Reliable NonLinear Model-Predictive Control via Validated Simulation

Julien Alexandre Dit Sandretto

Reliable NonLinear Model-Predictive Control via Validated Simulation

Julien Alexandre dit Sandretto[1][2]

September 8, 2017

[1]This research benefited from the support of the "Chair Complex Systems Engineering - Ecole Polytechnique, FX, THALES, DGA, DASSAULT AVIATION, DCNS Research, ENSTA ParisTech, Télécom ParisTech, Fondation ParisTech and FDO ENSTA". This research is also partially funded by DGA MRIS.

[2]Authors are with ENSTA ParisTech, Université Paris-Saclay, 828 bd des maréchaux, 91762 Palaiseau Cedex, France alexandre@ensta.fr

Abstract

Model-Predictive Control (MPC) is one of the most advanced control technique nowadays. Indeed, MPC approaches are well known for their robustness and stability properties. Nevertheless, Nonlinear Model-Predictive Control (NMPC), the extension of MPC in the nonlinear world, still poses challenging theoretical, computational and implementation issues. By the help of validated simulation, which can handle nonlinear models, a new algorithm for a robust by-construction control strategy based on NMPC is proposed.

Chapter 1

INTRODUCTION

In the large family of controllers, *Model-Predictive Control* (MPC), or *Receding Horizon Control*, is a model-based control which provides stability and safety (in a certain way) [1]. This approach has proven its efficiency in industrial processes, mainly for systems with slow dynamics, as chemical processes [1, 2]. MPC has been used with success for the control of a ducted fan [3]. MPC consists in computing the control input of a system in order to (i) minimize a cost function, and (ii) to lead to a state respecting physical constraints, which can be on state for safety or on control variables to model the limitation of actuators. For a class of problems, a Lyapunov method can be used to ensure the stability as in [4]. Nonlinear Model-Predictive Control (NMPC) consists in the same approach than MPC, but considering a nonlinear model, a non-quadratic cost function and/or some nonlinear constraints. The main idea behind MPC or NMPC approach is to used a model of the plant in order to predict its behavior following a given control input and so to be able to select the right one considering the cost function and the constraints.

For stability and safety issues, it is obvious that the quality of prediction is very important, even in presence of uncertainties. One of the tools able to consider uncertainties in a safety point of view is the one of interval analysis [5]. Interval analysis (IA) has been already used in control field [6]. More precisely, IA can be a powerful way to develop MPC methods such as in [7, 8] for discrete-time plant or as in [2, 9, 10, 11, 12] for continuous-time model of the plant, see Section 3 for more details.

With the requirement of a reliable and safe-by-construction controller, NMPC problems are solved by the help of validated simulation methods. They are mainly based on Taylor series [13, 14] or on Runge-Kutta methods [15, 16]. The latter is efficient in short simulation with interval initial values and parameters. Moreover, it is also embedded into the constraint satisfaction problems framework [17] offering new capabilities that are the requirements for the synthesis of robust NMPC methods.

The contribution is a generic algorithm based on validated simulation and branching methods to address NMPC without any linearization or problem rewriting as found in the previous works. An heuristic is also given to reduce the overall complexity of the approach.

The paper is divided as follows. In Section 2, we introduce some preliminaries on MPC, interval analysis and validated simulation. In this section, we also introduce some notations used in the following. In Section 3, the validated NMPC is presented. In Section 4, we present the main algorithm of our robust controller. In Section 5, the whole approach is tested on one of the most famous example in the literature. A

conclusion is given in Section 6.

Chapter 2

Preliminary Notions

In this section, we present the three main tools exploited in this paper: MPC, interval analysis and validated simulation.

2.1 Model-Predictive Control

Model-Predictive Control is a closed loop controller formulated as an iterative open-loop controller. More precisely, the later consists in a finite horizon optimal control, considering the system dynamics. In addition, this optimal controller is built in order to validate some constraints on inputs and states. The controller is illustrated in Figure 2.1. In this sketch, no disturbances are considered for the simplicity.

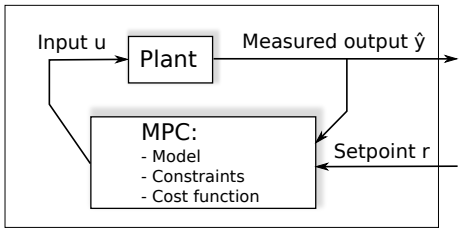

Figure 2.1: Block diagram of a MPC.

2.1.1 Principle of MPC

Starting from measurements at time t, the controller predicts the dynamic behavior of the system over a prediction horizon $T_p = N_p \times T_c$, with N_p the number of optimal pre-computed inputs $U = \{\mathbf{u}_1, \ldots, \mathbf{u}_{N_p}\}$ (or horizon size) and T_c the control sampling time. Only the first input $\mathbf{u}_1$ is injected into the system, the others are forgotten. At the next sampling period $t + T_c$, measurements are taken again. If there were no disturbances and if the model-plant is sufficiently realistic, all the inputs computed previously can be used. Obviously, it is never the case, and all the process is restarted. The timeline of the principle is given in Figure 2.2.

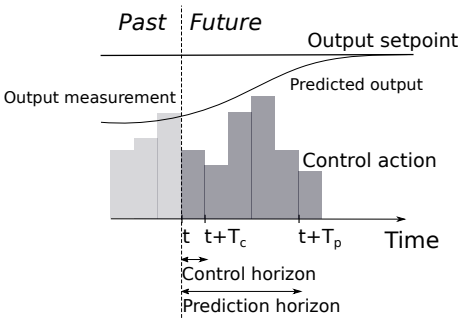

Figure 2.2: Timeline of the MPC process.

2.1.2 Mathematical Formulation of MPC

In the block diagram (see Figure 2.1), the block of MPC makes appear a model, some constraints and a cost function. In this paper, we consider the class of continuous-time systems, approximated with a model described by the following nonlinear differential equation

$$\dot{\mathbf{y}}(t) = f(t, \mathbf{y}(t), \mathbf{u}(t)) \tag{2.1}$$

subject to the constraints

$$\mathbf{y}(t) \in \mathbb{Y}, \forall t \geq 0 \tag{2.2}$$

$$\mathbf{u}(t) \in \mathbb{U}, \forall t \geq 0 \tag{2.3}$$

Here $\mathbf{y}(t) \in \mathbb{R}^n$ and $\mathbf{u}(t) \in \mathbb{R}^m$ denote the vector of states and inputs, respectively. Finally, the optimal control problem successively solved by the NMPC, at each sampling time, is given by

Problem 1

$$\textit{Find } \hat{\mathbf{u}}(.) = \underset{\mathbf{u}(.)}{\operatorname{argmin}} J(\mathbf{y}(t), \mathbf{u}(.))$$

Subject to:

$$\begin{aligned} &\dot{\hat{\mathbf{y}}}(\tau) = f(\tau, \hat{\mathbf{y}}(\tau), \hat{\mathbf{u}}(\tau)), \hat{\mathbf{y}}(t) = \mathbf{y}(t), \\ &\hat{\mathbf{u}}(\tau) \in \mathbb{U}, \ \forall \tau \in [t, t+T_p], \\ &\mathbf{y}(\tau) \in \mathbb{Y}, \ \forall \tau \in [t, t+T_p], \end{aligned} \tag{2.4}$$

with an additive constraint, due to the control sampling

$$\hat{\mathbf{u}}(\tau) = \hat{\mathbf{u}}(t + (i+1)T_c), \forall \tau \in]t + iT_c, t + (i+1)T_c],$$

$$\forall i \in \{0, \ldots, N_p - 1\}, \ \textit{and a cost function}$$

$$J(\mathbf{y}(t), \hat{\mathbf{u}}(.)) = \int_t^{t+T_p} F(\hat{\mathbf{y}}(\tau), \hat{\mathbf{u}}(\tau)) d\tau.$$

The cost functional J is defined as the integral of step cost F. The step cost can arise from economical considerations such as fuel expenditure or from preservation of system such as lifetime optimization. In general, F is used in a quadratic form:

$$F(\mathbf{y}, \mathbf{u}) = (\mathbf{y} - \mathbf{y}_s)^T \mathbf{W} (\mathbf{y} - \mathbf{y}_s) + \mathbf{u}^T \mathbf{Z} \mathbf{u}. \tag{2.5}$$

Here $\mathbf{y}_s$ denotes the reference trajectory, constant or time-varying ($\mathbf{y}_s \in \mathbb{Y}$ is not necessary), and $\mathbf{W},\mathbf{Z}$ are two weight matrices. Mainly three different techniques can be used to choose these weight matrices:

- *Lyapunov (linearization and closed-loop study), see [18]*
- *Normalization between control and state magnitude*
- *Artificial choice to privilege convergence to the set-point or control cost.*

In this paper, we are interesting in a method to obtain a reliable controller, robust to uncertainties. Under these requirements, interval analysis is a powerful tool, which has already proved its efficiency.

2.2 Interval Analysis

The simplest and most common way to represent and manipulate sets of values is *interval arithmetic* (see [5]). An interval $[x_i] = [\underline{x_i}, \overline{x_i}]$ defines the set of reals x_i such that $\underline{x_i} \leq x_i \leq \overline{x_i}$. $\mathbb{IR}$ denotes the set of all intervals over reals. The size or the width of $[x_i]$ is denoted by $w([x_i]) = \overline{x_i} - \underline{x_i}$.

Interval arithmetic extends to $\mathbb{IR}$ elementary functions over $\mathbb{R}$. For instance, the interval sum, *i.e.*, $[x_1]+[x_2] = [\underline{x_1}+\underline{x_2}, \overline{x_1}+\overline{x_2}]$, encloses the image of the sum function over its arguments. An interval vector or a *box* $[\mathbf{x}] \in \mathbb{IR}^n$, is a Cartesian product of n intervals. The enclosing property basically defines what is called an *interval extension* or an *inclusion function*.

Definition 1 (Inclusion function) *Consider a function $f : \mathbb{R}^n \to \mathbb{R}^m$, then $[f] : \mathbb{IR}^n \to \mathbb{IR}^m$ is said to be an extension of f to intervals if*

$$\forall [\mathbf{x}] \in \mathbb{IR}^n, \quad [f]([\mathbf{x}]) \supseteq \{f(\mathbf{x}), \mathbf{x} \in [\mathbf{x}]\} \ .$$

It is possible to define inclusion functions for all elementary functions such as $\times$, $\div$, sin, cos, exp, and so on. The *natural* inclusion function is the simplest to obtain: all occurrences of the real variables are replaced by their interval counterpart and all arithmetic operations are evaluated using interval arithmetic. More sophisticated inclusion functions such as the centered form, or the Taylor inclusion function may also be used (see [6] for more details).

2.3 Validated Simulation

When dealing with validated computation, mathematical representation of an IVP-ODE is as follows:

$$\begin{cases} \dot{\mathbf{y}}(t) = f(t, \mathbf{y}(t)) \\ \mathbf{y}(0) \in [\mathbf{y}_0] \subseteq \mathbb{R}^n. \end{cases} \tag{2.6}$$

We assume that $f : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$ is continuous in t and globally Lipschitz in $\mathbf{y}$, so Equation (2.6) admits a unique solution.

The set (expressed as a box) $[\mathbf{y}_0]$ of initial conditions is used to model some (bounded) uncertainties. For a given initial condition $\mathbf{y}_0 \in [\mathbf{y}_0]$, the solution at time $t > 0$ when

it exists is denoted $\mathbf{y}(t;\mathbf{y}_0)$. The goal, for validated (or rigorous) numerical integration methods, is then to compute the set of solutions of (2.6), *i.e.*, the set of possible solutions at time t given the initial condition in the set of initial conditions $[\mathbf{y}_0]$:

$$\mathbf{y}(t;[\mathbf{y}_0]) = \{\mathbf{y}(t;\mathbf{y}_0) \mid \mathbf{y}_0 \in [\mathbf{y}_0]\}. \tag{2.7}$$

Validated numerical integration schemes, exploiting set-membership framework, aims at producing the solution of the IVP-ODE that is the set defined in (2.7). It results in the computation of an outer approximation of $\mathbf{y}(t;[\mathbf{y}_0])$.

The use of set-membership computation for the problem described above makes possible the design of an inclusion function for the computation of $[\mathbf{y}](t;[\mathbf{y}_0])$ which is an outer approximation of $\mathbf{y}(t;[\mathbf{y}_0])$ defined in (2.7). To do so, a sequence of time instants $t_1, \ldots, t_n$ such that $t_1 < \cdots < t_n$ and a sequences of boxes $[\mathbf{y}_1], \ldots, [\mathbf{y}_n]$ such that $\mathbf{y}(t_{i+1};[\mathbf{y}_i]) \subseteq [\mathbf{y}_{i+1}]$, $\forall i \in [0, n-1]$ are computed. From $[\mathbf{y}_i]$, computing the box $[\mathbf{y}_{i+1}]$ is a classical 2-step method (see [13]):

- *Phase 1:* compute an a priori enclosure $[\mathbf{y}(\xi)]$ of the set $\{\mathbf{y}(t_k;\mathbf{y}_i) \mid t_k \in [t_i, t_{i+1}], \mathbf{y}_i \in [\mathbf{y}_i]\}$ such that $\mathbf{y}(t_k;[\mathbf{y}_i])$ is guaranteed to exist,
- *Phase 2:* compute a tight enclosure of the solution $[\mathbf{y}_{i+1}]$ at time t_{i+1}.

Two main approaches can be used to compute the tight enclosure in *Phase 2*. The first one, and the most used, is the Taylor method [5, 14]. The second one, more recently studied, is the validated Runge-Kutta method [16]. In this paper, we focus on system described by ODEs, but validated simulation can handle Constrained ODEs [16] or Differential Algebraic Equations (DAEs) [19].

Chapter 3

Validated Nonlinear Model Predictive-Control

Different techniques have been tried to validate a NMPC. First one is based on interval analysis and uses the fact that system is discrete with an explicit control [9]. This approach allows authors to compute a validated control for the inverse pendulum. Nevertheless, an inverse pendulum is a continuous system, and then the results obtained are theoretical. Moreover, the control variable is not often explicit in the system equations. An interesting approach is based on the sensitivity analysis (or adjoint) [2, 10]. The adjoint is used to compute the derivatives of dynamics w.r.t. control variable, and to define a least square method in order to optimize the control law. The idea is interesting, but the search is local while a global solution is desired. Recently, a paper [12] detailed the common algorithm of Branch&Bound applied to NMPC. This method is in theory the best one for the purpose, but the problems of complexity and combinatorial avoid its use in a general case. We focus in this paper on general continuous differential equations which implies the use of other techniques.

3.1 Constraints Satisfaction Differential Problem

Firstly, the NMPC problem consists in a differential equation as shown in Equation (2.1) and recalled here:

$$\dot{\mathbf{y}}(t) = f(t, \mathbf{y}(t), \mathbf{u}(t))$$

and at least the two constraints:

- $\mathbf{y}(t) \in \mathbb{Y}, \forall t \geq 0$ for the safety of the system,
- $\mathbf{u}(t) \in \mathbb{U}, \forall t \geq 0$ for the input limits (or saturation).

An interesting value for $\mathbb{Y}$ can be the viability kernel, *i.e.*, the set of viable states, as computed in [20]. In a simple interval notation, the system is described by the following constraint differential equation:

$$\begin{cases} \dot{\mathbf{y}}(t) = f(t, \mathbf{y}(t), \mathbf{u}(t)) \\ \mathbf{y}(t) \in [\mathbf{Y}], \forall t \geq 0 \\ \mathbf{u}(t) \in [\mathbf{U}], \forall t \geq 0 \end{cases} \tag{3.1}$$

A validated simulation procedure starts with the interval enclosures $[\mathbf{y}_0]$ (initial condition of the sliding horizon) and a given constant input $[\mathbf{u}]$, and produces two lists of boxes:

- the list of discretization time steps. $\{[\mathbf{y}_0],\ldots,[\mathbf{y}_{\text{end}}]\}$;
- the list of *a priori* enclosures: $\{[\widetilde{\mathbf{y}}_0],\ldots,[\widetilde{\mathbf{y}}_{\text{end}}]\}$.

Based on these lists, two functions depending on time can be defined

$$R:\left\{\begin{array}{c}\mathbb{R}\mapsto\mathbb{IR}^n\\ t\to[\mathbf{y}]\end{array}\right. \tag{3.2}$$

with $\{\mathbf{y}(t;\mathbf{y}_0):\forall\mathbf{y}_0\subseteq[\mathbf{y}_0]\}\subseteq[\mathbf{y}]$, and

$$\widetilde{R}:\left\{\begin{array}{c}\mathbb{IR}\mapsto\mathbb{IR}^n\\ [\underline{t},\overline{t}]\to[\widetilde{\mathbf{y}}]\end{array}\right. \tag{3.3}$$

with $\{\mathbf{y}(t;\mathbf{y}_0):\forall\mathbf{y}_0\in[\mathbf{y}_0]\wedge\forall t\in[\underline{t},\overline{t}]\}\subseteq[\widetilde{\mathbf{y}}]$.

These functions are abstracted, but guaranteed, solutions of (3.1). Therefore, the process of validated simulation, mixed with constraint programming and abstraction of time functions provides an efficient tool for the prediction of the system evolution, w.r.t. a given control input. A contractor approach to solve the constraints on state, as in [9], provides an outer approximation of the control, but does not prove that the safety is fulfilled. An approach based on Branch&Prune is then a better choice to show that the safety is respected.

3.2 Optimization Cost

We consider the general class of continuous cost function

$$J(\mathbf{y}(t),\hat{\mathbf{u}}(.))=\int_t^{t+T_p}F(\hat{\mathbf{y}}(\tau),\hat{\mathbf{u}}(\tau))d\tau \tag{3.4}$$

with $\hat{\mathbf{y}}(t)$ the solution of (2.1) driven by the input $\hat{\mathbf{u}}(t)$. Considering the quadratic form F as given in (2.5), the cost is computed by

$$\begin{aligned}&J(\mathbf{y}(t),\hat{\mathbf{u}}(.))\\&=\int_t^{t+T_p}\left[(\hat{\mathbf{y}}(\tau)-\mathbf{y}_s)^T\mathbf{W}(\hat{\mathbf{y}}(\tau)-\mathbf{y}_s)+\hat{\mathbf{u}}(\tau)^T\mathbf{Z}\hat{\mathbf{u}}(\tau)\right]d\tau\\&=\int_t^{t+T_p}\left[(\hat{\mathbf{y}}(\tau)-\mathbf{y}_s)^T\mathbf{W}(\hat{\mathbf{y}}(\tau)-\mathbf{y}_s)\right]d\tau\\&+\int_t^{t+T_p}\left[\hat{\mathbf{u}}(\tau)^T\mathbf{Z}\hat{\mathbf{u}}(\tau)\right]d\tau\end{aligned} \tag{3.5}$$

We consider in NMPC approach that $\hat{\mathbf{u}}(.)$ is piece-wise constant, then

$$\begin{aligned}&J(\mathbf{y}(t),\hat{\mathbf{u}}(.))\\&=\int_t^{t+T_p}\left[(\hat{\mathbf{y}}(\tau)-\mathbf{y}_s)^T\mathbf{W}(\hat{\mathbf{y}}(\tau)-\mathbf{y}_s)\right]d\tau+\sum_{i=1}^{N_p}\left[\hat{\mathbf{u}}_i^T\mathbf{Z}\hat{\mathbf{u}}_i\right]\end{aligned} \tag{3.6}$$

By the help of validated simulation (see Section 3.1), $\hat{\mathbf{y}}(\tau)$ can be enclosed by $R(\tau)$. From [5], the interval integral is bounded according to the following lemma.

Lemma 1 *If f is continuous in $X = [a,b]$, then $\int_a^b f(x)dx \in (b-a)[f](X)$*

Hence, on a given control horizon $\mathrm{T}_i = [t+(i-1)T_c, t+iT_c]$, the following enclosures hold:

- $\hat{\mathbf{y}}(\tau) \in \tilde{R}(\mathrm{T}_i), \forall \tau \in \mathrm{T}_i$
- $\int_t^{t+T_p} \left[(\hat{\mathbf{y}}(\tau)-\mathbf{y}_s)^T \mathbf{W}(\hat{\mathbf{y}}(\tau)-\mathbf{y}_s)\right] d\tau$
  $\in T_c \sum_{i=1}^{N_p} (\tilde{R}(\mathrm{T}_i)-\mathbf{y}_s)^T \mathbf{W}(\tilde{R}(\mathrm{T}_i)-\mathbf{y}_s)$

Finally, the cost can be bounded by:

$$\begin{aligned} & J(\mathbf{y}(t), \hat{\mathbf{u}}(.)) \\ & \in T_c \sum_{i=1}^{N_p} (\tilde{R}(\mathrm{T}_i)-\mathbf{y}_s)^T \mathbf{W}(\tilde{R}(\mathrm{T}_i)-\mathbf{y}_s) + \sum_{i=1}^{N_p} \left[\hat{\mathbf{u}}_i^T \mathbf{Z} \hat{\mathbf{u}}_i\right] \\ & \leq ub(T_c \sum_{i=1}^{N_p} (\tilde{R}(\mathrm{T}_i)-\mathbf{y}_s)^T \mathbf{W}(\tilde{R}(\mathrm{T}_i)-\mathbf{y}_s) + \sum_{i=1}^{N_p} \left[\hat{\mathbf{u}}_i^T \mathbf{Z} \hat{\mathbf{u}}_i\right]) \end{aligned} \tag{3.7}$$

where $ub()$ denotes the upper bound.

3.3 Global Approach

Taking into account all the previous tools and remarks, the proposed approach is the following:

- First, starting from the interval of input (it validates the input limitations by construction), a filtering w.r.t. safety is done with a Branch&Prune algorithm;
- Second, an optimization is performed on the inputs, w.r.t. cost function, with a Branch&Bound algorithm.

To counteract the effect of the past (accepting values in the filtering part at t that not permit to find a valid control at instant $t+T_c$), a restart procedure is needed. Finally, to avoid the combinatorial issue, a direct pruning is used in both branching algorithms. An important remark is that, in this paper, we are focusing on the constraint satisfaction (*i.e.*, safety of the system), and we relax the optimization part by accepting a sub-optimal solution, in order to reduce the complexity of the algorithm.

Chapter 4

ALGORITHMS

The algorithms used to solve NMPC are now given.

4.1 Closed loop

The first algorithm is the main closed-loop presented in Figure 2.1. In Algorithm 1, T_f is the final time, used to terminate the loop. In a simulated experiment, Acquire is not a measurement but the result of a simulation from a previous value $\mathbf{y}(t - T_c)$. A noise can be added to test the robustness of the NMPC. The function Send makes nothing in simulation, otherwise it will be a communication with the motor controller for example.

4.2 NMPC

In Algorithm 2, the two step method used to solve NMPC is presented. Filtering and Optimization are described in the following.

4.3 Filtering

The first part of our NMPC approach is the filtering which consists in computing inputs such that constraints are respected with guarantee, given in Algorithm 3. It consists into two nested loops to allow a restart if the filtering arrives in a deadlock (as explained in Section 3.3). The Prune function performs a bisection but keeps only one side (direct

Algorithm 1 Closed loop

Require: $T_c, T_f, t = 0, \mathbf{y}_c$
  while $t < T_f$ **do**
    Acquire $\mathbf{y}(t)$
    $\hat{\mathbf{u}}_1 = \text{NMPC}(\mathbf{y}(t), T_c, \mathbf{y}_c)$
    Send $\hat{\mathbf{u}}_1$
    $t = t + T_c$
  end while

Algorithm 2 NMPC procedure

Require: $\mathbf{y}(t), T_c, \mathbf{y}_c$
$\mathbf{u} = \text{Filtering}(\mathbf{u}, \mathbf{y}(t), T_c)$
$\hat{\mathbf{u}}_1 = \text{Optimization}(\mathbf{u}, \mathbf{y}_c)$

pruning), more details are given in the following section. Simulation is done with a validated integration method, see Section 2.3. If a simulation has been already computed at the considered time instant, it is possible to use a propagation as described in [21]. The function Test_adjoint exploits the result of a simulation with sensitivity analysis. Sensitivity has been used in a different manner in [2]. It consists to simultaneously simulate an ODE described by

$$\dot{s} = \frac{\partial f}{\partial y} \times s + \frac{\partial f}{\partial u}. \tag{4.1}$$

to obtain the adjoint function $s(t) = \frac{\partial y}{\partial u}$, which is the evolution of state w.r.t. control. Using the adjoint function, Test_adjoint() returns the Boolean formula:

$$\begin{aligned}((\exists \tau \in [t, t+T_c] :: 0 \in [s(\tau)] \text{ AND } [s(\tau)] \not\subset [-\varepsilon, \varepsilon]) \\ \text{OR } (\exists \tau \in [t, t+T_c] : y(\tau) \not\subset [Y]))\end{aligned} \tag{4.2}$$

The first part of this formula means that the control (an interval of control) leads to two opposite directions, and the second one is the safety constraint.

For a given box $\mathbf{a} = (a_1, a_2, ..., a_n)$, the function left($\mathbf{a}$) - resp. right($\mathbf{a}$) - returns the left part of $\mathbf{a}$ after a bisection (with a largest first heuristic for example) - resp. the right part of $\mathbf{a}$), such that $\mathbf{a} = \text{left}(\mathbf{a}) \cup \text{right}(\mathbf{a})$.

4.4 Pruning

As explained before, an important choice has been made in our approach that consists in a direct elimination in the branch algorithm. Indeed, branches are not kept to avoid combinatorial problem. The pruning part of Algorithm 3 is highly important and decides the characteristic of the obtained solution (optimal or sub-optimal). Two main ways can be considered:

- The pruning can be done w.r.t. optimum by eliminating a branch which leads to a completely invalid state or with a cost strictly greater than the other branch (it is rare with interval methods)
- Or w.r.t. sub-optimum, for example, by eliminating a branch leading to a state further from the set-point or partially outside the safety set, with a cost with an upper bound greater than the other branch, etc.

We will see in the experimentation that it is interesting to compare different pruning methods. The cost computation for the left and right branches can be easily parallelized and computed using propagation (see [21]).

4.5 Optimization

After the filtering procedure, the vector of inputs is valid w.r.t. state constraints. We can then perform a final optimization procedure to find a punctual u_1.

Algorithm 3 Filtering

Require: $\mathbf{y}(t), T_c, \mathbf{y}_c, N_p, Tol$
  while not success **do**
    $i = 1$
    while $i \leq N_p$ **do**
      if $w(\mathbf{u}_i) > Tol$ **then**
        $\mathbf{u}_i$=Prune(left($\mathbf{u}_i$) or right($\mathbf{u}_i$))
      end if
      $\mathbf{y}(t+T_c)$ = Simulation($\mathbf{y}(t), \mathbf{u}_i, T_c$) - or Propagate($\mathbf{u}_i$)
      if $\mathbf{y}(t+T_c) \subset \mathbf{y}_c$ **then**
        success = true
        $i = i+1$; $\mathbf{y}(t) = \mathbf{y}(t+T_c)$; $t = t+T_c$
      else
        if Test_adjoint() **then**
          if $w(\mathbf{u}_i) > Tol$ **then**
            $\mathbf{u}_i$=Prune(left($\mathbf{u}_i$) or right($\mathbf{u}_i$))
          else
            success = false and break the loop
          end if
        else
          success = true
          $i = i+1$; $\mathbf{y}(t) = \mathbf{y}(t+T_c)$; $t = t+T_c$
        end if
      end if
    end while
  end while

Algorithm 4 Optimization

Require: $\mathbf{y}(t), U, Tol$
  while $w(\mathbf{u}_1) > Tol$ **do**
    $U_l = \{\text{left}(\mathbf{u}_1), \mathbf{u}_2, \ldots, \mathbf{u}_{N_p}\}$
    $U_r = \{\text{right}(\mathbf{u}_1), \mathbf{u}_2, \ldots, \mathbf{u}_{N_p}\}$
    if $J(\mathbf{y}(t), U_l) > J(\mathbf{y}(t), U_r)$ **then**
      $U = Ur$
    else
      $U = U_l$
    end if
  end while
  if $lb(|\mathbf{u}_1|) > ub(|\mathbf{u}_1|)$ **then**
    $\mathbf{u}_1 = ub(\mathbf{u}_1)$
  else
    $\mathbf{u}_1 = lb(\mathbf{u}_1)$
  end if

Chapter 5

EXPERIMENTS ON INVERTED PENDULUM

We implement the presented algorithms with the library DynIbex[1]. This tool is based on Runge-Kutta methods and provides some differential constraint programming facilities. We focus our experiments on the impact of different pruning. As example, we consider the classical inverted pendulum benchmark. It is defined by the dynamic equation:

$$\begin{cases} \dot{y}_1 = y_2 \\ \dot{y}_2 = K_s \times \sin y_1 - K_c \times u \times \cos y_1 \end{cases} \tag{5.1}$$

where K_s and K_c are the parameters of the pendulum and u the input of the system, which is the acceleration of the carrier. We assume that friction is negligible and pendulum is a rigid body. Values of parameters (in seconds and radians) are taken from [9], they are $K_s = 109$, $K_c = 11.11$, $N_p = 10$, $T_c = 0.01$, $T_p = 0.1$, $y_1(0) = -\pi$, $y_2(0) = 0$, and $T_f = 0.5$. The safety constraints on state variables and inputs are: $y_1 \in [-5\pi/2, \pi/2]$, $y_2 \in [-50, 50]$, $u \in [-100, 100]$, and $y_c \in [-0.2, 0.2]$.

Remark 1 *With a tolerance on box width equal to* 1 *for the bisection, the input given in* $[-100, 100]$ *and with* $N_p = 10$*, a classical branch algorithm leads to* 200^{10} *branches.*

Three experiments are made, with a pruning based on cost functional with $Z = 0.5$ and $W = 100$ (Figures 5.1 and 5.2), and $W = 1000$ (Figures 5.3 and 5.4), and with a pruning based on the following rules (Figures 5.5 and 5.6):

1. If $y(\tau, U_l) \not\subset [Y], \forall \tau \in [t, t + T_p]$ then we keep U_r
2. If $y(\tau, U_r) \not\subset [Y], \forall \tau \in [t, t + T_p]$ then we keep U_l
3. If $|y(T_p, U_l) - y_c| < |y(T_p, U_l) - y_c|$ then we keep U_l, otherwise U_r.

The Hausdorff distance is used. The cost is computed only in the optimization process, with $W = 100$. It leads to a strong gain of time computation (divided by two). With different pruning functions, we are able to obtain the three possible ways to control the inverted pendulum:

[1] `http://perso.ensta-paristech.fr/~chapoutot/dynibex/`

- with a long but decreasing impulse to try to catch the equilibrium and stabilize at this instant (see Figure 5.2)
- with a strong impulse and a bang bang controller (see Figure 5.4 and theory on inverted pendulum)
- with a too strong impulse and a balancing control around the equilibrium (see Figure 5.6)

Discussion

The presented approach succeed to control and stabilize a complex system. By tuning the pruning procedure, three different solutions can be found. This capability is interesting but also implies to have an *a priori* knowledge about the behavior of the system dynamics.

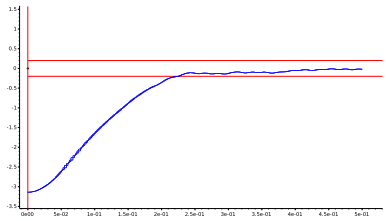

Figure 5.1: State of system w.r.t. time with a weight of 100.

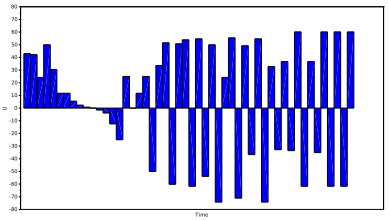

Figure 5.2: Inputs computed by NMPC with a weight of 100.

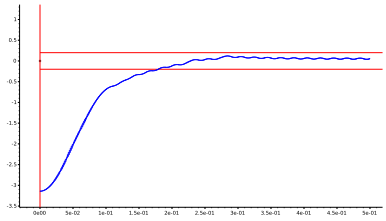

Figure 5.3: State of system w.r.t. time with a weight of 1000.

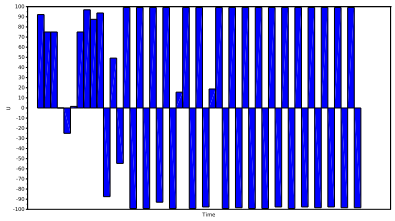

Figure 5.4: Inputs computed by NMPC with a weight of 1000.

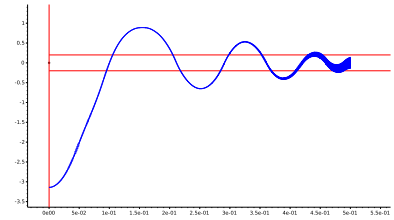

Figure 5.5: State of system w.r.t. time with other prune.

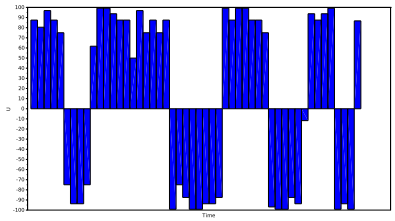

Figure 5.6: Inputs computed by NMPC with other prune.

Chapter 6

Conclusion and future work

In this paper, we proposed a complete algorithm to solve NMPC in a validated way. We focused on the safety and relaxed the optimization in order to counteract the combinatorial problem. Different techniques can be used to prune between branches. We showed two different approaches, as well as the impact of weight in the cost function.

As future work, it is important to optimize the implementation of NMPC and complete the experiments on other examples, such as distillation column, as DynIBEX is able to consider differential algebraic equations.

Bibliography

[1] F. Allgower, R. Findeisen, Z. K. Nagy *et al.*, "Nonlinear model predictive control: from theory to application," *Journal of the Chinese Institute of Chemical Engineers*, vol. 35, no. 3, pp. 299–315, 2004.

[2] A. Rauh, L. Senkel, J. Kersten, and H. Aschemann, "Reliable control of high-temperature fuel cell systems using interval-based sliding mode techniques," *IMA Journal of Mathematical Control and Information*, 2014.

[3] R. Franz, M. Milam, and J. Hauser, "Applied receding horizon control of the caltech ducted fan," in *Proceedings of the 2002 American Control Conference (IEEE Cat. No. CH37301)*, vol. 5. IEEE, 2002, pp. 3735–3740.

[4] A. Jadbabaie, J. Yu, and J. Hauser, "Stabilizing receding horizon control of nonlinear systems: a control lyapunov function approach," in *American Control Conference, 1999. Proceedings of the 1999*, vol. 3. IEEE, 1999, pp. 1535–1539.

[5] R. E. Moore, *Interval Analysis*, ser. Series in Automatic Computation. Prentice Hall, 1966.

[6] L. Jaulin, M. Kieffer, O. Didrit, and E. Walter, *Applied Interval Analysis*. Springer, 2001.

[7] J. M. Bravo, T. Alamo, and E. F. Camacho, "Robust MPC of constrained discrete-time nonlinear systems based on approximated reachable sets," *Automatica*, vol. 42, no. 10, pp. 1745–1751, Oct. 2006.

[8] D. Limon, T. Alamo, J. M. Bravo, E. F. Camacho, D. R. Ramirez, D. Muñoz de la Peña, I. Alvarado, and M. R. Arahal, *Interval Arithmetic in Robust Nonlinear MPC*. Springer, 2007, pp. 317–326.

[9] F. Lydoire and P. Poignet, "Nonlinear model predictive control via interval analysis," in *Conference on Decision and Control*. IEEE, 2005, pp. 3771–3776.

[10] A. Rauh and H. Aschemann, "Interval-based sliding mode control and state estimation for uncertain systems," in *Methods and Models in Automation and Robotics*. IEEE, 2012, pp. 595–600.

[11] L. Senkel, A. Rauh, and H. Aschemann, "Interval-based sliding mode observer design for nonlinear systems with bounded measurement and parameter uncertainty," in *Methods and Models in Automation and Robotics*. IEEE, 2013, pp. 818–823.

[12] B. J. Kubica, "Preliminary experiments with an interval model-predictive-control solver," in *Parallel Processing and Applied Mathematics*. Springer, 2016, pp. 464–473.

[13] R. J. Lohner, "Enclosing the Solutions of Ordinary Initial and Boundary Value Problems," *Computer Arithmetic*, pp. 255–286, 1987.

[14] N. S. Nedialkov, K. Jackson, and G. Corliss, "Validated solutions of initial value problems for ordinary differential equations," *Appl. Math. and Comp.*, vol. 105, no. 1, pp. 21 – 68, 1999.

[15] O. Bouissou and M. Martel, "GRKLib: a Guaranteed Runge-Kutta Library," in *Scientific Computing, Computer Arithmetic and Validated Numerics*, 2006.

[16] J. Alexandre dit Sandretto and A. Chapoutot, "Validated explicit and implicit Runge-Kutta methods," *Reliable Computing*, vol. 22, pp. 79–103, 2016.

[17] J. Alexandre dit Sandretto, A. Chapoutot, and O. Mullier, "Formal verification of robotic behaviors in presence of bounded uncertainties," in *International Conference on Robotic Computing*. IEEE, 2017.

[18] H. Chen and F. Allgower, "A quasi-infinite horizon nonlinear model predictive control scheme with guaranteed stability," *Automatica*, vol. 34, no. 10, pp. 1205 – 1217, 1998.

[19] J. Alexandre dit Sandretto and A. Chapoutot, "Validated Simulation of Differential Algebraic Equations with Runge-Kutta Methods," *Reliable Computing electronic edition*, vol. 22, 2016.

[20] D. Monnet, L. Jaulin, J. Ninin, J. Alexandre dit Sandretto, and A. Chapoutot, "Viability kernel computation based on interval methods," in *Small Workshop on Interval Methods*, 2015.

[21] J. Alexandre dit Sandretto and A. Chapoutot, "Contraction, propagation and bisection on a validated simulation of ODE," 2016.