Formal Analysis of Robustness at Model and Code Level*

Timothy Wang†
Georgia Tech
Atlanta, Georgia, USA
first.last@gatech.edu

Pierre-Loïc Garoche
ONERA – The French
Aerospace Lab
Toulouse, FRANCE
first.last@onera.fr

Pierre Roux
ONERA – The French
Aerospace Lab
Toulouse, FRANCE
first.last@onera.fr

Romain Jobredeaux
Georgia Tech
Atlanta, Georgia, USA
jobredeaux@gatech.edu

Éric Féron
Georgia Tech
Atlanta, Georgia, USA
first.last@gatech.edu

ABSTRACT

Robustness analyses play a major role in the synthesis and analysis of controllers. For control systems, robustness is a measure of the maximum tolerable model inaccuracies or perturbations that do not destabilize the system. Analyzing the robustness of a closed-loop system can be performed with multiple approaches: gain and phase margin computation for single-input single-output (SISO) linear systems, mu analysis, IQC computations, etc. However, none of these techniques consider the actual code in their analyses.

The approach presented here relies on an invariant computation on the discrete system dynamics. Using semi-definite programming (SDP) solvers, a Lyapunov-based function is synthesized that captures the vector margins of the closed-loop linear system considered. This numerical invariant expressed over the state variables of the system is compatible with code analysis and enables its validation on the code artifact.

This automatic analysis extends verification techniques focused on controller implementation, addressing validation of robustness at model and code level. It has been implemented in a tool analyzing discrete SISO systems and generating over-approximations of phase and gain margins. The analysis will be integrated in our toolchain for Simulink and Lustre models autocoding and formal analysis.

1. ANALYSIS OF CLOSED LOOP SYSTEMS

The typical development of control systems is often performed in two steps. First, control experts design the controller using a model of the plant, i.e., the system to be controlled. This design phase is supported by analyses and eventually lead to a discrete controller description to be embedded in the final hardware.

A second phase focuses on the discrete controller artifact produced by the first phase and addresses its implementation in the target embedded language. This phase is now often supported by the use of dedicated models, typically synchronous dataflow languages such as Matlab Simulink, ANSYS Scade or Lustre, and by the use of associated code generators.

In terms of analyses, the classical system level properties addressed are stability, robustness and performance.

Stability.

A stable system guarantees that a small change in the input will not produce a large change in the output. Mathematically speaking, the notion of asymptotic stability ensures that with a null input, the system converges to zero. This stability can be studied in two ways: open loop stability and closed loop stability. In the open loop setting the stability of the controller itself is studied while in the closed loop setting the complete system integrating the feedback interconnection of controller and plant is addressed.

While closed-loop stability is the main stability property of interest – that is, the controlled system will have a stable behavior – ensuring open-loop stability avoids the undesirable situation where the feedback interconnection is stable, while the controller alone is intrinsically unstable. In terms of system implementation, an open-loop stable controller has a reasonable behavior on its own, e.g., assuming only bounded input, it will provide a bounded output. This is called the bounded input bounded output (BIBO) property.

Stability properties can be assessed in different ways. A system's dynamics are expressed as transfer functions mapping inputs to outputs. These are obtained by taking the Fourier or Laplace transform of the impulse response of a system. This so-called frequency domain approach is commonly used for linear systems, along with graphical tools such as Bode plots or Nyquist diagrams. An alternative approach, temporal domain analysis, is performed on the state-space representation, and is based on Lyapunov functions. Lyapunov functions express a notion of positive energy that decreases along the trajectories of the system and captures its asymptotic stability. For linear systems, such functions

*This work has been partially supported by the following grants: ANR-INSE-2012-CAFEIN, NSF CrAVES (1135955), ARO MURI W911NF-11-1-0046, and NSF SORTIES (1446758).

†The author is currently with United Technologies Research Center

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

are usually defined using a positive definite matrix $P \succeq 0$ such that:

$$A^\intercal P A - P \prec 0, \tag{1}$$

where A is the state matrix of the system.

Code Analyses.

The Verification and Validation (V&V) of controllers is a major part of the cost of the development process for such systems. Since about a decade, the formal analysis of these control software has been the main application of formal methods in the industry [23]. However, most properties addressed, either exhaustively with formal methods or with tests, are focused on low level functionalities and hardly capture the real intended behavior of the controller and the controlled system. For example a famous application of static analysis on control software, the Astrée analyzer [7], only focused on language-related properties such as the absence of runtime errors, *i.e.*, overflow, null pointer dereferencing, division by zero, etc. These low level properties can depend on the BIBO property associated to stable systems. Therefore Astrée relies on analyses specialized for digital filters [11].

Regarding control-level properties evaluated on the implementation, the expression of stability properties using Lyapunov function requires either to be provided with the Lyapunov function or to be computed automatically. Note that, in general, the plant dynamics are no longer available at code level. However, it is important to be able to validate the actual implementation with respect to its high level system properties, such as stability.

This line of work has been developed recently, addressing either the annotation of code with a provided Lyapunov function [15] and the proof of those annotations or their computation using SDP solvers [21]. In order to express the stability of the closed-loop system, a discrete encoding of the plant dynamics is expressed along the code and enables to reason on the extended state space, which includes both controller and plant state variables. Both the Lyapunov annotations scheme [25] and the automatic computation [21] approaches have been extended to closed-loop systems [20].

Expressing Lyapunov functions at code level not only enables the expression of high level properties on code artifacts, but also provides bounds on reachable states. Indeed, stable systems with bounded inputs admit bounded outputs. The Lyapunov function characterized by the matrix $P \succ 0$ also bounds reachable states: for some α,

$$\forall x, \qquad x^\intercal P x \leq \alpha.$$

Systems Considered and Notations.

In the following we focus on linear systems, *i.e.*, a linear plant with a linear controller feedback. We also focus specifically on the discrete model artifact that represents the implementation. In order to enable the analysis of the closed-loop system, we consider a discrete version of the plant dynamics. Therefore both the controller and the plant dynamics are expressed as discrete linear systems. Without loss of generality, we denote, in the following, (A^c, B^c, C^c, D^c) and (A^p, B^p, C^p) the matrices defining the controller and plant dynamics, respectively; e denotes the input of the controller, often referred to as the error, *i.e.*, the distance to the target

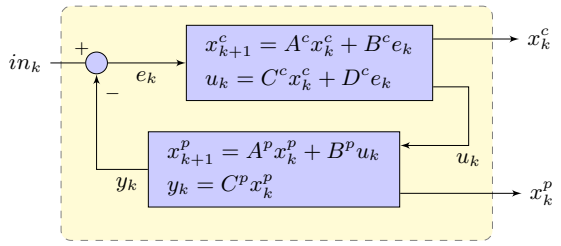

Figure 1: Closed-loop system.

reference in; u denotes both the output of the controller and the input of the plant, such as the effect of actuator commands; y denotes the measure of the plant state, *i.e.*, the feedback, e.g., as obtained by sensors:

$$\begin{cases} x^c_{k+1} = A^c x^c_k + B^c e_k \\ u_k = C^c x^c_k + D^c e_k \end{cases} \qquad \begin{cases} x^p_{k+1} = A^p x^p_k + B^p u_k \\ y_k = C^p x^p_k \end{cases} \tag{2}$$

A closed-loop representation of the system is given in Fig. 1; it is expressed over the state space defined by vectors $\begin{pmatrix} x^c & x^p \end{pmatrix}^\intercal$. Let $\mathbf{x}$ be such vectors. The error e is computed using a reference command in and the feedback y obtained from the plant.

$$e_k = in_k - y_k$$

One can consider in as the input of the closed-loop system, and $\mathbf{x}$ as its output.

Open-loop and closed-loop stability analyses performed in [20, 21], compute, respectively, positive definite matrices P^o and P^c denoting Lyapunov functions for these open- and closed-loop systems. In terms of static analysis, that is, the exhaustive analysis of the system behavior, considering actual implementation, these stability properties can be expressed as inductive invariants.

For the open-loop system, the Lyapunov function P^o is used to express a BIBO property of the controller alone: to bound reachable states x^c assuming a bounded input e:

$$\|e\|_\infty \leq 1 \implies x^{c\intercal} P^o x^c \leq 1$$

For the closed-loop system, integrating the feedback of the plant in the controller input,a similar property is expressed. For a bounded target reference in, the closed-loop system will admit only bounded reachable states $\mathbf{x}$:

$$\|in\|_\infty \leq 1 \implies \mathbf{x}^\intercal P^c \mathbf{x} \leq 1$$

These boundedness properties may seem weak to control engineers compared to the asymptotic stability properties expressed by the Lyapunov functions. However they are of extreme importance to guarantee that the implementation will behave properly, without diverging and causing runtime errors, e.g., producing numerical overflows. Once provided with a quadratic bound on reachable states using the Lyapunov function characterizing the stability of the controller, static analyses of the discrete model and the code can rely on policy iterations [12] to infer bounds on x^c and $\mathbf{x}$.

Robustness.

A robustness analysis, like a closed-loop stability analysis, requires a plant description. This system-level property evaluates "how much" the closed-loop system is stable and which kind of perturbations or uncertainty can be sustained without losing stability.

Again, this property is classical in control and part of any curriculum in an engineering degree. Bode plots or Nyquist diagrams can be used to characterize these measures depending on frequency related properties. These margins are required in the development process as a quality measure of the proposed controlled system.

However, theses margins are never analyzed or computed on the code artifact, taking into account the real implementation using floating-point arithmetic.

Goal: Hypotheses, Contributions.

The goal of this paper is therefore to continue introducing control-level concepts into the computer science community and, more specifically, the formal verification community. It addresses the automatic expression and computation of robustness properties for SISO (single-input single-output) systems over the code artifact. This class of systems is well studied and is representative enough of the realistic-size systems we are targeting, such as the NASA Transport Class Model [5], the ROSACE usecase [17], or industry-level Full Authority Digital Engine Control (FADEC).

Extending the state of the art of static analyses, this paper makes the following contributions:

- provide a computable algorithm to evaluate robustness margins of a discrete SISO system;
- compute the result automatically on the code artifact;
- take into account the floating-point semantics and its associated numerical errors;
- and validate the output of the SDP solvers used to compute the results.

Our approach has been implemented in a prototype and the approach is extensible to the MIMO (multiple-inputs multiple-outputs) setting.

Paper Organization.

The paper is structured as follows: the next section, Section 2 presents the key concepts behind margin computations for linear systems using frequency domain analyses including phase and gain margin computation. Section 3 proposes an alternate approach based on vector margin computation and presents an automated approach to margin computation. As for our past work on Lyapunov functions on code artifacts, we claim that vector margins are compatible with code level analyses. Section 4 addresses the soundness of the approach with respect to floating-point computations. Section 5 presents the application of our tooled approach to representative example of the literature. Last, Section 6 outlines an integration of the approach in a development process including margins computation and validation on model and code.

2. CLASSICAL ROBUSTNESS ANALYSIS

Beyond stability, an important property which needs to be verified is the robustness of the controller. The property of robustness is necessary in practice as there are many sources of imperfection in the feedback loop. They can include errors in modeling the plant, uncertainties in the plant that cannot be captured by the model, noises in the sensors and actuators, limitations of the controller design, i.e., not accounting for the complete range of behaviors of the system, non-linearities in the actuators, faulty actuators, etc.

The standard metric used in the industry to gauge the robustness of linear SISO controllers consists of phase and gain margins. A way to measure the phase and gain margins of a control system is by constructing a Nyquist plot. Before we describe the Nyquist plot, we introduce the following preliminaries.

2.1 Preliminaries

Consider the z-transform which maps a discrete-time signal $(x_k)_{k\in\mathbb{N}}$ to a function $X(z)$ over the complex field or the z-domain. The z-transform is the discrete-time analogue of the Laplace transform and is defined as

$$\mathcal{Z} : x \mapsto \sum_{k=0}^{\infty} x_k z^{-k}. \tag{3}$$

Consider the feedback system in Fig. 1. Let C be the controller, which has input e and output u. Let P be the plant, which has input u and output y. Applying z-transform on e and u results in a pair of functions $E(z)$ and $U(z)$. Let the ratio of $U(z)$ and $E(z)$ be $C(z)$ i.e. $C(z) := \frac{U(z)}{E(z)}$. The function $C(z)$ is a transfer function representation of the state-space system C. Likewise $P(z) := \frac{Y(z)}{U(z)}$ is the transfer function representation of P. To compute the transfer function of a linear state-space model parameterized by the matrices $A \in \mathbb{R}^{n\times n}, B \in \mathbb{R}^{n\times m}, C \in \mathbb{R}^{k\times n}, D \in \mathbb{R}^{k\times m}$, the algebraic formula

$$C\left(zI_{n\times n} - A\right)^{-1} B + D \tag{4}$$

is used. The formula in (4) can be obtained from applying the z-transform in (3) to the linear state-space system. For $m = 1$ and $k = 1$, *i.e.*, a SISO system, evaluating (4) results in a rational transfer function $G(z)$ with a number of zeros and poles. A simple example of a rational transfer function is the unit-delay function $\frac{1}{z}$, which has only one pole located at 0.

In the transfer function form, the composition of linear state-space systems can be computed using algebraic operations only. For example, the transfer function from $E(z)$ to $Y(z)$ or the loop transfer function is $L(z) := \frac{Y(z)}{E(z)} = \frac{Y(z)}{U(z)}\frac{U(z)}{E(z)} = P(z)C(z)$. Another example would be the closed-loop transfer function from $In(z) := \mathcal{Z}\left(in\right)$ to $Y(z)$ which is

$$\tilde{G}(z) = \frac{L(z)}{1 + L(z)}. \tag{5}$$

The formula in (5) can be deduced by noting that

$$\begin{aligned} Y(z) = L(z)\left(In(z) - Y(z)\right) &\implies Y(z)\left(1 + L(z)\right) = L(z)In(z) \\ &\implies \frac{Y(z)}{In(z)} = \frac{L(z)}{1 + L(z)}. \end{aligned} \tag{6}$$

2.2 Nyquist Plot and Stability Criterion

The Nyquist plot is the frequency response (magnitude and phase) of the loop transfer function to a sinusoidal input displayed using a polar coordinate system. To construct the Nyquist plot, the loop transfer function $L(z)$ is evaluated along the Nyquist contour Γ. The Nyquist contour, shown in Fig. 2, encircles the region outside of the unit disk (OUD)

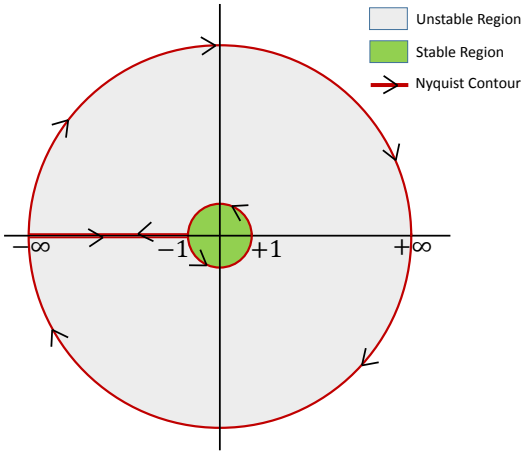

Figure 2: Nyquist contour in discrete-time.

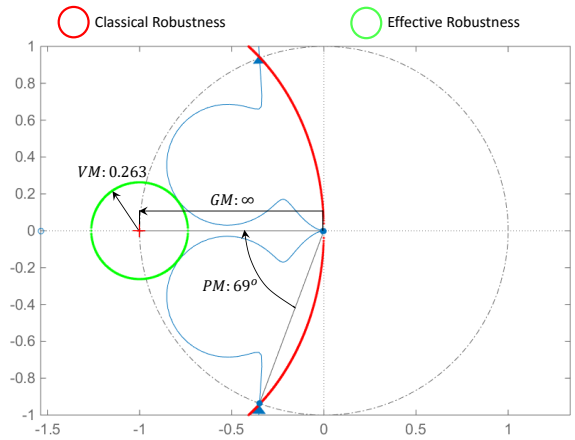

Figure 3: Classical margins versus vector margin shown on the Nyquist plot of (7).

centered at the origin. An example Nyquist plot for the loop transfer function

$$L(z) := \frac{7.552 \times 10^{-5} z^3 - 7.583 \times 10^{-5} z^2 - 7.454 \times 10^{-5} z + 7.488 \times 10^{-5}}{z^4 - 3.979 z^3 + 5.937 z^2 - 3.937 z + 0.979} \tag{7}$$

is shown in Fig. 3.

We now introduce the Nyquist stability criterion which uses the Nyquist plot to determine the closed-loop stability of the system. Let Z_i be the number of OUD zeros of $L(z)+1$ and let P_i be the number of OUD poles of $L(z)+1$. By Cauchy's principle of argument, the Nyquist plot should encircle clockwise[1] the $-1+0j$ point N_i number of times where

$$N_i = Z_i - P_i. \tag{8}$$

Using (8) and the Nyquist plot in Fig. 3, we can conclude the stability of the closed-loop system $\frac{L(z)}{1+L(z)}$ in the following way. First we know the loop transfer function $L(z)$ in (7) is stable, *i.e.*, $L(z)+1$ has 0 OUD poles, which means $P_i = 0$. Since the Nyquist plot in Fig. 3 does not encircle $-1+0j$, *i.e.*, the critical point, we can conclude that Z_i or the number of OUD zeros of $L(z)+1$ is also 0. Since OUD zeros of $L(z)+1$ are also the OUD poles of the closed-loop transfer function, we can conclude that the closed-loop system is also stable.

2.3 Phase and Gain Margins

From the Nyquist stability criterion, one can infer that a possible robustness metric would be the size of the gap between the Nyquist plot and the $-1+0j$ point. In fact, phase and gain margins are two different approximations of the "distance" from the Nyquist plot to the critical point.

The first approximation, phase margin, measures how much phase lag the system can tolerate. A phase lag of $\frac{\pi}{2}$ or 90° corresponds to a delay of a quarter of a period. Geometrically speaking, introducing a phase lag of Δ_ω in the feedback loop results in the original Nyquist plot rotated clockwise by Δ_ω i.e. $L(z) \to e^{j\Delta_\omega} L(z)$. The phase margin represents the amount of clockwise rotation that can be applied to the Nyquist plot before it hits the critical point. As shown in Fig. 3, the phase margin (PM) is precisely the clockwise angle between the point where the unit circle, centered at the origin, intersects with the Nyquist plot and $-1+0j$.

[1]Counter-clockwise encirclement counts as negative.

The second approximation, gain margin, measures how much feedback gain the system can tolerate, *i.e.*, how much one can scale up the Nyquist plot radially before it intersects with the $-1+0j$ point. As shown in Fig. 3, the gain margin (GM) is precisely $20\log_{10}\frac{1}{x}$ where x is the magnitude of the Nyquist plot at the phase angle of π. For good robustness, a typical requirement is a phase margin of at least 30° and a gain margin of at least 3db.

3. VECTOR MARGINS

Uncertainties in the feedback loop can introduce simultaneous phase lags and increases in the feedback gain. In those cases, interpreting the phase and gain margins could produce an overly optimistic view of the robustness of the feedback system. For example, a small phase lag combined with a small gain change would destabilize the system in Fig. 3, while a pure increase in gain would never do so and it would take a large phase lag alone to destabilize the system. To give a better indication of the robustness of the system, we look at the distance between $-1+0j$ and the Nyquist plot induced by the complex modulus, *i.e.*, $\min_{z\in\Gamma} |L(z)+1|$. In this paper, we call this robustness measure the vector margin. By plotting a circle of radius equal to the vector margin centered at the $-1+0j$ point, we get the effective robustness envelope in Fig. 3, which for this example, is far more pessimistic than the robustness envelope formed by the classical measures. There are several advantages to using the vector margin.

1. It is a more faithful measure of the robustness.

2. It can be translated into the time-domain and then expressed on the code as a quadratic invariant.

3. It readily extends to MIMO systems [13, 24].

3.1 Computing the Vector Margin

The vector margin can be computed by finding the inverse of the maximum modulus of the sensitivity function $S(z) := \frac{1}{1+L(z)}$ over the Nyquist contour Γ. This can be seen by

noting that

$$\min_{z\in\Gamma} |L(z)+1| = \frac{1}{\max\limits_{z\in\Gamma} \frac{1}{|L(z)+1|}} = \frac{1}{\max\limits_{z\in\Gamma} \left|\frac{1}{L(z)+1}\right|}.$$

The sensitivity function is a first-order approximation of the change in the output over the change in the input for the closed-loop system. The state-space representation of the sensitivity function $S(z) := \frac{1}{1+L(z)}$ where $L(z) := P(z)C(z)$ can be expressed in terms of the matrices which form the state-space realization of the plant $P(z)$ and the controller $C(z)$. For the example in Fig. 1, the sensitivity transfer function has the following state-space realization

$$\begin{aligned} x_{k+1} &= A_s x_k + B_s in_k \\ e_k &= C_s x_k + D_s in_k \end{aligned} \tag{9}$$

where

$$A_s := \begin{bmatrix} A_c & -B_c C_p \\ B_p C_c & A_p - B_p D_c C_p \end{bmatrix} \quad B_s := \begin{bmatrix} B_c \\ B_p D_c \end{bmatrix}$$
$$C_s := \begin{bmatrix} 0 & -C_p \end{bmatrix} \qquad D_s := \begin{bmatrix} I \end{bmatrix}. \tag{10}$$

By the application of the bounded real lemma [14, pg.821], we have the following result.

Proposition 3.1 If there exists a positive-definite matrix P and $\gamma > 0$, such that

$$x_{k+1}^\intercal P x_{k+1} - x_k^\intercal P x_k \leq \gamma^2 \|in_k\|_2^2 - \|e_k\|_2^2 \tag{11}$$

then $\max_{z\in\Gamma} |S(z)| \leq \gamma$.

The inequality in (11) is a dissipativity condition [26] and can be checked efficiently by solving a lineat matrix inequality (LMI) [4]. We have the following proposition.

Proposition 3.2 The previous inequality (11) can be written as the following LMI

$$\begin{pmatrix} A_s{}^\intercal P A_s - P + C_s{}^\intercal C_s & A_s{}^\intercal P B_s + C_s{}^\intercal D_s \\ B_s{}^\intercal P A_s + D_s{}^\intercal C_s & D_s{}^\intercal D_s + B_s{}^\intercal P B_s - \gamma^2 I \end{pmatrix} \prec 0. \tag{12}$$

By Proposition 3.2, for any $P \succ 0$ and $\gamma > 0$ satisfying (12), we have $\max_{z\in\Gamma} |S(z)| \leq \gamma$. By minimizing γ in (12), we get the vector margin $\delta = \frac{1}{\gamma}$.

Thus, summing (11) from time 0 to any time T, we get

$$x_{T+1}^\intercal P x_{T+1} - x_0^\intercal P x_0 \leq \gamma^2 \left(\sum_{k=0}^{T} \|in_k\|_2^2\right) - \sum_{k=0}^{T} \|e_k\|_2^2$$

and since P is positive definite, assuming $x_0 = 0$

$$\sum_{k=0}^{T} \|e_k\|_2^2 \leq \gamma^2 \left(\sum_{k=0}^{T} \|in_k\|_2^2\right). \tag{13}$$

3.2 Relationship with Phase and Gain Margins

While vector margins could be computed automatically on the linear system, including its implementation, the use of phase and gain margins is often required to interact with control engineers. We propose here classical projections of vector margins onto a safe approximation of their associated phase and gain margins.

Phase margins.

As explained in Sec 2, the phase margin denotes the angle between the intersection of the Nyquist plot of the transfer function with the unit circle and the point $-1+0j$.

This angle is necessary larger than the angle between the intersection of the computed safe circle of radius δ with the unit circle and the point $-1+0j$ (c.f., Fig. 3, where $\delta = VM$).

In that case a direct projection of vector margins to phase margins is

$$\phi_\delta = 2\ arcsin(\delta/2)$$

Gain margins.

Similarly a safe gain margin can be obtained by projecting the vector margin. Gain margin denotes the acceptable scale of the Nyquist plot to avoid intersection with the point $-1+0j$.

We can approximate the gain margin associated to the vector margin δ:

$$\Theta_\delta = \frac{1}{1-\delta}$$

This gain is usually reported in *dB*:

$$\Theta_\delta = 20 \cdot log10 \left(\frac{1}{1-\delta}\right)$$

4. FLOATING POINT ARITHMETIC

Up to now, we considered that every computation could be performed in the real field $\mathbb{R}$. In practice, for the sake of efficiency, all these computations will be performed using some sort of finite precision arithmetic. The most common one, floating-point arithmetic, is considered here.

We have to distinguish two fundamentally different issues regarding the use of floating-point arithmetic:

the analysis itself is performed in floating-point arithmetic, in particular the LMI (12) is solved using approximate SDP solvers, see Section 4.1;

the analyzed controller performs its computations using floating-point arithmetic rather than real numbers, this is discussed in Section 4.2.

4.1 Floating-Point Arithmetic in the Analysis

The analysis is performed by solving the LMI (12) thanks to an SDP solver. These solvers, due both to the algorithms they implement and their implementation using floating-point arithmetic, provide only approximate solutions. This means that for the returned values of P and γ^2, (12) is usually slightly not negative definite.

A simple solution is to slightly pad the LMI, replacing $M \prec 0$ by $M+\epsilon I \prec 0$. ϵ must be greater than the precision of the solver, for instance $\epsilon := 10^{-7}$ is a reasonable choice. This enables to get values of P and γ^2 actually satisfying (12) while remaining close from the optimal values.

Although, in practice, SDP solver return results satisfying the required precision ϵ, they perform all their computations using floating-point arithmetic and cannot be formally trusted. It then remains to formally check that the computed values of P and γ^2 indeed satisfy (12). This can be done by computing the matrix in (12), for instance with exact rational arithmetic, and then checking that the result is negative definite. This last check can be performed using a Cholesky decomposition. For the sake of efficiency, this

decomposition can itself be performed using floating-point arithmetic by carefully bounding the rounding errors [22]. The resulting algorithm being non trivial, it has been proved using the proof assistant Coq [18].

4.2 Floating-point Arithmetic in the Controller

The computations of the controller being performed using floating-point arithmetic, rounding errors unavoidably occur and x^c_{k+1} is not exactly equal to $A_c x^c_k + B_c e_k$.

Definition 1 *$\mathbb{F} \subset \mathbb{R}$ denotes the set of floating-point values and $\mathrm{fl}(e) \in \mathbb{F}$ the floating-point evaluation of expression e from left to right*[2].

In practice, a floating-point value $f \in \mathbb{F}$ is encoded as a mantissa $m \in \mathbb{Z}$ and an exponent $e \in \mathbb{Z}$ such that $f = m\beta^e$ where β is the radix (commonly 2). Since f is encoded with a constant number of figures[3], m lies in a bounded range. More precisely, if the mantissa m is encoded with a precision of *prec* figures: $|m| < \beta^{prec}$. To fully exploit the available precision, m and e can be chosen such that $|m| \geq \beta^{prec-1}$.

First Ignoring Over- and Underflows.

The exponent e also lies in a bounded range. When e becomes too large, an overflow occurs whereas e too small implies an underflow. We will first ignore both overflows and underflows as they only rarely happen in practice.

Thus, a real number $x \in \mathbb{R}$ can be represented by a floating-point value $f_x \in \mathbb{F}$ such that $|x - f_x| \leq \beta^{1-prec}|x|$ (or $\left(\beta^{1-prec}/2\right)|x|$ with a rounding to nearest).

Definition 2 *$\mathtt{eps} \in \mathbb{R}$ is a constant, depending on the floating-point format used, such that for all $x \in \mathbb{R}$, $f_x \in \mathbb{F}$ satisfies $|x - f_x| \leq \mathtt{eps}\,|x|$.*

Example 3 *For the IEEE754 [16] binary64 format*[4] *with rounding to nearest, we have $\mathtt{eps} = 2^{-53}$ ($\simeq 10^{-16}$).*

This constant allows to bound the rounding errors of the basic arithmetic operations.

Property 4 *For all $x, y \in \mathbb{F}$ and $\diamond \in \{+, -, \times\}$*

$$\exists\, \delta \in \mathbb{R},\ |\delta| \leq \mathtt{eps} \wedge \mathrm{fl}(x \diamond y) = (1+\delta)(x \diamond y).$$

This property is a direct consequence of Definition 2.

Back to our vector margins, what we need is not exactly the inequality (11) but rather[5]

$$\mathrm{fl}(x_{k+1})^\intercal\, P\, \mathrm{fl}(x_{k+1}) - x_k^\intercal P x_k \leq \gamma^2 \|in_k\|_2^2 - \|\mathrm{fl}(e_k)\,\|_2^2.$$

The following theorem gives a link between this and (11).

Theorem 5 *If $n' := \mathrm{sz}(x^c) + \mathrm{sz}(in) + 3$, with $\mathrm{sz}(x)$ denoting the size of the vector x, satisfies $2(n'-2)\mathtt{eps} < 1$ and*

$$x_{k+1}^T P x_{k+1} - x_k^T P x_k + \epsilon\left(\|x_k\|_2^2 + \|in\|_2^2\right) \leq \gamma^2\|in_k\|_2^2 - \|e_k\|_2^2$$

then

$$\mathrm{fl}(x_{k+1})^\intercal\, P\, \mathrm{fl}(x_{k+1}) - x_k^T P x_k \leq \gamma^2\|in_k\|_2^2 - \|\mathrm{fl}(e_k)\,\|_2^2$$

where $\epsilon := n^2\left(3\gamma_2 m''^2 + s\gamma_{n'} m'\left(2m + \gamma_{n'} m'\right)\right)$ where $s \in \mathbb{R}$ is such that $P \preceq sI$, $n := \mathrm{sz}(x) + \mathrm{sz}(in)$,
$\gamma_k = \frac{k\,\mathtt{eps}}{1 - k\,\mathtt{eps}}$, $m := \max_{i,j}\left(|A_s|_{i,j}, |B_s|_{i,j}\right)$,
$m'' := \max_{i,j}\left(1, |C_p|_{i,j}\right)$ and $m' := \max(m'_1, m'_2)$ with
$m'_1 := \max_{i,j}\left(|A_c|_{i,j}, |B_c|_{i,j}, \left(|B_c|\,|C_p|\right)_{i,j}\right)$ and
$m'_2 := \max_{i,j}\left(\left(|B_p|\,|C_c|\right)_{i,j}, \left(|B_p|\,|D_c|\right)_{i,j}, \left(|B_p|\,|D_c|\,|C_p|\right)_{i,j}\right)$.

Thus, instead of checking (12), we just have to check

$$\begin{pmatrix} A_s^\intercal P A_s - P + C_s^\intercal C_s + \epsilon I & A_s^\intercal P B_s + C_s^\intercal D_s \\ B_s^\intercal P A_s + D_s^\intercal C_s & D_s^\intercal D_s + B_s^\intercal P B_s - \gamma^2 I + \epsilon I \end{pmatrix} \prec 0.$$

In practice, $\epsilon \simeq 10^{-9}$ is small with respect to the ϵ already needed in Section 4.1 to compensate for the SDP solver precision. Thus the new condition is quite easy to satisfy.

Taking Over- and Underflows into Account.

To handle overflows, one has to prove their absence. This can be done by proving that x and e remain bounded. According to the upper left corner of the LMI (12), $x \mapsto x^\intercal P\, x$ is a Lyapunov function for the closed loop system (c.f., (1)). This means that as soon as the input in is bounded, x and e will remain bounded. Proving the absence of overflows then amounts to computing such an upper bound on the values of x and e and check that it is smaller than the largest representable floating-point value.

Handling underflows is much more tricky. Indeed, the error they induce is no longer relative but absolute and we can only prove

$$\mathrm{fl}(x_{k+1})^\intercal\, P\, \mathrm{fl}(x_{k+1}) - x_k^\intercal P x_k \leq \gamma^2\|in_k\|_2^2 - \|\mathrm{fl}(e_k)\,\|_2^2 + \eta$$

where η is a constant, depending neither on x, nor on in. Thus, instead of (13), we get

$$\sum_{k=0}^{T} \|e_k\|_2^2 \leq \gamma^2\left(\sum_{k=0}^{T}\|in_k\|_2^2\right) + (T+1)\eta.$$

However, in practice η is tiny ($\eta \simeq 10^{-300}$) so that it can remain negligible in front of the input in as long as the number T of iterations of the system remains bounded (for instance, the flight commands of a plane typically operate at 100Hz and certainly no longer that 100 hours [7], meaning less than $T := 100 \times 3600 \times 100 \simeq 10^8$ iterations).

5. EXPERIMENTS

We illustrate here the proposed approach on simple examples of the literature. A first one is a linearized model of a cruise control presented in [1, §5.11 and §10.3] while a second one is a spring-mass-damper on which we applied our previous code analyses for open and closed loop stability, applied on model and code artifacts [15, 20, 21, 25].

The method has been implemented in a backend of our Lustre and imperative code analyzer [19] for controllers, implemented in Ocaml. Considering the discrete code of the controller and a discrete version of the plant, the sensitivity system is automatically computed and the LMI synthesized. We rely on OSDP – the Ocaml SDP library – our sound in-

[2]Order of evaluation matters since floating-point operations are not associative.
[3]Bits for radix $\beta = 2$ or digits for $\beta = 10$ for instance.
[4]Usual implementation of the type `double` in C.
[5]x_{k+1} and e_k both incur floating-point computations in the controller (c.f., (9)) whereas in_k is just a real number.

terface to SDP solvers such as CSDP, Mosek or SDPA, to solve the optimization problem. Thanks to OSDP, LMI expressions are checked as seen in Section 4.1.

OSDP is available at https://cavale.enseeiht.fr/osdp/ while the following examples with automatic analyses are available at https://cavale.enseeiht.fr/robustness15/.

5.1 Cruise Control

Let us first focus on a classical example of the literature as presented in [1]. A non-linear dynamical model for the car accounting for rolling friction, aerodynamic drag and gravitational disturbance force is linearized around the equilibrium, where the force applied by the engine, balances the disturbance forces.

The resulting first order linear system is defined as

$$\frac{d(v - v_e)}{dt} = a(v - v_e) + b(u - u_e)$$

with $a := -0.0101$ and $b := 1.32$. v is the car velocity and u the throttle input.

A discrete-time version of the plant dynamics, with time step of $0.2s$ is

$$\begin{cases} x^p_{k+1} = 0.9998 * x^p_k + u_k \\ y_k = 0.0264 * x^p_k \end{cases}$$

It corresponds to the following z-expression:

$$P(z) := \frac{0.0264}{z - 0.9998}.$$

A first controller.

According to [1], a PI controller for this system with $\omega_0 = 1$ and $\zeta = 1$, the undamped natural frequency and the damping ratio of the dominant mode, is defined as

$$C_1(z) := 1.51 + 0.757\frac{0.2}{z - 1}.$$

From that z-expression, it is possible to extract the associated linear system:

$$\begin{cases} x^c_{k+1} = x^c_k + u_k \\ u_k = 0.0150x^c_k + 1.5070u_k. \end{cases}$$

The phase and gain margins computation of the closed loop system $L(z) = C_1(z)P(z)$ gives $\Theta = 34dB$ and $\phi = 75°$.

The sensitivity system is automatically built using Eq. (9):

$$\begin{pmatrix} x^c_{k+1} \\ x^p_{k+1} \end{pmatrix} = \begin{bmatrix} 1.0000 & -0.0264 \\ 0.0150 & 0.9600 \end{bmatrix} \begin{pmatrix} x^p_k \\ x^c_k \end{pmatrix} + \begin{bmatrix} 1.0000 \\ 1.5070 \end{bmatrix} in_k$$
$$e_k = in_k - 0.0264x^p.$$

And its vector margin computed using the LMI (12):

$$P = \begin{bmatrix} 0.1865 & -0.1245 \\ -0.1245 & 0.1010 \end{bmatrix} \qquad \gamma = 1.0202\,.$$

We obtain $\delta = 0.9802$.

The projection of this vector margin to conservative gain and phase margins returns:

$$\Theta_\delta = 34dB, \qquad \phi_\delta = 59°.$$

Fig. 4 presents the Nyquist plot and the vector margin.

A second controller.

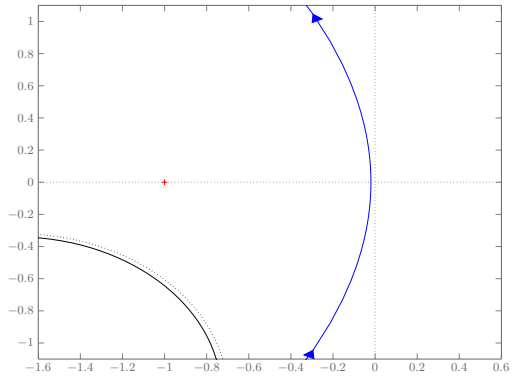

Figure 4: Nyquist plot of $P(z)C_1(z)$ with vector margin.

This PI controller is extremely stable but has low performances. Using an optimization tool, we can also design a second, higher performance controller. This process characterizes the following z-expression for the improved controller:

$$C_2(z) := \frac{2.72z^2 - 4.153z + 1.896}{z^2 - 1.844z + 0.8496}.$$

Again, its associated linear system can be expressed:

$$\begin{cases} x^c_{k+1} = \begin{bmatrix} 1.8440 & -0.8496 \\ 1.0 & 0.0 \end{bmatrix} x^c_k + \begin{bmatrix} 1.0 \\ 0.0 \end{bmatrix} u_k \\ u_k = \begin{bmatrix} 0.0366 & -0.0343 \end{bmatrix} x^c_k + \begin{bmatrix} 2.2720 \end{bmatrix} u_k. \end{cases}$$

The classical phase and gain margins of the feedback system are $\Theta = 31\,dB$ and $\phi = 84°$.

The sensitivity system is automatically built

$$\begin{pmatrix} x^c_{k+1} \\ x^p_{k+1} \end{pmatrix} = \begin{bmatrix} 1.8440 & -0.84 & -0.0264 \\ 1.0 & 0.0 & 0.0 \\ 0.0366 & -0.0343 & 0.9398 \end{bmatrix} \begin{pmatrix} x^p_k \\ x^c_k \end{pmatrix} + \begin{bmatrix} 1.0000 \\ 0.0 \\ 2.2720 \end{bmatrix} in_k$$
$$e_k = in_k - 0.0264x^p.$$

And its vector margin computed using the LMI provided in Eq. (3.2):

$$P = \begin{bmatrix} 0.1189 & -0.1002 & -0.0529 \\ -0.1002 & 0.0849 & 0.0447 \\ -0.0529 & 0.0447 & 0.0355 \end{bmatrix}, \qquad \gamma = 1.0307\,.$$

We obtain $\delta = 0.9703$.

The projection of this vector margin to conservative gain and phase margins returns:

$$\Theta_\delta = 31\,dB \qquad \phi_\delta = 58°$$

Fig. 5 presents the Nyquist plot and the vector margin.

5.2 Spring Mass Damper

We focus now on a more representative example, already analyzed in previous publications [15, 20, 21, 25]. The interested reader may refer to these to obtain more details on the system. We recall here its definition in a compact way.

The system consists in a simple spring-mass damper. Both the discrete controller and the discrete plant dynamics are expressed as linear systems.

For the sake of completeness we provide the matrices defin-

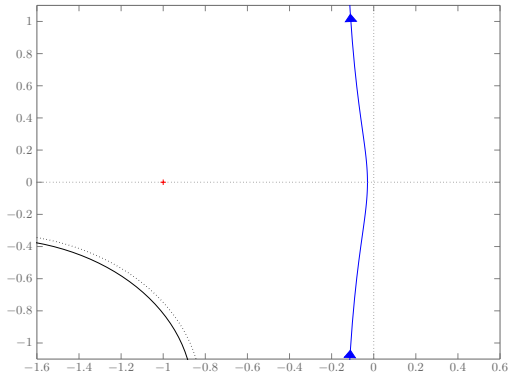

Figure 5: Nyquist plot of $P(z)C_2(z)$ with vector margin

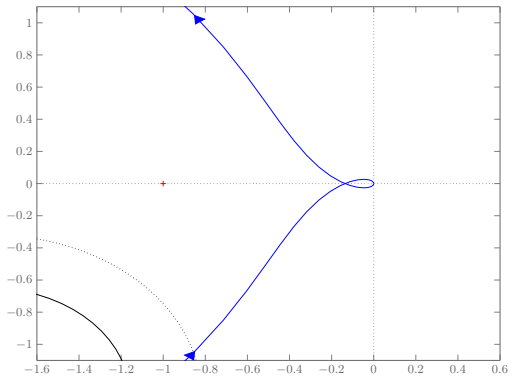

Figure 6: Nyquist plot of the spring mass damper system with vector margin

ing these system:

$$A^p := \begin{bmatrix} 1 & 0.01 \\ -0.01 & 1 \end{bmatrix}, \quad B^p := \begin{bmatrix} 0.00005 \\ 0.01 \end{bmatrix}, \quad C^p := \begin{bmatrix} 1 & 0 \end{bmatrix},$$

$$A^c := \begin{bmatrix} 0.4990 & -0.05 \\ 0.01 & 1 \end{bmatrix}, \qquad B^c := \begin{bmatrix} 1 \\ 0 \end{bmatrix},$$

$$C^c := \begin{bmatrix} 564.48 & 0 \end{bmatrix}, \qquad D^c := 1280\,.$$

The phase and gain margins computation of the closed loop system gives $\Theta = 17dB$ and $\phi = 49^\circ$.

From the discrete plant and controller description, the sensitivity system is automatically built and analyzed with the LMI (12), we obtain $\gamma = 1.4914$ and

$$P = \begin{bmatrix} 111.8330 & 88.4842 & -48.4990 & 8.8432 \\ 88.4842 & 278.5963 & -20.2482 & 6.9605 \\ -48.4990 & -20.2482 & 28.7964 & -3.7961 \\ 8.8432 & 6.9605 & -3.7961 & 0.7013 \end{bmatrix}.$$

The resulting vector margin is $\delta = 1/\gamma = 0.6705$. And its projection to conservative gain and phase margins returns:

$$\Theta_\delta = 10dB \qquad \phi_\delta = 39^\circ$$

Fig. 6 presents the Nyquist plot and the vector margin.

6. INTEGRATION IN A DEVELOPMENT PROCESS

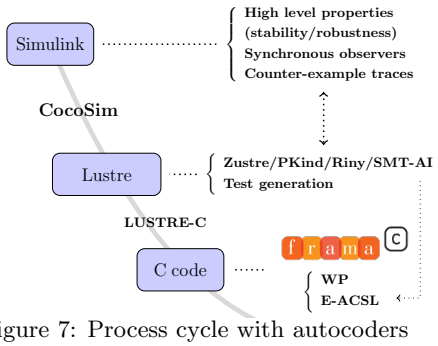

Figure 7: Process cycle with autocoders

In order to support the end-to-end formal verification of controller properties on the development artifacts, models and code, along the development process, we setup a representative toolchain, based on autocoders, to compile Simulink models into Lustre models and then C code [10, 25]. It is illustrated in Fig. 7.

This process is representative of industry standards [23] which largely rely on code generation for synchronous dataflow languages. The extraction from the discrete Simulink subset into Lustre is a one-to-one mapping of Simulink subsystems to Lustre nodes, the unit delay operator becoming a `pre` construct. Once the Lustre is obtained, it is compiled into C using the modular compilation scheme [3] used in Scade KCG compiler.

On the verification side, efficient analyses are performed at the Lustre level. Model-checking algorithms based on k-induction or property-directed reachability (PDR aka IC3) are used to address the verification of synchronous observers for safety nodes, while more advanced numerical analyses based on abstract interpretation, policy iteration and Lyapunov function based template synthesis, are focused on the controller core. These numerical analyses work on the model level representation of the systems, *i.e.*, without complex pointers and memory issues, but they consider floating-point semantics. A more detailed explanation of these interactions between solvers is presented in [6].

At the code level, state of the art analyzers are used, such as the open source Frama-C platform [8], a framework for the modular analysis of C code, that integrates a deductive method analysis based on weakest precondition computation.

We sketch here a possible use of the proposed approach within this framework. First, the controller and plant are described in Lustre (c.f., Fig 8).

The `plant` annotation enables the margin analysis which extracts the linear representation of the controller and the plant to build the sensitivity system and analyze it. The Lustre model is enriched with analysis results (c.f., Fig 9).

When generating the final C code, the code is annotated with ACSL predicates [2] relying on our linear algebra library [25]. An additional predicate encodes the dissipativity property (11). The `struct` definitions represent the internal state of each node in our modular compilation scheme. The plant internal state is declared as a ghost `struct` field within the controller own memory. This is presented in Fig. 10.

Finally the controller function is associated to an assert statement ensuring the dissipativity property after each iteration of the system dynamics, as presented in Fig. 11. It is worth noting that the plant code was introduced as ghost

Lustre

```
node spring (u:real) returns (y:real);
var xp0, xp1: real ;
let
 xp0 = 0. -> pre xp0 + 0.01 * pre xp1
     + 0.00005 * u;
 xp1 = 0. -> -0.01 * pre xp0 + pre xp1
      + 0.01 * u;
 y = xp0;
tel

--@ plant: spring
node ctl (in,y:real) returns (u:real);
var e, xc0, xc1: real ;
let
 e = in - y;
 xc0 = 0. -> 0.4990 * pre xc0 - 0.05 *
      pre xc1 - e;
 xc1 = 0. -> 0.01 * pre xc0 + pre xc1;
 u = 564.48 * xc0 + 1280. * e;
tel
```

Figure 8: Lustre model including discrete plant description, for the example of Section 5.2.

Lustre

```
--@ plant: spring
node ctl (in, y: real) returns (u:
    real);
--@ robustness/gamma: 1.4914;
--@ robustness/P: (computed P value);
```

Figure 9: Enriched model with computed properties.

C+ACSL

```
#include "acsl_matrices.h"

/*@logic matrix P = mat_of_4x4_scalar(
111.8330,  88.4842, -48.4990,  8.8432,
 88.4842, 278.5963, -20.2482,  6.9605,
-48.4990, -20.2482,  28.7964, -3.7961,
 8.8432,   6.9605,  -3.7961,  0.7013);
   logic real gamma = 1.4914; */

/*@ logic vector state(struct ctl_mem
    self) =
 vector_of_4_scalar (
   self->_reg.__ctl_3,
   self->_reg.__ctl_2,
   self->spec.plant._reg.__plant_3,
   self->spec.plant._reg.__plant_2);*/

/*@ predicate dissip(vector snxt,
    vector s, real in, real e, matrix
    P, real gamma) =
     normP(snxt, P) - normP(s, P) <=
         gamma**2 * in**2 - e**2; */

struct plant_mem {
  struct plant_reg {double __plant_2;
            double __plant_3; } _reg;
  struct _arrow_mem *ni_1; };
struct ctl_mem {
  struct ctl_reg {double __ctl_2;
           double __ctl_3; } _reg;
  struct _arrow_mem *ni_0;
  /*@ghost struct spec {
  struct plant_mem plant; } spec; */};
```

Figure 10: Header of the generated C code including node state description.

C code within the controller code, following the approach we developped in [25].

We only sketched the global approach; concerning the proof of this property at the code level, we already analyzed similar properties in PVS [15]. Furthermore the current property is simpler since it only involves the positivity of a quadratic form. State of the art SMT solvers such as Z3 [9] should be able to discharge the generated proof objectives without requiring the use of proof assistants.

7. CONCLUSION

This paper had two targets: verification engineers interested in the formal analysis of controller implementations on the one hand, and control engineers on the other hand.

For the first target, this paper provides a gentle introduction to robustness analyses of discrete dynamical SISO systems. It covers the classical definition of stability, phase and gain margin computation and their vector margin counterpart. The method proposed could be used by computer scientists and would enable the generation of conservative phase and gain margins on the code artifact.

For the second target, control engineers, this paper illustrates the fact that the notion of vector margin and its associated LMI characterization is compatible with implementation analysis and should be used instead of frequency based analyses. It also emphasizes the usually forgotten aspect of floating-point imprecision.

Apart from these pedagogical aspects, this paper provides the first automatic tool that analyses robustness of discrete closed-loop systems. It is compatible with analyses such as [19] that extract from the code the linear dynamics and enable its analysis. Our approach instruments the vector margin computation and extends its definition by taking into account floating-point semantics. It also addresses the unsoundness of the SDP solvers implemented with floats by checking *a posteriori* the soundness of the results.

In terms of perspectives, the integration of the approach in our toolchain and its application on more general systems is our next target. As opposed to phase and gain margins, vector margins are also more suited to be extended to evaluate robustness of MIMO systems. Another possibility would be to combine the current robustness characterization on more complex controllers including non- linear ones, *i.e.*, saturated controllers, piecewise systems or LPV controllers.

C+ACSL

```
void ctl_step (double in, double y,
               double (*u),
               struct ctl_mem *self) {
_Bool __ctl_1;
double e; double xc0; double xc1;
_arrow_step(1,0,&__ctl_1,self->ni_0);
if (__ctl_1) { xc1 = 0.; } else {
  xc1 = ((0.01 * self->_reg.__ctl_3) +
       self->_reg.__ctl_2); }
e = (in - y);
if (__ctl_1) { xc0 = 0.; } else {
  xc0 = (((0.499 * self->_reg.__ctl_3)
       - (0.05 * self->_reg.__ctl_2))
      - e); }
*u = ((564.48 * xc0) + (1280. * e));
self->_reg.__ctl_3 = xc0;
self->_reg.__ctl_2 = xc1;
/*@ghost
_Bool __plant_1;
double xp0; double xp1;
//plant - restricted to state update
_arrow_step (1, 0, &__plant_1,
    self->spec.plant.ni_1);
if (__plant_1) { xp0 = 0.; } else {
  xp0 = ((self->spec.plant._reg.
      __plant_3 + (0.01 * self->spec.
      plant._reg.__plant_2)) + (5e-05
      * *u)); }
if (__plant_1) { xp1 = 0.; } else {
  xp1 = ((((- 0.01) * self->spec.plant
      ._reg.__plant_3) + self->spec.
      plant._reg.__plant_2) + (0.01 *
      *u)); }
self->spec.plant._reg.__plant_3 = xp0;
self->spec.plant._reg.__plant_2 = xp1;
*/
//@assert dissip(state(self), \old(
    state(self)), in, e, P, gamma);
return;
}
```

Figure 11: Transfer function of the controller, including the plant description as annotation and the dissipativity property as function contract.

8. REFERENCES

[1] K. J. Astrom and R. M. Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, Princeton, NJ, USA, 2008.

[2] P. Baudin, J.-C. Filliâtre, C. Marché, B. Monate, Y. Moy, and V. Prevosto. Acsl: Ansi/iso c specification language. version 1.7. http://frama-c.com/download/acsl.pdf.

[3] D. Biernacki, J.-L. Colaćco, G. Hamon, and M. Pouzet. Clock-directed modular code generation for synchronous data-flow languages. In *LCTES*, 2008.

[4] S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan. *Linear Matrix Inequalities in System and Control Theory*. SIAM, 1994.

[5] G. Brat, D. Bushnell, M. Davies, D. Giannakopoulou, F. Howar, and T. Kahsai. Verifying the safety of a flight-critical system. In *FM*, 2015.

[6] A. Champion, R. Delmas, M. Dierkes, P. Garoche, R. Jobredeaux, and P. Roux. Formal methods for the analysis of critical control systems models: Combining non-linear and linear analyses. In *FMICS*, 2013.

[7] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. The Astrée analyzer. In *ESOP*, 2005.

[8] P. Cuoq, F. Kirchner, N. Kosmatov, V. Prevosto, J. Signoles, and B. Yakobowski. Frama-C: A software analysis perspective. SEFM, 2012.

[9] L. De Moura and N. Bjørner. Z3: An efficient smt solver. In *TACAS*, 2008.

[10] A. Dieumegard, P. Garoche, T. Kahsai, A. Taillar, and X. Thirioux. Compilation of synchronous observers as code contracts. In *SAC*, 2015.

[11] J. Feret. Static analysis of digital filters. In *ESOP*, 2004.

[12] T. Gawlitza, H. Seidl, A. Adjé, S. Gaubert, and E. Goubault. Abstract interpretation meets convex optimization. *J. Symb. Comput.*, 2012.

[13] K. Glover, G. Vinnicombe, and G. Papageorgiou. Guaranteed multi-loop stability margins and the gap metric. In *CDC*, 2000.

[14] W. M. Haddad and V. Chellaboina. *Nonlinear Dynamical Systems and Control: A Lyapunov-based Appr.* Princeton University Press, 2008.

[15] H. Herencia-Zapana, R. Jobredeaux, S. Owre, P.-L. Garoche, E. Feron, G. Perez, and P. Ascariz. Pvs linear algebra libraries for verification of control software algorithms in C/ACSL. In *NFM*, 2012.

[16] IEEE. Standard for Floating-Point Arithmetic. *IEEE Standard 754-2008*, 2008.

[17] C. Pagetti, D. Saussié, R. Gratia, E. Noulard, and P. Siron. The ROSACE case study: From simulink specification to multi/many-core execution. In *RTAS*, 2014.

[18] P. Roux. Formal proofs of rounding error bounds. *Journal of Automated Reasoning*, 2015.

[19] P. Roux and P.-L. Garoche. Integrating policy iterations in abstract interpreters. In *ATVA*, 2013.

[20] P. Roux, R. Jobredeaux, and P.-L. Garoche. Closed loop analysis of control command software. In *HSCC*, 2015.

[21] P. Roux, R. Jobredeaux, P.-L. Garoche, and E. Féron. A generic ellipsoid abstract domain for linear time invariant systems. In *HSCC*, 2012.

[22] S. M. Rump. Verification of positive definiteness. *BIT Numerical Mathematics*, 2006.

[23] J. Souyris, V. Wiels, D. Delmas, and H. Delseny. Formal verification of avionics software products. In *FM*, 2009.

[24] G. Vinnicombe. *Uncertainty and Feedback: H [infinity] Loop-shaping and the [nu]-gap Metric*. World Scientific, 2001.

[25] T. Wang, R. Jobredeaux, H. Herencia, P.-L. Garoche, A. Dieumegard, E. Feron, and M. Pantel. From design to implementation: An automated, credible autocoding chain for control systems. In *Advances in Control System Technology for Aerospace Applications*. 2016.

[26] J. C. Willems. Dissipative dynamical systems part i: General theory. *Archive for rational mechanics and analysis*, 1972.