Electronic Communications of the EASST
Volume X (2013)

Proceedings of the Automated Verification of Critical Systems (AVoCS 2013)

Preservation of Lyapunov-Theoretic Proofs: From Real to Floating-Point Numbers

Vivien Maisonneuve

14 pages

Guest Editors: Steve Schneider, Helen Treharne
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Preservation of Lyapunov-Theoretic Proofs: From Real to Floating-Point Numbers

Vivien Maisonneuve[1]

[1] vivien.maisonneuve@cri.mines-paristech.fr
CRI, Mathématiques et Systèmes
MINES ParisTech
Fontainebleau, France

Abstract: In [Fer10] Feron presents how Lyapunov-theoretic proofs of stability can be migrated toward computer-readable and verifiable certificates of control software behavior by relying of Floyd's and Hoare's proof system.

We address the issue of errors resulting from the use of floating-point arithmetic: we present an approach to translate Feron's proof invariants on real arithmetic to similar invariants on floating-point numbers and show how our methodology applies to prove stability, thus allowing to verify whether the stability invariant still holds when the controller is implemented.

We study in details the open-loop system of Feron's paper. We also use the same approach for Feron's closed-loop system, but the constraints are too tights to show stability in this second case: more leeway should be introduced in the proof on real numbers, otherwise the resulting system might be unstable.

Keywords: Lyapunov stability, proof preservation, ellipse, floating-point, IEEE 754, rounding errors, control system

1 Introduction

Provable stability constitutes an essential attribute of control systems, especially when human safety is involved, as in medical or aeronautical domains. Motivated by such applications, there exist many theorems to support system stability and performance under various assumptions and in various settings.

Stability criteria apply to a class of dynamical systems for which a stability proof is needed. Modern systems developments, such as adaptative control technologies, rely on robust stability and performance criteria as the primary justification for their relevance to safety-critical control applications. Lyapunov's stability theory plays a critical role in that regard.

The low-level software implementation of a control law can be inspected by analysis tools available to support the development of safety-critical computer programs. The simplest program analysis techniques consist of performing several simulations, sometimes including a software or hardware representation of the controlled system in the loop. However, simulations provide information about a large but only finite number of system behaviors. More advanced methods include model checking and abstract interpretation, e.g. using Astrée [CCF+13]. In these methods, inputs are computer programs and outputs are certificates of proper program behavior along

the chosen criterion. Another possibility is to use theorem-proving techniques, supported by computer tools such as Coq, Isabelle or PVS [Coq13, Pau13, PVS13]. These proof assistants can be used to establish properties of programs and more general mathematical constructs. Model checking, abstract interpretation, and theorem-proving tools are all used to verify safety-critical applications.

In [Fer10], Feron investigates how control-system domain knowledge and, in particular, Lyapunov-theoretic proofs of stability and performance, can be migrated toward computer-readable and verifiable certificates of control software behavior by relying on Floyd's and Hoare's proof system [Pel01], applied to MATLAB pseudocode (Sections 2 and 3). His article focuses on using this framework to support such properties, namely, bounded-input, bounded-state stability, as they apply to control-system code implementations. But errors resulting from the use of floating-point arithmetic are not addressed.

In this paper, we present an approach to translate Lyapunov-theoretic stability proof invariants on pseudocode with real arithmetic, as provided in Feron's article, to similar invariants on machine code that take into account rounding errors introduced by floating-point arithmetic. We use them to verify whether stability conditions still hold, in which case system stability with floating-point numbers can be established.

This document is organized as follows. The next section describes a second-order dynamical system example, with the corresponding controller. Next, the analysis of the open-loop controller is presented, followed by its translation to floating-point arithmetic. Then we discuss the case of the closed-loop system. The document concludes with a discussion of the generality of this successful approach.

2 Motivating Example

We consider the first system described in the article of Feron [Fer10]. It is a dynamical system composed of a single mass and a single spring shown in Figure 1.

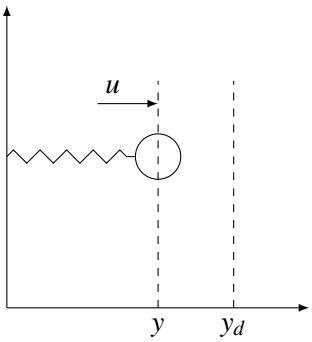

Figure 1: Mass-spring system

The position input y of the mass is available for feedback control. The signal y_d is the reference signal, that is, the desired position to be followed by the mass.

A discrete-time MATLAB implementation of the controller, using real numbers, is provided

in [Fer10]. The source code is shown below:

```
1  Ac = [0.4990, -0.0500; 0.0100, 1.0000];
2  Bc = [1; 0];
3  Cc = [564.48, 0];
4  Dc = -1280;
5  xc = zeros(2, 1);
6  receive(y, 2); receive(yd, 3);
7  while (1)
8     yc = max(min(y - yd, 1), -1);
9     u = Cc*xc + Dc*yc;
10    xc = Ac*xc + Bc*yc;
11    send(u, 1);
12    receive(y, 2);
13    receive(yd, 3);
14 end
```

Apart from the mechanical system state observation y and the desired system output y_d, variables in this code are:

- $x_c = \binom{xc_1}{xc_2} \in \mathbb{R}^2$ is the discrete-time controller state;
- $y_c \in [-1,1]$ is the bounded output tracking error, i.e. the input $y - y_d$ passed through a saturation function to avoid variable overflow in the controller;
- $u \in \mathbb{R}$ is the mechanical system input, i.e. the action to be performed according to the controller.

Constants A_c, B_c, C_c and D_c are the discrete-time controller state, input, output and feedthrough matrices. The commands `send` and `receive` are basically I/O: they respectively send and receive data given in the commands first argument through a specific channel given by the commands second argument.

3 Open-Loop Stability Proof

The stability proof of this system relies on Lyapunov theory. In simple terms, a system is *Lyapunov stable* if all states x_c reachable from an initial starting state belonging to a bounded neighborhood V of an equilibrium point x_e remain in V.

Lyapunov theory provides constraints that must be satisfied by such a V. On linear systems, they are equations that can be solved using linear matrix inequalities [BEFB94]. Commonly, V is an ellipsoid.

In this case, to prove Lyapunov stability, we need to show that at any time, x_c belongs to the set $\mathscr{E}_P$ chosen by Feron according to Lyapunov's theory:

$$\mathscr{E}_P = \{x \in \mathbb{R}^2 \,|\, x^T \cdot P \cdot x \leq 1\}, \qquad P = 10^{-3} \begin{pmatrix} 0.6742 & 0.0428 \\ 0.0428 & 2.4651 \end{pmatrix}.$$

using $\mathscr{E}_P$ as the stability neighborhood V.

This set is a full ellipse, centered around 0 and slightly slanted:

$$x_c \in \mathscr{E}_P \iff 0.6742x_{c_1}^2 + 0.0856x_{c_1}x_{c_2} + 2.4651x_{c_2}^2 \leq 1000.$$

$\mathscr{E}_P$ is drawn in Figure 2.

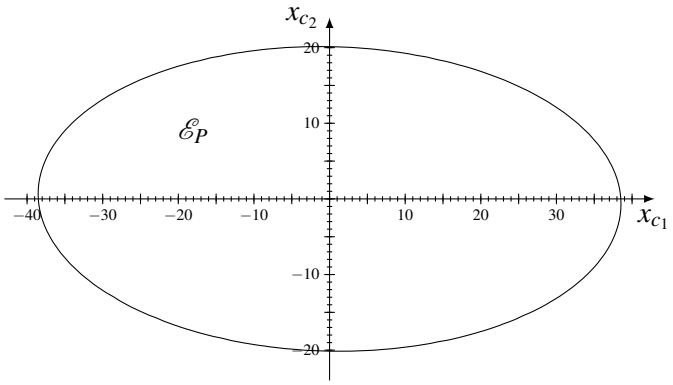

Figure 2: The stability domain $\mathscr{E}_P$

A stability proof of the controller is provided in [Fer10], using Floyd-Hoare program annotation technique [Pel01]: each program instruction comes with an invariant. The program annotated by Feron is reproduced below. We note $z_c = \begin{pmatrix} x_{c_1} \\ x_{c_2} \\ y_c \end{pmatrix}$.

```
5  xc = zeros(2,1);
   % x_c ∈ E_P
6  receive(y, 2); receive(yd, 3);
   % x_c ∈ E_P
7  while (1)
       % x_c ∈ E_P
8      yc = max(min(y - yd, 1), -1);
       % x_c ∈ E_P,   y_c^2 ≤ 1
       % z_c ∈ E_{Q_μ},   Q_μ = [μP  0_{2×1}; 0_{1×2}  1−μ],   μ = 0.9991
9      u = Cc*xc + Dc*yc;
       % z_c ∈ E_{Q_μ},   u^2 ≤ (C_c  D_c)·Q_μ^{-1}·(C_c  D_c)^{-1}
10     xc = Ac*xc + Bc*yc;
       % x_c ∈ E_P̃,   P̃ = [(A_c  B_c)·Q_μ^{-1}·(A_c  B_c)^T]^{-1},   u^2 ≤ (C_c  D_c)·Q_μ^{-1}·(C_c  D_c)^{-1}
11     send(u, 1);
       % x_c ∈ E_P̃
12     receive(y, 2);
       % x_c ∈ E_P̃
13     receive(yd, 3);
       % x_c ∈ E_P̃
```

```
    % x_c ∈ E_P
14  end
```

Most of the proof relies on algebraic arguments. For example, the invariant loosening that follows Line 8:

```
8       yc = max(min(y - yd, 1), -1);
        % x_c ∈ E_P,    y_c^2 ≤ 1
        % z_c ∈ E_Qμ,   Q_μ = [μP 0_2x1; 0_1x2 1-μ],   μ = 0.9991
```

means

$$x_c \in \mathscr{E}_P \wedge y_c^2 \leq 1 \Longrightarrow z_c \in \mathscr{E}_{Q_\mu} \text{ with } Q_\mu = \begin{pmatrix} \mu P & 0_{2\times 1} \\ 0_{1\times 2} & 1-\mu \end{pmatrix} \text{ and } \mu = 0.9991.$$

where $z_c = \begin{pmatrix} x_c \\ y_c \end{pmatrix}$ as defined above.

The correctness of this assertion stems from the fact that, given any value of $\mu \in [0,1]$, the domain $\mathscr{E}_{Q_\mu}$ is a solid ellipsoid, centered around 0, and whose intersection with the plane $y_c = 1$ is equal to $\mathscr{E}_P$. Consequently, the solid bounded cylinder $\mathscr{C} = \{z_c \,|\, x_c \in \mathscr{E}_P \wedge y_c^2 \leq 1\}$ is included within $\mathscr{E}_{Q_\mu}$ (Figure 3).

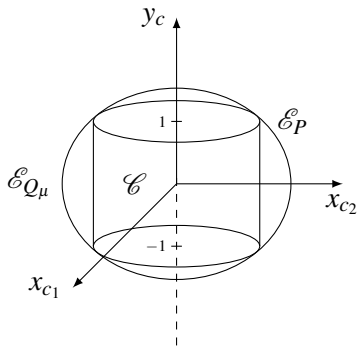

Figure 3: Inclusion of $\mathscr{E}_P$ within $\mathscr{E}_{Q_\mu}$

The following invariants

```
        % z_c ∈ E_Qμ,   Q_μ = [μP 0_2x1; 0_1x2 1-μ],   μ = 0.9991
9       u = Cc*xc + Dc*yc;
        % z_c ∈ E_Qμ,   u^2 ≤ (C_c  D_c)·Q_μ^-1·(C_c  D_c)^-1
10      xc = Ac*xc + Bc*yc;
        % x_c ∈ E_P̃,   P̃ = [(A_c  B_c)·Q_μ^-1·(A_c  B_c)^T]^-1,   u^2 ≤ (C_c  D_c)·Q_μ^-1·(C_c  D_c)^-1
```

also rely on algebraic arguments and theorems.

Other invariants are trivial. Finally, only the very last loosening

```
    % x_c ∈ E_P̃
    % x_c ∈ E_P
```

i.e. $\mathscr{E}_{\tilde{P}} \subset \mathscr{E}_P$, that "closes" the loop, is not purely algebraic since its validity relies on the numerical parameters A_c, B_c, C_c, D_c. This assertion needs to be checked to ensure the correctness of the proof statements. This can be done either numerically, or algebraically for at most two-dimensional systems like this one.

This stability proof has been checked with Mathematica [Wol13] and is available online[1].

4 Stability Proof Scheme with Floating-Point Numbers

We would like to check that the stability proof still holds on a controller implemented with floating-point numbers. When using floating-point arithmetic, the values of constants are slightly altered and calculations are likely to produce rounding errors.

In absolute terms, it is impossible to switch from real to floating-point numbers without affecting the behavior of the controller. Thus, the stability proof cannot be preserved in the general case. On the other hand, the program still can be stable if rounding errors are small enough and the final inclusion $\mathscr{E}_{\tilde{P}} \subset \mathscr{E}_P$ leaves them enough room. We study how proof invariants can be tweaked so that they correspond to a floating-point semantic. Our goal is to derive from the proof scheme for real numbers a proof scheme suited for floating-point arithmetic, whose correctness, although not guaranteed, can be checked as easily.

In this section, we use the IEEE 754 floating-point standard [IEE08] encoded on 64 bits[2] as most of today's floating-point units do. In this standard, both addition and multiplication are correctly rounded depending on the active rounding mode, which allows to bound the rounding error depending on the values of operands. The case of other numeric representations is discussed in Section 6.

4.1 Converting Constants

The first step is to convert real constants

```
Ac = [0.4990, -0.0500; 0.0100, 1.0000];
Bc = [1; 0];
Cc = [564.48, 0];
Dc = -1280;
```

to floating-point constants, assuming rounding to nearest value:

```
Acf = [0.4989999999999999991118215802998747676610946655273437 5,
      -0.0500000000000000002775557561562891351059079170227050781 25;
      0.0100000000000000000208166817117216851329430937767028808593 75,
      1.0000]
Bcf = [1; 0];
Ccf = [564.4800000000000181898940354856475830078125, 0]
```

[1] Mathematica source file is available at: http://www.cri.ensmp.fr/people/maisonneuve/lyafloat/resources/lyafloat_stability.nb, and the corresponding PDF file at: http://www.cri.ensmp.fr/people/maisonneuve/lyafloat/resources/lyafloat_stability.pdf.

[2] The procedure that follows would be exactly the same with 32-bit floating-point numbers, only with different numerical results.

```
4  Dcf = -1280
```

Theses matrices A_c^f, B_c^f, C_c^f and D_c^f will be used instead of A_c, B_c, C_c, D_c in the sequel of the proof.

Apart from constants, the proof scheme for the first part of the program is unchanged:

```
5  xc = zeros(2,1);
   % x_c ∈ E_P
6  receive(y, 2); receive(yd, 3);
   % x_c ∈ E_P
7  while (1)
       % x_c ∈ E_P
8      yc = max(min(y - yd, 1), -1);
       % x_c ∈ E_P,   y_c^2 ≤ 1
       % z_c ∈ E_{Q_μ},   Q_μ = ( μP  0_{2×1} ; 0_{1×2}  1−μ ),   μ = 0.9991
9      u = Cc*xc + Dc*yc;
```

4.2 Invariant on u

Then, we consider the next instruction in the original proof scheme:

```
       % z_c ∈ E_{Q_μ},   Q_μ = ( μP  0_{2×1} ; 0_{1×2}  1−μ ),   μ = 0.9991
9      u = Cc*xc + Dc*yc;
       % z_c ∈ E_{Q_μ},   u^2 ≤ (C_c  D_c) · Q_μ^{-1} · (C_c  D_c)^{-1}
```

First of all, matrices C_c and D_c must be replaced by their floating-point counterparts C_c^f and D_c^f both in the program instruction Line 9 and the ensuing invariant. This invariant relies only on algebraic arguments and does not depend on the values in the matrices, it still holds considering exact arithmetic operations. But this is not sufficient: indeed, this instruction is a sum of matrix multiplications, i.e. a set of additions and multiplications on floating-point numbers that yield rounding errors.

We can notice that entering this instruction, the values of matrices C_c^f, D_c^f and $\mathscr{E}_{Q_\mu}$ are known, and the values of x_c and y_c are bounded by the precondition

$$z_c = \begin{pmatrix} x_c \\ y_c \end{pmatrix} \in \mathscr{E}_{Q_\mu},$$

that is

$$0.000673593x_{c_1}^2 + 0.000085523x_{c_1}x_{c_2} + 0.00246288x_{c_2}^2 + 0.9991y_c^2 \leq 1. \tag{1}$$

From (1), we deduce:

$$\begin{cases} |x_{c_1}| & \leq & 3\cdot 10^5\sqrt{\frac{13695}{829322227639}} & < & 38.5515 \\ |x_{c_2}| & \leq & 10^5\sqrt{\frac{33710}{829322227639}} & < & 20.1614 \\ |y_c| & \leq & \frac{100}{\sqrt{9991}} & < & 1.00046 \end{cases} \tag{2}$$

Here we are able to find algebraic solutions, but this may be impossible with ellipsoids of higher dimension. Still, we would be able to find bounds using numerical methods.

In floating-point arithmetic, rounding errors created by addition and multiplication operators can be bounded when the operands are known or bounded by (2), provided that overflow, underflow, and denormalized numbers do not occur [Gol91, Hig02].

Here, we need to compute

$$C_c^f x_c + D_c^f y_c = C_{c_{(0,0)}}^f x_{c_1} + D_c^f y_c$$

where all values in the right-hand term are known or bounded. Thus, a constant can be computed that bounds the absolute rounding error created when computing u. We used Rangelab [Mar11], a static analysis tool to automatically validate the accuracy of floating-point or fixed-point computations, to compute an upper bound e for the error term[3]:

$$e = 5.90 \cdot 10^{-12}$$

This way, starting from the algebraic result obtained on real numbers

$$|u| \leq \sqrt{(C_c \; D_c) \cdot Q_\mu^{-1} \cdot (C_c \; D_c)^T}$$

we can ensure that with floating-point numbers, the inequality holds:

$$|u| \leq U_f = \sqrt{(C_c^f \; D_c^f) \cdot Q_\mu^{-1} \cdot (C_c^f \; D_c^f)^T} + e$$

which leads to the invariants:

$$\% \;\; z_c \in \mathscr{E}_{Q_\mu}, \quad Q_\mu = \begin{pmatrix} \mu P & 0_{2\times 1} \\ 0_{1\times 2} & 1-\mu \end{pmatrix}, \quad \mu = 0.9991$$

```
9   u = Cc*xc + Dc*yc;
```

$$\% \;\; z_c \in \mathscr{E}_{Q_\mu}, \quad u^2 \leq U_f^2$$

4.3 Invariant on x_c

The next instruction, considering constant changes, is:

```
10  xc = Acf*xc + Bcf*yc;
```

$$\% \;\; x_c \in \mathscr{E}_{\tilde{P}_f}, \quad \tilde{P}_f = \left[(A_c^f \;\; B_c^f) \cdot Q_\mu^{-1} \cdot (A_c^f \;\; B_c^f)^T\right]^{-1}, \quad u^2 \leq (C_c \;\; D_c) \cdot Q_\mu^{-1} \cdot (C_c \;\; D_c)^{-1}$$

where $\tilde{P}_f$ is defined the same way $\tilde{P}$ is, using floating-point terms A_c^f, B_c^f instead of the real-valued counterparts A_c, B_c. Again, this invariant holds independently of matrices values.

Here, we compute

$$\begin{cases} A_{c_{(0,0)}}^f x_{c_1} + A_{c_{(0,1)}}^f x_{c_2} + y_c \\ A_{c_{(1,0)}}^f x_{c_1} + x_{c_2} \end{cases}$$

[3] Rangelab source file is available at: http://www.cri.ensmp.fr/people/maisonneuve/lyafloat/resources/lyafloat_fp.m.

Using the same method as above, absolute rounding errors introduced by floating-point operations can be bounded by constants

$$e_1 = 7.42 \cdot 10^{-15}, \quad e_2 = 3.62 \cdot 10^{-15}$$

These constants must be taken into account in the postcondition. Also, invariant on u must be replaced as previously. Then the postcondition can be replaced by

$$\% \;\; x_c \in \mathscr{E}^f_{\tilde{P}_f}, \quad u^2 \leq U_f^2$$

where $\mathscr{E}^f_{\tilde{P}_f}$ is an ellipse that includes $\mathscr{E}_{\tilde{P}_f}$ plus the rounding error terms (see Figure 4). Replacing the ellipse in the postcondition by another ellipse has the advantage of introducing little change in the stability proof sketch (instead of using a different domain, which would involve using different theorems), which can greatly facilitate tweaking the rest of the proof in longer codes. Formally, $\mathscr{E}^f_{\tilde{P}_f}$ must satisfy:

$$\forall x_c \in \mathscr{E}_{\tilde{P}_f}, \forall x'_c \in \mathbb{R}^2, |x'_{c_1} - x_{c_1}| \leq e_1 \wedge |x'_{c_2} - x_{c_2}| \leq e_2 \Longrightarrow x'_c \in \mathscr{E}^f_{\tilde{P}_f} \tag{3}$$

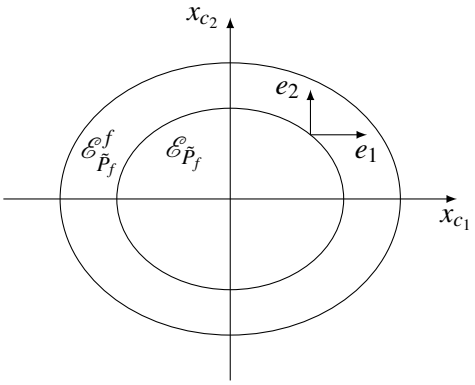

Figure 4: Relation between $\mathscr{E}_{\tilde{P}_f}$ and $\mathscr{E}^f_{\tilde{P}_f}$

At the end of the proof scheme, the system is stable with floating-point numbers if and only if the inclusion

$$\mathscr{E}^f_{\tilde{P}_f} \subset \mathscr{E}_P$$

holds. To succeed, $\mathscr{E}^f_{\tilde{P}_f}$ should be as narrow as possible with respect to Equation (3). This is not a clear criterion, as several shapes are possible for $\mathscr{E}^f_{\tilde{P}_f}$ with no clear winner. We propose to define $\mathscr{E}^f_{\tilde{P}_f}$ as the smallest homothety of $\mathscr{E}_{\tilde{P}_f}$ centered around 0 that satisfies (3). It can be computed rather easily, for any number of dimension; we give details for two dimensions.

Let a, b, c be the coefficients of $\mathscr{E}_{\tilde{P}_f}$:

$$\mathscr{E}_{\tilde{P}_f} = \{(x_{c_1}, x_{c_2}) \,|\, a x_{c_1}^2 + b x_{c_2}^2 + c x_{c_1} x_{c_2} \leq 1\}.$$

a, b and c are known positive values. Then there exists $k \geq 0$ s.t.

$$\mathscr{E}^f_{\tilde{P}_f} = \{(x_{c_1}, x_{c_2}) \,|\, ax_{c_1}^2 + bx_{c_2}^2 + cx_{c_1}x_{c_2} \leq k\}.$$

As $\mathscr{E}^f_{\tilde{P}_f}$ is wider than $\mathscr{E}_{\tilde{P}_f}$, $k \geq 1$. We need a condition on k that guarantees (3).

We consider a point (x_{c_1}, x_{c_2}) located on the border of $\mathscr{E}_{\tilde{P}_f}$:

$$ax_{c_1}^2 + bx_{c_2}^2 + cx_{c_1}x_{c_2} = 1. \tag{4}$$

By construction, for any values $\varepsilon_1, \varepsilon_2 \in \mathbb{R}$ s.t. $|\varepsilon_1| \leq e_1 \wedge |\varepsilon_2| \leq e_2$, the relation

$$(x_{c_1} + \varepsilon_1, x_{c_2} + \varepsilon_2) \in \mathscr{E}^f_{\tilde{P}_f}$$

must hold, that is to say:

$$a(x_{c_1} + \varepsilon_1)^2 + b(x_{c_2} + \varepsilon_2)^2 + c(x_{c_1} + \varepsilon_1)(x_{c_2} + \varepsilon_2) \leq k.$$

It develops into

$$(ax_{c_1}^2 + bx_{c_2}^2 + cx_{c_1}x_{c_2}) + (2a\varepsilon_1 + c\varepsilon_2)x_{c_1} + (2b\varepsilon_2 + c\varepsilon_1)x_{c_2} + (a\varepsilon_1^2 + b\varepsilon_2^2 + c\varepsilon_1\varepsilon_2) \leq k,$$

that is

$$1 + (2a\varepsilon_1 + c\varepsilon_2)x_{c_1} + (2b\varepsilon_2 + c\varepsilon_1)x_{c_2} + (a\varepsilon_1^2 + b\varepsilon_2^2 + c\varepsilon_1\varepsilon_2) \leq k.$$

due to (4).

Greatest values for the left-hand term are reached with $\varepsilon_1 = |e_1| \wedge \varepsilon_2 = |e_2|$, depending on the signs of x_{c_1} and x_{c_2}. As the ellipse $\mathscr{E}_{\tilde{P}_f}$ is symmetric about the origin point $(0,0)$, we can set $\varepsilon_1 = e_1$, which lets only two cases to study. Finally, we numerically verify that greatest values of the term are reached when $\varepsilon_2 = e_2$. This is the only case we detail here.

We can write:

$$1 + \alpha x_{c_1} + \beta x_{c_2} + \gamma \leq k$$

with values $\alpha = (2ae_1 + ce_2)$, $\beta = (2be_2 + ce_1)$ and $\gamma = (ae_1^2 + be_2^2 + ce_1e_2)$.

We known that x_{c_1} and x_{c_2} are bounded, thus so is the term $\alpha x_{c_1} + \beta x_{c_2}$: we can compute a minimum bound δ s.t. $\alpha x_{c_1} + \beta x_{c_2} \leq \delta$. So it is sufficient that k satisfies:

$$k \geq 1 + \gamma + \delta$$

Consequently, we define $\mathscr{E}^f_{\tilde{P}_f}$ as the smallest homothety of $\mathscr{E}_{\tilde{P}_f}$ that satisfy (3), obtained with $k = 1 + \gamma + \delta$. The instruction becomes:

```
% zc ∈ E_{Qμ},   u^2 ≤ U_f^2
xc = Ac*xc + Bc*yc;
% xc ∈ E^f_{P̃f},   u^2 ≤ U_f^2
```

In our case, starting from the ellipse

$$\mathscr{E}_{\tilde{P}_f} = \{(x_{c_1}, x_{c_2}) \,|\, 0.00269007 x_{c_1}^2 + 0.000341414 x_{c_1} x_{c_2} + 0.00247323 x_{c_2}^2 \leq 1\}$$

we get the following values:

$$\alpha = 1.03246 \cdot 10^{-17}, \quad \beta = 1.84829 \cdot 10^{-17}, \quad \gamma = 7.17582 \cdot 10^{-32}.$$

Using Mathematica, we found $\delta = 5.35754 \cdot 10^{-16} \gg \gamma$ and finally

$$k = 1 + 5.35754 \cdot 10^{-16}$$

that gives $\mathscr{E}^f_{\tilde{P}_f}$.

4.4 End of Proof Scheme

Then, what remains of the stability proof scheme becomes:

```
    % x_c ∈ E^f_P̃,   u^2 ≤ U_f^2
11  send(u, 1);
    % x_c ∈ E^f_P̃
12  receive(y, 2);
    % x_c ∈ E^f_P̃
13  receive(yd, 3);
    % x_c ∈ E^f_P̃
    % x_c ∈ E_P
14  end
```

As previously, the final assertion $\mathscr{E}^f_{\tilde{P}_f} \subset \mathscr{E}_P$ must be checked. Two cases are possible:

- either $\mathscr{E}^f_{\tilde{P}_f} \subset \mathscr{E}_P$, then we proved that the program is Lyapunov stable on a floating-point architecture;
- or $\mathscr{E}^f_{\tilde{P}_f} \not\subset \mathscr{E}_P$: as $\mathscr{E}^f_{\tilde{P}_f}$ was obtained through overapproximations, we cannot conclude about the program behavior.

In this open-loop case, we are able to check that $\mathscr{E}^f_{\tilde{P}_f} \subset \mathscr{E}_P$ (Mathematica script is available online: see footnote 1). Thus, the stability of the open-loop system with a 64-bit IEEE 754 compliant implementation is formally proven to hold using our proof translation scheme.

5 Closed-Loop Stability Proof

We now show how the proof of state boundedness of the closed-loop system specifications can be migrated to the level of the controller code and executable model of the system. To be more precise, we exploit the invariance of the ellipsoid $\mathscr{E}_P$ to develop a proof of proper behavior, that

is, stability and variable boundedness, for the computer program that implements the controller as it interacts with the physical system. Unlike the developments related to open-loop controller, this proof necessarily involves the presence of the physical system. In [Fer10], Feron chooses to represent the physical system and the computer program by two concurrent programs, as shown below.

Controller dynamics:

```
1c  Ac = [0.4990, -0.0500;
           0.0100, 1.0000];
2c  Bc = [1; 0];
3c  Cc = [564.48, 0];
4c  Dc = -1280;
5c  xc = zeros(2, 1);
6c  receive(y, 2); receive(yd,
           3);
7c  while (1)
8c     yc = max(min(y - yd, 1),
              -1);
9c     u = Cc*xc + Dc*yc;
10c    xc = Ac*xc + Bc*yc;
11c    send(u, 1);
12c    receive(y, 2);
13c    receive(yd, 3);
14c end
```

Physical system dynamics:

```
1p  Ap = [1.0000, 0.0100;
            -0.0100, 1.0000];
2p  Bp = [0.00005; 0.01];
3p  Cp = [1, 0];
4p  while (1)
5p     yp = Cp * xp;
6p     send(yp, 2);
7p     receive(up, 1);
8p     xp = Ap * xp + Bp * up;
9p  end
```

In this scheme, the computer program representation of the physical system is to remain unchanged, since it only exists for modeling purposes and does not correspond to any actual program, whereas the controller code is allowed to evolve to reflect the various stages of its implementation.

Establishing proofs of stability of the closed-loop system at the code level is necessarily tied to understanding the joint behavior of the controller and the plant. The entire state space therefore consists of the direct sum of state spaces of the controller and the physical system. The approach described in the previous sections is used to document the corresponding system of two processes. One interesting aspect of these processes is their concurrency, which can complicate the structure of the state transitions. However, a close inspection of the programs reveals that the transition structure of the processes does not need to rely on the extensions of Hoare's logic to concurrent programs: one program at a time is running, through the blocking nature of the `receive` primitive.

Feron's stability proof with real numbers is much longer than for the open-loop system. We do not detail it, the interested reader is referred to [Fer10] for full information. To be noticed, the resulting comments are not much more complex than those available from the study of the controller alone. On the good side, as already mentioned, the Hoare formalism is not significantly affected by the concurrent structure of the closed-loop system.

A floating-point representation of the closed-loop system consists of keeping the right column of the listing above in its original settings, while replacing the left column with the corresponding floating-point implementation, as we did in Section 4. Using similar techniques to the study of the controller alone, proof invariants can be tweaked to take into account constant changes and rounding errors resulting from the use of floating-point arithmetic. Unfortunately, using these invariants it cannot be shown that the stability condition holds at the end of the loop body. In this case, we are unable to prove the system stability on a floating-point architecture: either the system is not stable with the floating-point based controller, or the proof parameters ($\mathscr{E}_P$, μ, ...) must be chosen more carefully by the controller designer.

6 Conclusion

The general idea is to replace some of the invariants in the original proof scheme by wider ones that include rounding errors, with the hope that the stability condition is strong enough and still holds. This approach is made possible by the fact that rounding errors introduced by the operations used in the code are bounded on bounded inputs and bounded controller state variables.

In this document, we study the case of a floating-point representation of real numbers. They are not available on all architectures, especially on microcontrollers that are commonly used to implement control systems. We quickly discuss the alternative real-number representations.

- We can deal with fixed-point arithmetic the same way we do with floating-point, as long as we stand far enough from extremal values that can lead to overflows.

- Another way to represent real numbers is to use two integers, a numerator and a denominator. Considering that the input values are exact, the elementary operations do not introduce rounding errors but can easily lead to overflows, e.g. when computing

$$\frac{p_1}{q_1} + \frac{p_2}{q_2} = \frac{p_1 q_2 + p_2 q_1}{q_1 q_2}.$$

  A strategy must be used to prevent overflows by introducing approximations: in this case, the question is to quantify the errors introduced by these approximations.

In our example, we exclusively used additions and multiplications: divisions are not involved in linear control. Still, programs with divisions can also be analyzed, if the numerator can be shown to be far enough from zero: it is a supplementary constraint, but it is reasonable to assume that it should be respected on a realistic control system that uses divisions. Differentiable, periodic functions such as (sin) can be computed with an abacus and an interpolation function, thus with bounded error. In the same way, functions not periodic, but restricted to finite domains, can also be approximated. Other functions, such as (tan) or $\sqrt{}$, should raise more issues.

Bibliography

[BEFB94] S. Boyd, L. El Ghaoui, E. Feron, V. Balakrishnan. *Linear Matrix Inequalities in System and Control Theory*. Studies in Applied Mathematics 15. SIAM, Philadelphia, PA, June 1994.

[CCF+13] P. Cousot, R. Cousot, J. Feret, A. Miné, X. Rival et al. The Astrée Static Analyzer. 2001–2013.
http://www.astree.ens.fr/

[Coq13] The Coq Proof Assistant. 1984–2013.
http://coq.inria.fr/

[Fer10] E. Feron. From Control Systems to Control Software. *IEEE Control Systems Magazine* 30(6):50–71, Dec. 2010.
doi:10.1109/MCS.2010.938196
http://dx.doi.org/10.1109/MCS.2010.938196

[Gol91] D. Goldberg. What Every Computer Scientist Should Know About Floating Point Arithmetic. *ACM Computing Surveys* 23(1):5–48, 1991.
http://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html#689

[Hig02] N. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, 2002.
http://books.google.fr/books?id=epilvM5MMxwC

[IEE08] IEEE Task P754. *IEEE 754-2008, Standard for Floating-Point Arithmetic*. Aug. 2008.
doi:http://dx.doi.org/10.1109/IEEESTD.2008.4610935
http://ieeexplore.ieee.org/servlet/opac?punumber=4610933

[Mar11] M. Martel. Rangelab. 2011.
http://perso.univ-perp.fr/mmartel/rangelab.html

[Pau13] L. Paulson. Isabelle. 1990–2013.
http://isabelle.in.tum.de/

[Pel01] D. Peled. *Software Reliability Methods*. Texts in Computer Science Series. Springer, 2001.
http://books.google.fr/books?id=jJ-lTSlB71kC

[PVS13] PVS Specification and Verification System. 1992–2013.
http://pvs.csl.sri.com/

[Wol13] Wolfram Research. Mathematica. 1988–2013.
http://www.wolfram.com/mathematica/