Tuning PI controller in non-linear uncertain closed-loop systems with interval analysis*

Julien Alexandre dit Sandretto, Alexandre Chapoutot, and Olivier Mullier

Unité d'Informatique et d'Ingénierie des Systèmes,
ENSTA ParisTech, Université Paris-Saclay,
828 bd des Maréchaux, 91762 Palaiseau cedex France
alexandre@ensta.fr, chapoutot@ensta.fr, mullier@ensta.fr

Abstract

The tuning of a PI controller is usually done through simulation, except for few classes of problems, e.g., linear systems. With a new approach for validated integration allowing us to simulate dynamical systems with uncertain parameters, we are able to design guaranteed PI controllers. In practical, we propose a new method to identify the parameters of a PI controller for non-linear plants with bounded uncertain parameters using tools from interval analysis and validated simulation. This work relies on interval computation and guaranteed numerical integration of ordinary differential equations based on Runge-Kutta methods. Our method is applied to the well-known cruise-control problem, under a simplified linear version and with the aerodynamic force taken into account leading to a non-linear formulation.

1998 ACM Subject Classification B.5.2 Simulation; I.6.6 Simulation Output Analysis; G.1.7 Initial Value Problem; G.1.0 Interval Arithmetic

Keywords and phrases PID Tuning, huaranteed numerical integration, non-linear ordinary differential equations

Digital Object Identifier 10.4230/OASIcs.SynCoP.2015.91

1 Introduction

Recently [2], we developed a new tool for validated simulation [17, 6, 14, 15] of Ordinary Differential Equations (ODE). This tool, by using affine arithmetic [9], is able to handle ODEs with uncertain (and bounded) parameters. This new capability allows us to simulate a dynamical system controlled by a proportional integral (PI), whose parameters K_p (proportional gain), K_i (integral gain) are not well-known. The direct advantage is to use the simulation process to validate or reject some parameter values. This approach is, in philosophy, similar to Ziegler–Nichols approach [20]. Another method, the model of Broida [7], is also based on identification of parameters but works only for a linear problem in open loop. All the existing methods are empirical and do not provide any guarantee on the behavior of the system, and cannot consider uncertainties on the physical part. Interval analysis is often used in applications like robust control [13]. Indeed, the methods provided by the interval formalization are able to consider any kind of bounded uncertainties, manage with non-linear models and offer a guarantee on the numerical computation.

* This research benefited from the support of the «Chair Complex Systems Engineering - Ecole Polytechnique, THALES, DGA, FX, DASSAULT AVIATION, DCNS Research, ENSTA ParisTech, Télécom ParisTech, Fondation ParisTech and FDO ENSTA».

2nd International Workshop on Synthesis of Complex Parameters (SynCoP'15).
Editors: Étienne André and Goran Frehse; pp. 91–102

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Until now, to the best of our knowledge, interval analysis for robust control only considers linear systems [19, 4]. Indeed for this class of problems, robust control problem is cast into a problem of stability and performance based on the characteristic polynomial associated to the transfer function of the closed-loop system. Unfortunately, for non-linear closed-loop systems this approach is no longer possible.

The paper is organized as followed. Main interval analysis tools are recalled in Section 2. The tuning algorithm is presented in Section 3 including also a remainder of the PI controller theory. Some experimental results are presented in Section 4 before concluding in Section 5.

2 Interval analysis tools

We recall in this section the main tools coming from interval analysis and particularly affine arithmetic to tighten the issue of interval arithmetic used in our work.

2.1 Interval arithmetic

The simplest and most common way to represent and manipulate sets of values is *interval arithmetic* [16]. 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. The size or the width of $[x_i]$ is denoted by $w([x_i]) = \overline{x_i} - \underline{x_i}$.

Interval arithmetic [16] 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, and this enclosing property basically defines what is called an *interval extension* or an *inclusion function*.

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

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

2.2 Affine arithmetic

Interval arithmetic provides a good solution to manage with uncertainties. Nevertheless, this representation usually produces too much over-approximated results in particular because of the *dependency problem*. An iterative scheme, such as a mathematical series or an integration scheme, leads typically to a dependency problem: each step depends on the previous ones.

▶ **Example 2.** Consider the ordinary differential equation $\dot{x}(t) = -x$ solved with the Euler's method with an initial value ranging in the interval $[0, 1]$ and with a step-size of $h = 0.5$. For one step of integration, we have to compute with interval arithmetic the expression $e = x + h \times (-x)$ which produces as a result the interval $[-0.5, 1]$. Rewriting the expression e such that $e' = x(1-h)$, we obtain the interval $[0, 0.5]$ which is the exact result. Unfortunately, we cannot in general rewrite expressions with only one occurrence of each variable. More generally, it can be shown that for most integration schemes the width of the result can only grow if we interpret sets of values as intervals [18]. ■

To avoid this problem we use an improvement over interval arithmetic named *affine arithmetic* [9] which can track linear correlation between program variables. A set of values is represented by an *affine form* $\hat{x}$, i.e., a formal expression of the form $\hat{x} = \alpha_0 + \sum_{i=1}^{n} \alpha_i \varepsilon_i$ where the coefficients α_i are real numbers, α_0 being called the *center* of the affine form, α_i, $i \geqslant 1$, are called *partial deviations* and the ε_i, called *noise symbols*, are independent components of the total uncertainty on $\hat{x}$ with unknown values ranging over the interval

$[-1,1]$. Obviously, an interval $a=[a_1,a_2]$ can be seen as the affine form $\hat{x}=\alpha_0+\alpha_1\varepsilon$ with $\alpha_0=(a_1+a_2)/2$ and $\alpha_1=(a_2-a_1)/2$.

Affine arithmetic extends usual operations on real numbers in the expected way. For instance, the affine combination of two affine forms $\hat{x}=\alpha_0+\sum_{i=1}^{n}\alpha_i\varepsilon_i$ and $\hat{y}=\beta_0+\sum_{i=1}^{n}\beta_i\varepsilon_i$ with $a,b,c\in\mathbb{R}$, is an affine form given by

$$a\hat{x}\pm b\hat{y}+c \quad = \quad (a\alpha_0\pm b\beta_0+c)+\sum_{i=1}^{n}(a\alpha_i\pm b\beta_i)\varepsilon_i \ . \tag{1}$$

However, unlike the addition, non linear operations create new noise symbols. Multiplication for example can be defined by

$$\hat{x}\times\hat{y} \quad = \quad \alpha_0\alpha_1+\sum_{i=1}^{n}(\alpha_i\beta_0+\alpha_0\beta_i)\varepsilon_i+\nu\varepsilon_{n+1} \quad \text{with} \quad \nu=\left(\sum_{i=1}^{n}|\alpha_i|\right)\times\left(\sum_{i=1}^{n}|\beta_i|\right) \ . \tag{2}$$

Operations as sin or exp are translated into affine forms with Chebyshev polynomials [9]. For practical details on soundness w.r.t. floating-point computations and the performance of this arithmetic w.r.t. the number of noise symbols see [5].

▶ **Example 3.** Consider again $e=x+h\times(-x)$ with $h=0.5$ and $x=[0,1]$ which is associated to the affine form $\hat{x}=0.5+0.5\varepsilon_1$. Evaluating e with affine arithmetic without rewriting the expression, we obtain $[0,0.5]$ as a result. ■

Note that the set-based evaluation of an expression only consists in substituting all the mathematical operators, like + or sin, by their counterpart in affine arithmetic. We denote by $\mathrm{Aff}(e)$ the evaluation of the expression e using affine arithmetic.

2.3 Guaranteed numerical integration with Runge-Kutta methods

In this section, we recall our previous work [5] on which the extension in [2] is based on.

▶ **Definition 4** (Initial Value Problem (IVP))**.** Consider an Ordinary Differential Equation (ODE) with a given initial condition

$$\dot{y}(t)=f\left(t,y(t),d\right) \quad \text{with} \quad y(0)\in Y_0, \tag{3}$$

with $f:\mathbb{R}\times\mathbb{R}^n\times\mathbb{R}^m\to\mathbb{R}^n$ assumed to be continuous in t and d and globally Lipschitz in y. We assume that parameters d are constant and bounded. An IVP consists in finding a function $y(t)$ described by the ODE for all d and satisfying the initial condition.

A numerical integration method computes a sequence of approximations (t_n,y_n) of the solution $y(t;y_0)$ of the IVP defined in Equation (3) such that $y_n\approx y(t_n;y_{n-1})$.

The simplest method is Euler's method in which $t_{i+1}=t_i+h$ for some step-size h and $y_{i+1}=y_i+h\times f(t_i,y_i,d)$; so the derivative of y at time t_i, $f(t_i,y_i,d)$, is used as an approximation of the derivative on the whole time interval to perform a linear interpolation. This method is very simple and fast, but requires small step-sizes. More advanced methods coming from the Runge-Kutta family use a few intermediate computations to improve the approximation of the derivative. The general form of an explicit s-stage Runge-Kutta formula, that is using s evaluations of f, is

$$y_{n+1}=y_n+h\sum_{i=1}^{s}b_ik_i \ , \tag{4a}$$

$$k_1=f\big(t_n,\,y_n,d\big) \ , \qquad k_i=f\Big(t_n+c_ih,\,y_n+h\sum_{j=1}^{i-1}a_{ij}k_j,d\Big) \ , \quad i=2,3,\ldots,s \ . \tag{4b}$$

The coefficients c_i, a_{ij} and b_i fully characterize the method. To make Runge-Kutta validated, the challenging question is how to compute a bound on the distance between the true solution and the numerical solution, defined by $y(t_n; y_{n-1}) - y_n$. This distance is associated to the *local truncation error* (LTE) of the numerical method.

To bound the LTE, we rely on *order condition* [12] respected by all Runge-Kutta methods. This condition states that a method of this family is of order p iff the $p+1$ first coefficients of the Taylor expansion of the solution and the Taylor expansion of the numerical methods are equal. In consequence, LTE is proportional to the Lagrange remainders of Taylor expansions. In previous work [5], LTE is defined by

$$y(t_n; y_{n-1}) - y_n = \frac{h^{p+1}}{(p+1)!}\left(f^{(p)}\left(\xi, y(\xi; y_{n-1}, d)\right) - \frac{d^{p+1}\phi}{dt^{p+1}}(\eta)\right) \\ \xi \in]t_k, t_{k+1}[\text{ and } \eta \in]t_n, t_{n+1}[\ . \quad (5)$$

The function $f^{(n)}$ stands for the n-th derivative of function f w.r.t. time t that is $\frac{d^n f}{dt^n}$ and $h = t_{n+1} - t_n$ is the step-size. The function $\phi : \mathbb{R} \to \mathbb{R}^n$ is defined by $\phi(t) = y_n + h\sum_{i=1}^{s} b_i k_i(t)$ where $k_i(t)$ are defined as Equation (4b).

The challenge to make Runge-Kutta integration schemes safe w.r.t. the true solution of IVP is then to compute a bound of the result of Equation (5). In other words we have to bound the value of $f^{(p)}\left(\xi, y(\xi; y_{n-1}), d\right)$ and the value of $\frac{d^{p+1}\phi}{dt^{p+1}}(\eta)$. The latter expression is straightforward to bound because the function ϕ only depends on the value of the step-size h, and so does its $(p+1)$-th derivative. The bound is then obtain using the affine arithmetic.

However, the expression $f^{(p)}\left(\xi, y(\xi; y_{n-1}), d\right)$ is not so easy to bound as it requires to evaluate f for a particular value of the IVP solution $y(\xi; y_{n-1})$ at an unknown time $\xi \in]t_n, t_{n+1}[$. The solution used is the same as the one found in [17, 6] and it requires to bound the solution of IVP on the interval $[t_n, t_{n+1}]$. This bound is usually computed using the Banach's fixpoint theorem applied with the Picard-Lindelöf operator, see [17]. This operator is used to compute an enclosure of the solution $[\tilde{y}]$ of IVP over a time interval $[t_n, t_{n+1}]$, that is for all $t \in [t_n, t_{n+1}]$, $y(t; y_{n-1}) \in [\tilde{y}]$. We can hence bound $f^{(p)}$ substituting $y(\xi; y_{n-1})$ by $[\tilde{y}]$.

The main drawback of the previous approach is that implicit Runge-Kutta methods cannot be validated because ϕ is defined implicitly as a function of k_i. Implicit Runge-Kutta methods are important to deal with *stiff ordinary differential equations* and they have very good stability properties that make them suitable for validated numerical integration. Moreover, as function ϕ is defined over f then $\frac{d^{p+1}\phi}{dt^{p+1}}$ involves time derivatives of f. Hence computing separately $\frac{d^{p+1}\phi}{dt^{p+1}}$ and $f^{(p)}$, without taking into account shared intermediate computation, increases the simulation time. In [2], we define a new formula for LTE for any Runge-Kutta methods based on Fréchet derivatives, see [8] for more details. The new result in [2] is the definition of new validated numerical integration methods based on implicit Runge-Kutta methods.

▶ **Example 5.** Here we present an example coming from the `System 61` in the Vericomp database [3], which is defined by:

$$\begin{pmatrix} \dot{x}_0 \\ \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} 1 \\ x_2 \\ \frac{1}{6}x_1^3 - x_1 + 2\sin\left(d \cdot x_0\right) \end{pmatrix} \quad (6)$$

with $d = [2.78, 2.79]$ and the initial condition is $x_0(0) = x_1(0) = x_2(0) = 0$. Using our guaranteed integration method to compute the value of $x(10)$, we obtain $x(10) =$

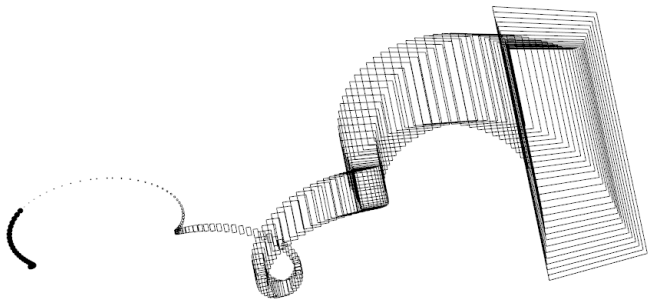

Figure 1 Complete simulation of `System 61`, see Eq. (6), in the Vericomp database over the time interval $[0, 10]$.

$([10, 10], [-1.6338, 1.69346], [-1.55541, 1.4243])^T$, with 4 rejected Picard-Lindelöf and 196 accepted ones. The minimum step-size is $h_{\min} = 0.00636859$ and the maximum step-size $h_{\max} = 0.070553$. Finally the maximum truncature error of the method is 8.9278×10^{-8}. Figure 1 presents the complete simulation of Equation (6). ■

2.4 Paving

We call paving of a set $\mathcal{S} \subset \mathbb{R}^n$ the list of non-overlapping[1] boxes $[\mathbf{x_i}]$ with a non null width, such that each boxes $[\mathbf{x_i}] \subset \mathcal{S}$ [13]. This tool can be used to describe a set, by a list of inner boxes ($[\mathbf{x_i}] \subset \mathcal{S}$), the outer boxes ($[\mathbf{x_i}] \not\subset \mathcal{S}$) and the frontier, i.e. a list of boxes for which we cannot conclude of the membership to $\mathcal{S}$ in an acceptable computation time. For example, if the set $\mathcal{S}$ describes a ring such as $\mathcal{S} = \{(x, y) \mid x^2 + y^2 \in [1, 2]\}$, the paving of $\mathcal{S}$ in the box $[-2, 2] \times [-2, 2]$ gives the Figure 2.

3 Validated tuning method for PI controllers

3.1 PID controllers

PID controllers are widely used in industrial setting for integrating processes (see, e.g., this survey [10]). Therefore many techniques for their design and tuning have been proposed. In the next section we deal with the design of guaranteed PID controllers.

▶ **Definition 6.** A proportional-integral-derivative controller (PID), see [1] for more details, is represented by a tuple of tuning parameters $(K_p, K_i, K_d) \in \mathbb{R}^3$ designed to compute an error value

$$e(t) = r(t) - y(t) \tag{7}$$

with $r(t)$ a desired setpoint and $y(t)$ a measured process. The general mathematical description of PID is

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{d}{dt} e(t) \tag{8}$$

[1] This is true in $\mathbb{R}^n$ but no longer stands with a floating point representation due to rounding preserving over approximations.

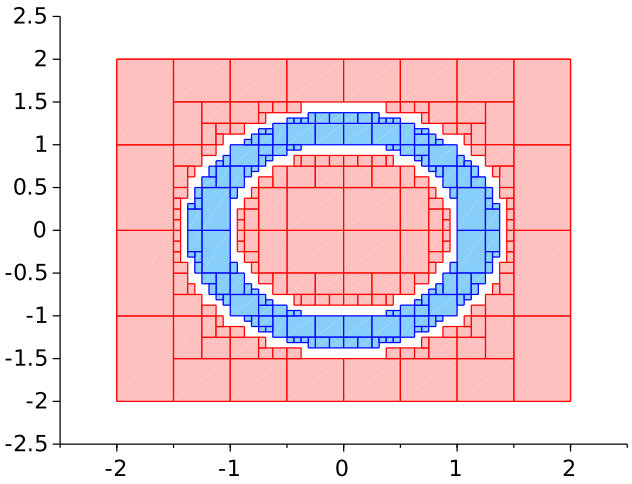

Figure 2 Paving of $x^2 + y^2 \in [1, 2]$, in blue boxes included in $\mathcal{S}$ and in red boxes which do not intersect $\mathcal{S}$.

where τ is the integration variable, $u(t)$ is the input signal of the plant model, the controller parameters are the proportional gain (K_p), the integral gain (K_i), and the derivative gain (K_d).

The choice of the tuple (K_p, K_i, K_d) is directed by the desire of obtaining a trade-off between fast response (convergence to the desired value) and good stability (no unbounded oscillation) of the control system.

3.2 Contribution

For our methods we restrict ourselves to the case where the derivative of the error is not taken into account. In this case we want to compute the set of validated PI parameters $(K_p, K_i) \in [P] \times [I]$ for a given control system. A dynamical system controlled by a PI can be written as

$$\dot{y}(t) = f(y, K_p, K_i, r), \tag{9}$$

with r the constant setpoint. Our method then consists on the computation of a paving of the set of validated PI parameters. For that, we simulate the dynamical system controlled by a PI whose parameters are in an interval. A PI controller defined by (K_p, K_i) is validated if it satisfies

$$\begin{cases} y(t_{end}) \in [r - \alpha\%, r + \alpha\%], \ 100 \geqslant \alpha > 0 \text{ (the desired setpoint is reached);} \\ \dot{y}(t_{end}) \in [-\epsilon, \epsilon], \ \epsilon > 0 \text{ (the system reached the stability zone).} \end{cases} \tag{10}$$

Algorithm 1 describes the computation of the paving of validated PI controllers. It produces two stacks of guaranteed boxes, the one of the accepted parameters and the one of the rejected parameters using a branch algorithm. For a given box $[x]$ of PI controller parameters, if the constraints defined in Equation (10) are satisfied then $[x]$ is put in the stack of guaranteed boxes, otherwise if the diameter of $[x]$ is greater than a given tolerance,

Algorithm 1 Compute the paving of PI parameters.

Require: $Stack = \emptyset$, $Stack_{accepted} = \emptyset$, $Stack_{rejected} = \emptyset$, $[x]_0 = ([P], [I])$
  Push $[x]_0$ in $Stack$
  while $Stack \neq \emptyset$ **do**
    Pop a $[x]$ from $Stack$
    Compute $y(t_{end}), \dot{y}(t_{end})$ using validated simulation of controlled plant with $[x]$
    if $(y(t_{end}) \in [r - \alpha\%, r + \alpha\%])$ && $(\dot{y}(t_{end}) \in [-\epsilon, \epsilon])$ **then**
      Push $[x]$ in $Stack_{accepted}$
    else if $(width([x]) > tol)$ && $(y(t_{end}) \ni setpoint)$ **then**
      $([x]_{left}, [x]_{right}) = Bisect([x])$
      Push $[x]_{left}$ in $Stack$
      Push $[x]_{right}$ in $Stack$
    else if $width([x]) >$ tol **then**
      Push $[x]$ in $Stack_{rejected}$
    else
      $[x]$ forgotten (cannot conclude)
    end if
  end while

$[x]$ is split into two boxes that are to be treated in the same way $[x]$ was. If $[x]$ does not meet the tolerance, it is rejected.

4 Experiments

In this section, we apply our PI tuning tool on the classic problem of cruise control. Our algorithm is developed with the IBEX library[2]. We developed a validated IVP solver inside this library, which will be released soon. This solver is the main brick of a quite classical branch and prune algorithm, already available in the library.

4.1 Modeling of the cruise-controller

To demonstrate the computation of PI controller parameters, our method is applied to the problem of automatic cruise control of a vehicle. The goal of this control is to maintain a constant vehicle speed despite external disturbances, such as change in wind or road grade. Action on the throttle has to be made if the vehicle speed is not the desired one.

4.1.1 In simplified form

The first considered example is the modeling of the cruise-controller with a simplified form of the vehicle dynamics. The vehicle to be controlled has a mass m, a velocity v, and is acted by a control force u representing the force generated at the road/tire interface. It is assumed that control on u can be done directly, that neglects the dynamics of the powertrain, tires, etc. In this simplified form is also considered that the resisting forces bv of the rolling resistance and wind drag vary linearly with v and act in the opposite direction of the vehicle's

[2] IBEX is a C++ library for constraint processing based on interval arithmetic: http://www.ibex-lib.org/.

motion. The vehicle system is described by $m\dot{v} + bv = u$ which can be rewritten as the ODE

$$\dot{v} = \frac{(u - bv)}{m}$$

The setpoint v_{set} on the speed is defined and corresponds to the particular velocity we want for the vehicle. So the error $e(t) = v_{set} - v$ and the force needed to make the vehicle go with a velocity in v_{set} is given by the PI

$$u = K_p(v_{set} - v) + K_i \int (v_{set} - v)ds,$$

and then the controlled ODE is

$$\dot{v} = \frac{(K_p(v_{set} - v) + K_i \int (v_{set} - v)dt - bv)}{m} \quad . \tag{11}$$

Let $\text{int}_{\text{err}} = \int (v_{set} - v)dt$ then ODE in Equation (11) can be written as the system

$$\begin{cases} \dot{v} = \dfrac{(K_p(v_{set} - v) + K_i \text{int}_{\text{err}} - bv)}{m} \\ \dfrac{d\text{int}_{\text{err}}}{dt} = v_{set} - v \end{cases}$$

4.1.2 With aerodynamic force

The case where aerodynamic force is not neglected is also considered here. Air particles flow over the hood of the vehicle causing the aerodynamic drag which can be modeled by

$$F_{drag} = (1/2)\rho CdAv^2$$

where ρ is the density of air, CdA is the coefficient of drag for the vehicle times the reference area, and v is the velocity of the vehicle. Here the density of the air is considered equal to 1.2041 kg/m^3 obtained with a temperature of 20°C and an atmospheric pressure of 101kPa. The consideration of aerodynamic drag leads to a non-linear differential system

$$\begin{cases} \dot{v} = \dfrac{(k_p(v_{set} - v) + k_i \text{int}_{\text{err}} - bv - (1/2)\rho CdAv^2)}{m} \\ \dfrac{d\text{int}_{\text{err}}}{dt} = v_{set} - v \end{cases}$$

4.2 Results of paving of PI parameters

Results on the application of our method on the previously described problem are now discussed. The values taken into account for the problem are $m \in [990, 1010]$, $v_{set} = 10$, $v_0 = 0$,$b = 50$, $t_{end} = 10$, $\alpha = 2\%$ and $\epsilon = 0.2$, $(1/2)\rho CdA = 0.4$, $PI_0 = ([1, 4000], [1, 120])$, $\text{tol} = 1$.

Simulation with interval parameters.

Firstly, to validate our simulation tool, we integrate the dynamical system with the simplified form of the vehicle dynamics with interval parameters for the PI controller. We start a simulation with $K_p = [900, 950]$ and $K_i = [35, 45]$, from $t = 0$ to $t = 15$. The result is shown in Figure 3. On this figure, we can see the boxes computed by each steps of integration, and that a part of them cross the limit at $t = 10$. It means that some of the parameters are not satisfying for the purpose. During the paving computation, the interval parameters would be split and two new simulations would start.

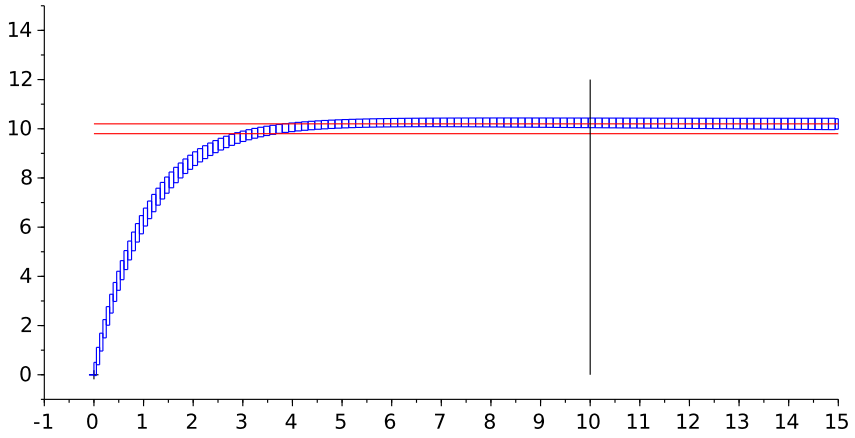

Figure 3 Simulation of the cruise-controller with linear dynamics and interval PI parameters.

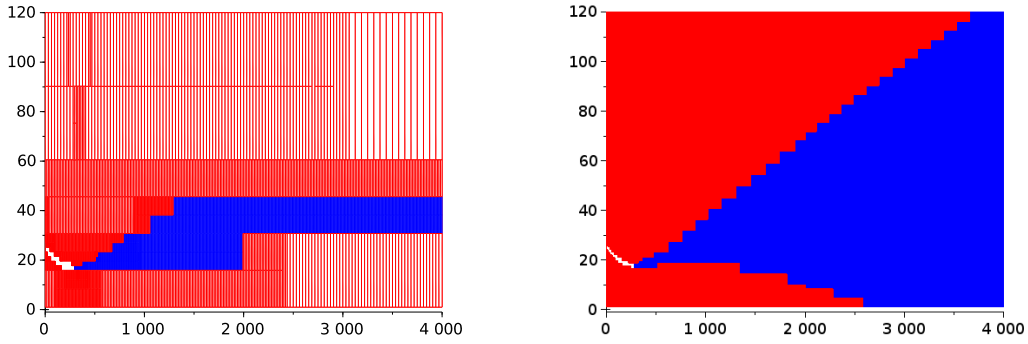

Figure 4 Paving of PI parameters for the linear (on the left) and non-linear (on the right) cruise-controller – in blue accepted and in red rejected.

Complete paving.

Figure 4 presents the result of paving for the guaranteed PI parameters for the linear and non linear modeling of the automatic cruise controller.

4.3 Response of controller along time

To verify our results, we plot the response of the cruise controller along time, that is to say the validated simulation of speed of the vehicle from 0 to 40 seconds. We recall that the setpoint is 10km/h and that we would like to attain this value at 10s. Figure 5 gathers the responses for the linear model of the cruise controller, with a validated set of parameters (a), a rejected set of parameters (b) and a set of parameters found in the literature [11] (c). The results of our tool clearly match the constraints for setpoint and stability. The latter set of parameters, if it is correct for an infinite time, does not lead to the setpoint at an acceptable response time.

Figure 6 gathers the responses for the non linear model of the cruise controller (considering the aerodynamic force), with a validated set of parameters (a), a rejected set of parameters (b)

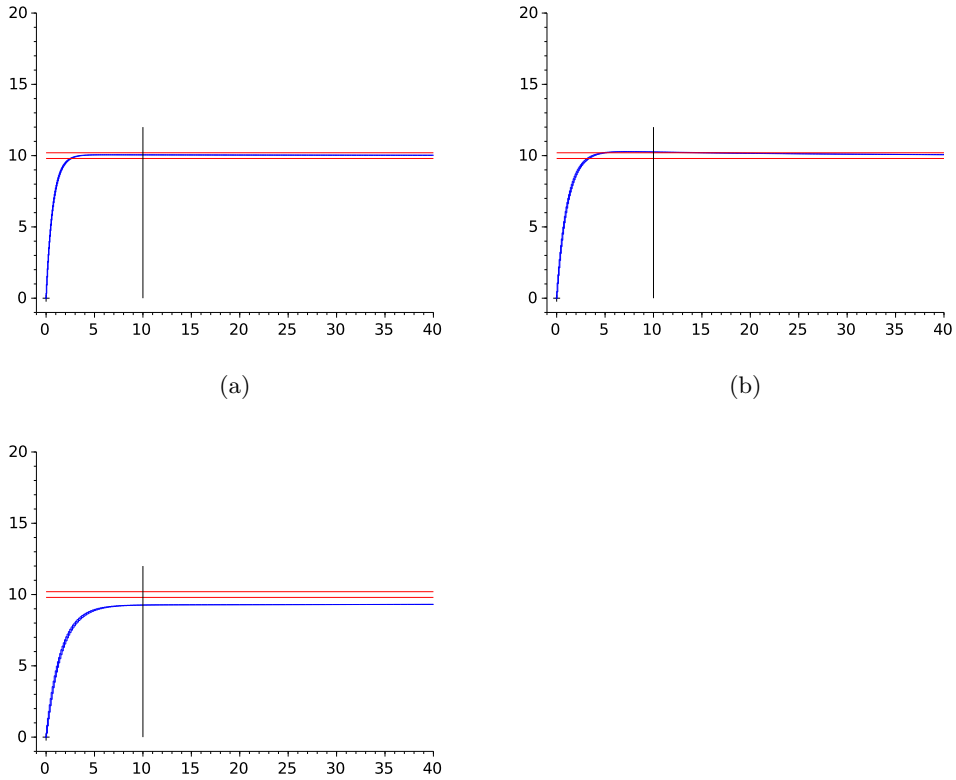

Figure 5 Response of controller for linear plant with parameters validated ($K_p = 1400$ and $K_i = 35$, in (a)), rejected ($K_p = 900$ and $K_i = 40$, in (b)) and from literature [11] ($K_p = 600$ and $K_i = 1$, in (c)) – in blue the guaranteed response, in red the supervision of the objective, and in black the deadline.

and a set of parameters found in the literature [11] (c). This set coming from an optimization process of a PID which leads to $K_d = 0$. The results of our tool clearly match the constraints for setpoint and stability. For the set of parameters coming from literature (c), the response is correct in term of objective, but this set of parameters leads to a large overshoot, and a strong instability. We can conclude from these results and the current literature that our tool can provide guarantees on sets of parameter for a controller of a non-linear system.

5 Conclusion

We presented a new method for the tuning of PI controllers. Our approach is based on the guaranteed simulation of controlled systems and on the paving of the K_p and K_i parameter space. We can then guarantee a set of parameters for which the response validates two conditions: reach the setpoint and reach the stability zone after a given time lapse. Our tool used to simulate the plant allowing us to perform non-linear integration, we applied it to the non-linear cruise controller. Our results are compared with some values found in the literature and lead to think that our approach is promising. This work may be improved in many ways. We can extend the kind of properties a PI controller must satisfy as a maximum overshoot. Moreover, we will consider PID controllers in the future.

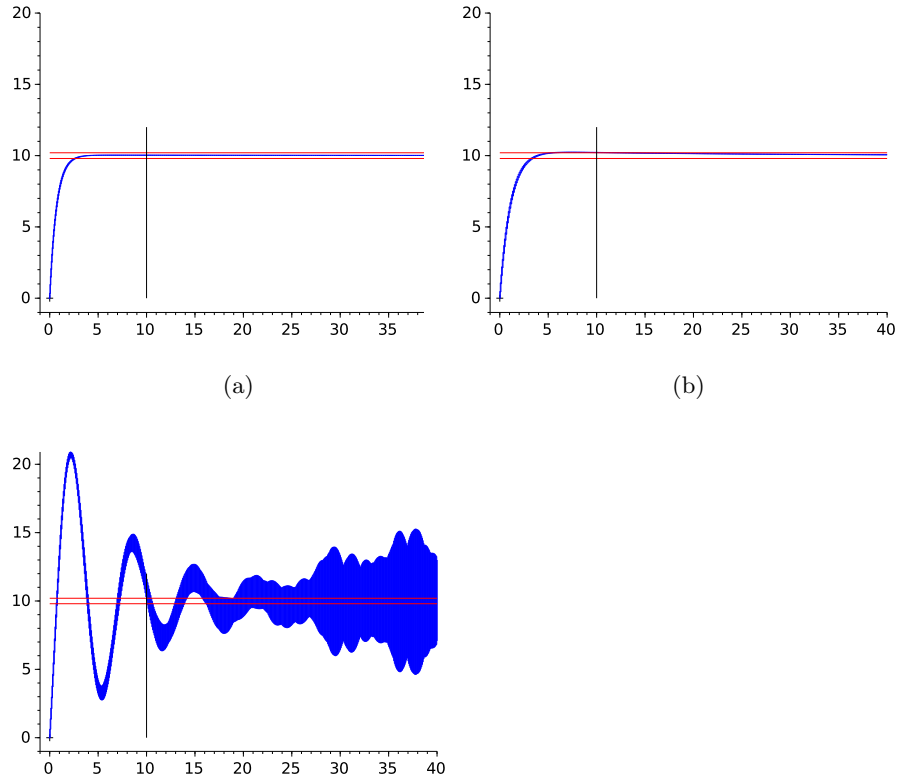

Figure 6 Response of controller for non-linear plant with parameters validated ($K_p = 1400$ and $K_i = 35$, in (a)), rejected ($K_p = 900$ and $K_i = 40$, in (b)) and from literature [11] ($K_p = 232.58$ and $K_i = 1000$ in (c)) – in blue the guaranteed response, in red the supervision of the objective, and in black the deadline.

References

1 K. J. Ãström and T. Hägglund. PID controllers: theory, design, and tuning. *Instrument Society of America, Research Triangle Park, NC*, 1995.

2 J. Alexandre dit Sandretto and A. Chapoutot. Validated Solution of Initial Value Problem for Ordinary Differential Equations based on Explicit and Implicit Runge-Kutta Schemes. Research report, ENSTA ParisTech, January 2015.

3 E. Auer and A. Rauh. Vericomp: a system to compare and assess verified IVP solvers. *Computing*, 94(2-4):163–172, 2012.

4 J. Bondia, M. Kieffer, E. Walter, J. Monreal, and J. Picó. Guaranteed tuning of PID controllers for parametric uncertain systems. In *Decision and Control*, page 2948–2953. IEEE, 2004.

5 O. Bouissou, A. Chapoutot, and A. Djoudi. Enclosing temporal evolution of dynamical systems using numerical methods. In *NASA Formal Methods*, number 7871 in LNCS, pages 108–123. Springer, 2013.

6 O. Bouissou and M. Martel. GRKLib: a Guaranteed Runge Kutta Library. In *Scientific Computing, Computer Arithmetic and Validated Numerics*, 2006.

7 V. Broida. Extrapolation des résponses indicielles apériodiques. *Automatisme*, XVI, 1969.

8 J. C. Butcher. Coefficients for the study of Runge-Kutta integration processes. *Journal of the Australian Mathematical Society*, 3:185–201, 5 1963.

9 L. H. de Figueiredo and J. Stolfi. *Self-Validated Numerical Methods and Applications*. Brazilian Mathematics Colloquium monographs. IMPA/CNPq, 1997.

10 L. Desborough and R. Miller. Increasing customer value of industrial control performance monitoring - Honeywell's experience. In *AIChE Symposium Series*, pages 169–189, 2002.

11 A. Dowling. Modeling and PID controller example - cruise control for an electric vehicle.

12 E. Hairer, Syvert P. Norsett, and G. Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems*. Springer-Verlag, 2nd edition, 2009.

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

14 Y. Lin and M. A. Stadtherr. Validated solutions of initial value problems for parametric odes. *Applied Numerical Mathematics*, 57(10):1145–1162, 2007.

15 R. J. Lohner. Enclosing the solutions of ordinary initial and boundary value problems. *Computer Arithmetic*, pages 255–286, 1987.

16 R. Moore. *Interval Analysis*. Prentice Hall, 1966.

17 N. Nedialkov, K. Jackson, and G. Corliss. Validated solutions of initial value problems for ordinary differential equations. *Appl. Math. and Comp.*, 105(1):21 – 68, 1999.

18 A. Neumaier. The wrapping effect, ellipsoid arithmetic, stability and confidence regions. *Computing Supplementum*, 9, 1993.

19 J. Vehì, I. Ferrer, and M. À. Sainz. A survey of applications of interval analysis to robust control. In *IFAC World Congress*, 2002.

20 J.G. Ziegler and N.B. Nichols. Optimum settings for automatic controllers. *Journal of dynamic systems, measurement, and control*, 115(2B):220–222, 1993.