0,[G^{(J)}(p)]^T[G^{(g)}(p)]\}}{\varepsilon+\|G^{(g)}(p)\|^2} \tag{21}$$

Based on remark 1, equations (20)-(21) may be used to define partial gradient based search direction $d^{(I)}(p)$ for a give subset of indices I. For this, each $G^{(J)}(p)$ and $G^{(g)}(p)$ must be simply replaced by $G^{(J,I)}(p)$ and $G^{(g,I)}(p)$ respectively.

Using the definiton of $d(p)$ it can be shown that the differential system $\dot{p}=d(p)$ with the full set of indices I leads to a stationary point p^∞ that is of one of the two following types:

- Either $G^{(J)}(p^\infty)=0$ and $g(p^\infty)<0$ (local unconstrained minimum), or
- $[G^{(J)}(p^\infty)]=-\gamma\cdot[G^{(g)}(p^\infty)]+O(\varepsilon)$ are and $g(p^\infty)\in[-\varepsilon,0]$. This means that p^∞ is asymptotically (when $\varepsilon\to 0$) a solution of the KKT conditions.

C. The Possible Family of Solvers

The concept of partial gradient and the corresponding definition of the search direction $d^{(I)}(p)$ enables a family of solvers with different computation cost per iteration to be derived from the algorithm described in the preceding sections. Indeed:

- ✓ Assume that a partition of the set of indices $\{1,\dots,n_p\}$ is defined through n_b subsets $I_\sigma\subset\{1,\dots,n_p\}$ s.t:

$$\bigcup_{\sigma=1}^{n_b} I_\sigma = \{1,\dots,n_p\} \tag{22}$$

✓ The algorithm described in section V-A can then be applied using successively the search directions $d^{(I_\sigma)}$, for $\sigma = 1, \dots, n_b$. At the end of each search step (for a given value of σ) the iterate $p^{(i+1)}$ is updated, σ is incremented (odulo n_b) and the operation is repeated.

Note that by doing so, the cost of the iteration for a given $\sigma \in \{1, \dots, n_b\}$ is the one associated to

$$card(I_\sigma) + 3 \quad \text{system simulations}$$

since $card(I_\sigma)$ simulations are needed to compute the partial gradients $G^{(J,I_\sigma)}$ and $G^{(g,I_\sigma)}$ to which three simulations have to be added in order to implement the algorithm described in section V-A [2 simulations at step 2. and 1 simulation at step 4.)

Note also that since all the components are visited during the above mentioned cyclic definition of the partial gradient, the conclusion regarding the stationarity of the resulting p^∞ holds for the partial gradient version of the algorithm.

In the following section, an illustrative example is used that involves the two following solvers:

1) **Solver $\mathcal{S}_1$:** This solver is the full gradient version which is defined by the a single partition defined by $n_b = 1$ and $I_1 = \{1, \dots, n_p\}$. This cost $n_p + 3$ simulations per iteration.

2) **Solver $\mathcal{S}_2$:** This solver is defined by the partition corresponding to the choices $n_b = n_p$ and $I_\sigma = \sigma$ which costs 4 simulations per iteration.

VI. Illustrative Example

Let us consider the following nonlinear system studied in [7]:

$$\dot{x}_1 = x_2 \tag{23}$$
$$\dot{x}_2 = u \tag{24}$$
$$\dot{x}_3 = x_4 \tag{25}$$
$$\dot{x}_4 = -g\sin(x_3) - u\cos(x_3) - bx_4 \tag{26}$$

where the control is restricted to the interval $[-1, +1]$. The control objective is to asymptotically stabilize the origin $x = 0$. To achieve this task, a nonlinear model predictive control scheme is used with the following cost function:

$$\sum_{i=1}^{N} \|x(k+i)\|_Q^2 + \|u(k+i)\|_R^2 \tag{27}$$

where the weighting matrices $Q = \mathbb{I}$ and $R = 0.001$ are used. Note that similarly to [7], no final stability-related constraint is used following the remark regarding realistic NMPC practice.

The control parametrization is defined as follows: a piece-wise constant control profile with a basic sampling period of $\tau = 0.15$ seconds is defined over a prediction horizon of 3 seconds. This leads to $N = 20$. However, linear interpolation is used to reduce the number of decision variables to $n_p = 10$. This means that only $u(0)$, $u(2\tau)$, $u(4\tau)$, ...etc. are free while intermediate values are obtained by linear interpolation.

Based on the definition of the two solvers $\mathcal{S}_1$ and $\mathcal{S}_2$ defined at the end of section V-C, the number of simulations needed for a single iteration of $\mathcal{S}_1$ and $\mathcal{S}_2$ are respectively 13 and 4. Therefore, one can write:

$$\tau_1^{\mathcal{S}_2} = \frac{4}{13} \times \tau_1^{\mathcal{S}_1} \tag{28}$$

Figure 4 shows typical behavior of the resulting closed-loop trajectories. Precise comparison of the closed-loop performances are given later on.

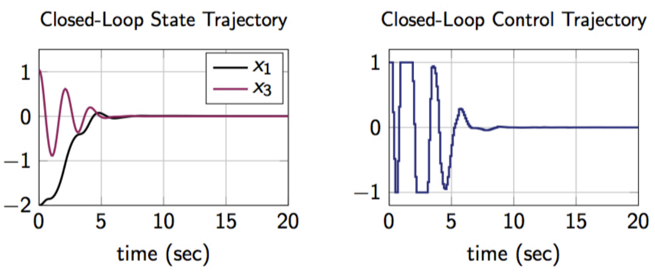

Fig. 4. Typical behavior of the closed-loop trajectories

Three closed-loop simulations are performed for each of the two solvers $\mathcal{S}_1$ and $\mathcal{S}_2$. These simulations differ by the values of the control updating period being used. More precisely, the following settings are used:

$$\tau_u^{(2)} = \frac{4}{13} \times \tau_u^{(1)} \quad , \quad \tau_u^{(1)} \in \{100, 50, 5\} msec \tag{29}$$

This may be viewed as resulting from the use of different hardware leading to different computational speeds.

The following numerical investigations are intended to assess the following facts:

✓ The full gradient algorithm $\mathcal{S}_1$ is more efficient than the partial gradient algorithm $\mathcal{S}_2$ on a static problem (fixed value of x). This is shown in Figure 5.

✓ When considering the on-line evolution of the cost function $J(t_k)$, the closed-loop behavior under the less efficient (by iteration) algorithm $\mathcal{S}_2$ shows better performance than the one corresponding to the full gradient version $\mathcal{S}_1$. This can be shown on Figure 6.

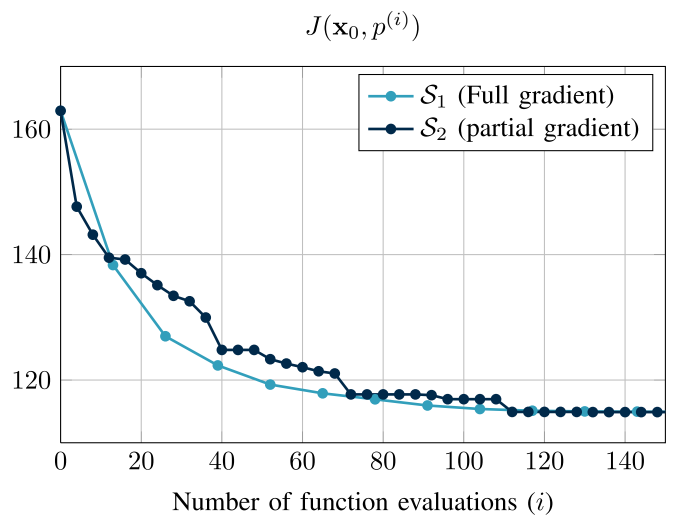

Fig. 5. Static solution of a given optimization problem (for fixed state x_0) using $\mathcal{S}_1$ and $\mathcal{S}_2$. Decrease of the cost function as a function of the number of simulations. This clearly shows that the full gradient algorithm $\mathcal{S}_1$ is more efficient (by iteration) than the partial gradient algorithm $\mathcal{S}_2$.

Note that the positivity of the term $D(\tau_u)$ can be observed on the evolution of the closed-loop system under $\mathcal{S}_1$ as the predicted cost is not monotonically decreasing.

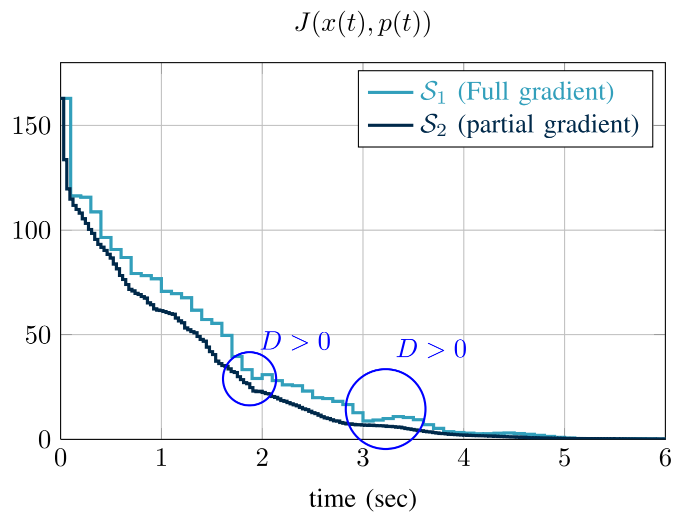

Fig. 6. Example of on-line evolution of the predicted cost $J(x(t), p(t))$ using the full gradient algorithm $\mathcal{S}_1$ and the partial gradient algorithm $\mathcal{S}_2$. Case where $\tau_u^{(1)} = 100$ msec.

Finally, Table I shows comparison of the effectively obtained cost defined by:

$$\frac{1}{N_{sim}} \sum_{i=1}^{N_{sim}} \|x(i)\|_Q^2 + \|u(i)\|_R^2 \tag{30}$$

for the six closed-loop simulation scenarios. It comes out that the cost is better when high frequency updating is enabled

$\tau_u^{(1)}$	100 ms	50 ms	5 ms
Full Gradient	0.728	0.430	0.027
Partial Gradient	0.300	0.110	0.008
Gain %	59%	74%	70%

TABLE I
COMPARAISON BETWEEN THE EFFECTIVE COST UNDER THE SOLVERS $\mathcal{S}_1$ AND $\mathcal{S}_2$ AND FOR DIFFERENT CONTROL UPDATING PERIODS $\tau_u^{(1)}$ USED FOR $\mathcal{S}_1$.

by the hardware for a given solver (horizontal comparison). More interestingly, the table clearly shows that the less efficient iteration $\mathcal{S}_2$ (with higher updating frequency) systematically outperforms $\mathcal{S}_1$ by an amount that can reach 75% (vertical comparison).

VII. Conclusion

In this paper, the paradigm of fast NMPC is recalled and the success conditions of the closed-loop control are analyzed. In particular, it has been shown that beside the efficiency per iteration property, it is crucial to monitor the unbreakable amount of computation that is needed to perform a single iteration. A qualitative analysis has been proposed which suggests that sometimes, it is worth using less efficient iteration provided that it corresponds to significantly less amount of computation.

References

[1] M. Alamir. *Stabilization of nonlinear systems using receding-horizon control schemes: A parameterized approach for fast systems*. Springer-Verlag, 2006.
[2] M. Alamir. A framework for monitoring control updating period in real-time NMPC schemes. In Lalo Magni, DavideMartino Raimondo, and Frank Allgower, editors, *Nonlinear Model Predictive Control*, volume 384 of *Lecture Notes in Control and Information Sciences*, pages 433–445. Springer Berlin Heidelberg, 2009.
[3] M. Alamir. Monitoring control updating period in fast gradient-based NMPC. proceedings of the european control conference, zurich, 2013.
[4] M. Alamir and F. Allgower (Eds). Special issue on fast model predictive control. *International Journal of Robust and Nonlinear Control*, 18(8), 2008.
[5] M. Diehl, H. G. Bock, and J. P. Schlöder. A real-time iteration scheme for nonlinear optimization in optimal feedback control. *SIAM Journal on Control and Optimization*, 43:1714–1736, 2005.
[6] M. Diehl, R. Findeisen, F. Allgower, H.G. Bock, and J.P. Schloder. Nominal stability of real-time iteration scheme for nonlinear model predictive control. *Control Theory and Applications, IEE Proceedings -*, 152(3):296–308, May 2005.
[7] B. Houska, H. J. Ferreau, and M. Diehl. Anauto-generatedreal-time iteration algorithm for nonlinear mpc in the microsecond range. *Automatica*, 47(10):22792285, 2011.
[8] C. N. Jones, A. Domahidi, M. Morari, S. Richter, F. Ullmann, and M. Zeilinger. Fast predictive control: Real-time computation and certification. In *Proceeding of the IFAC Nonlinear Predictive Control Conference*, Noordwijkerhout, NL, 2012.
[9] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. M. Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36:789–814, 2000.
[10] Victor M. Zavala, Carl D. Laird, and Lorenz T. Biegler. Fast implementations and rigorous models: Can both be accommodated in NMPC? *International Journal of Robust and Nonlinear Control*, 18(8):800–815, 2008.