Quantifying the gap between embedded control models and time-triggered implementations

Hakan Yazarel, Antoine Girard, George J. Pappas, Rajeev Alur
University of Pennsylvania
Philadelphia, PA 19104 USA
{hakan,agirard,pappasg}@seas.upenn.edu,   alur@cis.upenn.edu

Abstract

Mapping a set of feedback control components to executable code introduces errors due to a variety of factors such as discretization, computational delays, and scheduling policies. We argue that the gap between the model and the implementation can be rigorously quantified leading to predictability if the implementation is viewed as a sequence of control blocks executed in statically allocated time slots on a time-triggered platform. For linear systems controlled by linear controllers, we show how to calculate the exact error between the model-level semantics and the execution semantics of an implementation, allowing us to compare different implementations. The calculated error of different implementations is demonstrated using simulations on illustrative examples.

1 Introduction

Contemporary industrial control design relies heavily on tools for mathematical modelling and analysis. Even though such tools support automatic code generation from the model, many issues relevant to correctness and optimality of the implementation with respect to the timed semantics of the model are not satisfactorily addressed. Consequently, analysis results established for the model are not meaningful for the implementation, and the execution of the implementation is not predictable, causing problems in system integration. The challenge of bridging the gap between the model and the implementation motivates our research.

In this paper, we begin to address the challenge in the context of implementing feedback control loops by software [9]. Consider a physical plant interacting with a digital controller that measures some plant signals and generates appropriate control signals in order to influence the behavior of the plant. The models of both the plant and the controller have well-defined timed semantics (continuous-time or discrete-time) that can be used for simulation and analysis.

Once the controller design is complete, the designed controller model is typically expressed as a set of control (usually MATLAB) blocks, where each control block computes outputs that influence other blocks or the plant being controlled. Typically, the software implementation of such blocks relies on the support offered by a real-time operating system for scheduling periodic tasks. Each control block is compiled into an executable code in a host language such as C, and the control designer specifies a period for the corresponding task. To implement the resulting periodic tasks on a specific platform, one needs to determine the worst-case-execution-time for each block, and check whether the task set is schedulable using standard scheduling algorithms such as earliest-deadline-first (EDF) or rate monotonic scheduling (c.f. [7, 19]).

While the real-time scheduling based implementation offers a separation of concerns using the abstraction of real-time tasks with periods and deadlines, it introduces several sources of unpredictability. The only guarantee provided by the scheduler is that a control block will get a chance to complete its execution once during its period. In particular, there are no guarantees regarding when a control block actually reads its inputs and when its outputs become available to its environment, and the order in which the various blocks execute. While there is extensive research on reformulating the scheduling problem (c.f. [7]), for instance, by introducing ordering constraints among tasks, and introducing constraints on the latency between successive executions of a task, quantifying the error between the timed semantics of the control blocks and the possible executions of scheduled tasks, and understanding its impact on the application-level quality-of-service, remains difficult.

The recent emergence of time-triggered architecture as an implementation platform for embedded systems offers opportunities for a more predictable mapping of control models [20, 19]. In a time-triggered implementation, instead of mapping control blocks to periodic tasks, the compiler can allocate well-defined time slots to control blocks. For example, if each time slot is δ units, and a control block is mapped to the k-th slot, then we can assume that this control block reads its inputs at time $k\delta$, finishes its computation within time δ, and makes its outputs available to its environment at time $(k+1)\delta$. Given a mapping of all the control blocks to the time slots, one can precisely define the trajectories of the implementation and quantify the error with respect to the model-level semantics [1].

In our formalization, given a controller model as a set of interacting control blocks, we define the controller implementation on a time-triggered platform using a *dispatch sequence* that gives the order in which the blocks are repeatedly executed, and a *timing function* that gives the number of time slots needed to execute each block. For a given model of the plant, we can precisely define the semantics of the implementation, and measure its quality by metrics, such as the L_2-norm, of the discrepancy between the trajectories of the model and the implementation. Two questions naturally arise: can we compute such metrics, and thereby, compare two different implementations, and can we decide an "optimal" dispatch sequence from a given model.

In this paper, we focus on the former question for the case of plants and control designs modeled as linear control systems. Given a control design, a dispatch sequence, and a timing function, we model the controller implementation naturally as a *periodic linear time-varying system* (PLTV). Using the existing machinery for transforming a PLTV to a linear time-invariant linear system, we can then check if the implementation preserves stability as well as compute the error with respect to the model semantics measured by L_2-norms. It should be noted that even though only the controller model is implemented in real-time software, our analysis focuses on the error of the overall closed-loop system and therefore includes the plant dynamics in the analysis. Since different implementations correspond to different PLTVs, this gives us a way of comparing two implementations quantitatively. We explain our results using analysis and simulations on two illustrative examples.

Related Work.

Bridging the gap between high-level modeling or programming abstractions, and implementation platforms has been identified as a key challenge for embedded software research by many researchers (c.f. [24, 21]). Programming abstractions for embedded real-time controllers include synchronous reactive programming (languages such as ESTEREL and LUSTRE [4, 12, 11]), and the related *Fixed Logical Execution Time* (FLET) assumption used in the Giotto project [13, 14]. In particular, Giotto provides predictable execution semantics that is independent of the scheduling policy by enforcing that a periodic task reads its inputs at the beginning of its period, and delays its outputs till the end of the period. However, this line of research does not address the problem of errors introduced by mapping control models to programs. Research on time-triggered architecture has focused on achieving clock synchronization, fault tolerance, real-time communication, and automotive applications (c.f. [19, 20]).

Recently, the problem of generating code from timed and hybrid automata has been considered in [2, 16, 27], but in these papers the focus has been on choosing the sampling period so as to avoid errors due to switching and communication. The work on mapping Simulink blocks to Lustre focuses on signal dependencies [8]. In [1], *relative scheduling* as a way of generating a dispatch sequence for a control model for soft real-time applications has been explored but it did not have a framework for quantifying the errors. Model-based development of embedded systems is also promoted by other projects with orthogonal concerns: Ptolemy supports integration of heterogeneous models of computation [10], GME supports integration of multiple views of the system [18], and platform based design provides a framework for defining several layers of abstractions for system implementations [23]. Many variations of basic scheduling model have been explored, but the emphasis is not on quantifying the errors introduced during mapping control model to the task model. Perhaps the most related of these efforts is control-aware scheduling [26], in which periods for tasks are determined by optimizing a performance index.

There is a rich literature on sampled control systems [3] along two main approaches. The first approach starts with a continuous plant, and given implementation-dependent sampling times, the plant is discretized, and a controller is designed for the discretized plant. The disadvantage of this approach is that the control design must be repeated in order to accommodate changes in the implementation platform and does not therefore provide separation of concerns between the control engineer and the software implementation. The second approach starts with a designed continuous controller and focuses on discretizing the controller on some implementation platform [3]. Even though this is the spirit of our approach, the resulting error analysis has historically focused on quantifying the errors introduced due to sampling without paying attention to more detailed models of implementation platforms that must sequentially execute multiple control blocks. More sophisticated models of implementation platforms have recently considered platform constraints [22] or limited-bandwidth communication constraints [5, 15, 17], with a focus on designing controllers that take into account such constraints. Even though we use

[1]Admittedly, there are other sources of errors such as uncertainty in sensing, but the focus of this paper is on analyzing errors due to computation delays and scheduling.

techniques similar to those in [17], in this paper we decouple control from scheduling and focus on quantifying the implementation error induced by sequentializing the software execution of interdependent control blocks.

2 Modelling

In this section, we define the models of a real-time embedded control system and define the model-level semantics used for control design, the implementation-level semantics used for software execution on a time-triggered platform.

2.1 Feedback Control Model

Consider a finite set $X = \{x_1, x_2, \ldots, x_n\}$ of environment or *plant variables*, a set $Y = \{y_1, \ldots, y_p\}$ of *plant output variables*, and a set $U = \{u_1, \ldots, u_m\}$ of plant *control variables*. All variables take values in $\mathbb{R}$. A state over a set of variables is a mapping from the set of variables to corresponding values. The set of all possible plant states is thus $\mathbb{R}^n$, and we obtain similar sets of states for all other variables.

A feedback control model is a tuple $\mathcal{M} = \langle \mathcal{M}_P, \mathcal{M}_C \rangle$ consisting of a *plant model* $\mathcal{M}_P$ and a *controller model* $\mathcal{M}_C$. A plant model $\mathcal{M}_P$ consists of

- A function $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ that define the dynamics of variables x_i in terms of the current plant state and control inputs.
- A function $h : \mathbb{R}^n \to \mathbb{R}^p$ that expresses the observable output of the plant given the current plant state.

A controller model $\mathcal{M}_C$ consists of a finite set of control blocks $\mathcal{M}_C = (\mathcal{B}_1, \ldots, \mathcal{B}_m)$, one control block $\mathcal{B}_j$ for every control variable $u_j \in U$. Every control block $\mathcal{B}_j$ consists of a function $k_j : \mathbb{R}^{j-1} \times \mathbb{R}^p \to \mathbb{R}$ that expresses the feedback control law for the control variable u_j as a function of other control variables and observable plant outputs. We assume that the dependence among the control variables is acyclic and thus the control variables can be ordered so that the feedback law for the control variable u_j is a function of the plant outputs and the control variable $u_1, \ldots, u_{j-1}$. The feedback law of u_1 is only a function of the plant outputs.

2.2 Model Level Semantics

Given a feedback control model $\mathcal{M} = \langle \mathcal{M}_P, \mathcal{M}_C \rangle$ with variables X, Y, U, a trajectory for $\mathcal{M}$ is a function from the time domain $\mathbb{R}^{\geq 0}$ to the set of states over all variables. Let $x(t) = (x_1(t), \ldots, x_n(t))$ denote plant trajectories in vector notation, and, similarly, $y(t) = (y_1(t), \ldots, y_p(t))$, and $u(t) = (u_1(t), \ldots, u_m(t))$. Given feedback control model $\mathcal{M}$, we denote the *continuous-time semantics* of the feedback control model by $[\![\mathcal{M}]\!]_C$ and define $[\![\mathcal{M}]\!]_C$ as the collection of all trajectories $(x(t), y(t), u(t))$ that satisfy the following differential and algebraic constraints modelling both the plant and the controller dynamics:

$$M_P : \begin{cases} \dot{x}(t) &= f(x(t), u(t)) \\ y(t) &= h(x(t)) \\ x(0) &\in \mathbb{R}^n \end{cases} \quad (1)$$

$$M_C : \begin{cases} u_1(t) &= k_1(y(t)) \\ u_j(t) &= k_j(y(t), u_1(t) \ldots u_{j-1}(t)), \\ & \quad 2 \leq j \leq m \end{cases} \quad (2)$$

We assume that the feedback composition, illustrated in Figure 1, to be well-posed, meaning that for any initial plant state $x(0)$ the above equations have unique solutions. Given $x(0)$, we denote the unique solutions for the continuous-semantics as

$$(x(t), y(t), u(t)) = [\![\mathcal{M}]\!]_C(x(0)) \quad (3)$$

The continuous-time semantics is *implementation independent* semantics that is used for the mathematical analysis and design of controllers that achieve desired performance specifications of the output trajectories $y(t)$. Given a control design $\mathcal{M}_C$, the continuous-time semantics will serve as the ideal semantics. Our goal in this paper is to quantify the deviation from this ideal semantics when the controller $\mathcal{M}_C$ is implemented on a given time-triggered platform.

2.3 Implementation Level Semantics

The ideal continuous-time semantics assumes that all control blocks of controller $\mathcal{M}_C = (\mathcal{B}_1, \ldots, \mathcal{B}_m)$ are "computed" *instantaneously* and *simultaneously*. Of course, any software implementation of $\mathcal{M}_C$ will violate both assumptions. As discussed in the introduction, mapping control blocks to periodic tasks does not allow a mathematically rigorous execution semantics. Instead, we assume that the implementation is on a time-triggered platform in which time can be allotted in fixed-size slots.

To model the order in which the control blocks are executed we consider a *dispatch sequence* ρ, which is an infinite string over the set $\{\mathcal{B}_0, \mathcal{B}_1, \ldots, \mathcal{B}_m\}$. Here, $\mathcal{B}_0$ is used to model idling from the viewpoint of the controller (e.g., idling, or allocation of a time slot to activities other than the computation of control outputs). Typically, ρ will be periodic, and will be specified by a finite string that repeats. Each control block is to be executed without pre-emption, and when one control block completes its execution, the next block can start immediately. For example, given a controller $\mathcal{M}_C = (\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3)$ with three control blocks, possible dispatch sequences are the uniform sequence $(\mathcal{B}_1\mathcal{B}_2\mathcal{B}_3)^\omega$ or the nonuniform sequence $(\mathcal{B}_1\mathcal{B}_2\mathcal{B}_1\mathcal{B}_3\mathcal{B}_0)^\omega$ that also includes idling. Note that

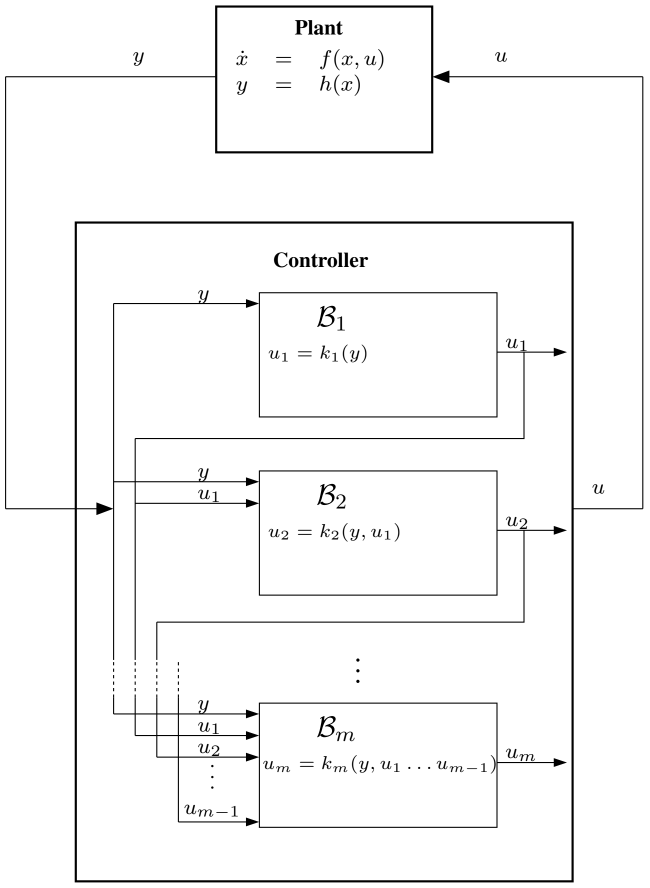

Figure 1. Continuous semantics of control loops

a dispatch sequence contains only ordering information, and thus, can potentially be independent of the processing speed of the platform.

A time-triggered platform provides an atomic time slot of *length* δ, and each block is assigned a fixed number of such slots. The computation of each control block $\mathcal{B}_i$ consists of reading the plant output variables using sensors, updating the control variable u_i, and finally writing the computed control value to the actuators at the end of its allotted time. We assume that sensor reading and actuator writing take zero time. The computation time of each control block is captured by a *timing function* $\tau : \{\mathcal{B}_1, \dots, \mathcal{B}_m\} \to \mathbb{Z}^+$ which associates to each control block the number of time slots needed to execute a control block. Assume $\tau(\mathcal{B}_0) = 1$.

Given a feedback control model $\mathcal{M} = \langle \mathcal{M}_P, \mathcal{M}_C \rangle$, a dispatch sequence ρ, a timing function τ and a time slot length δ, we can define the *implementation semantics* associated with $\mathcal{M}$, denoted as $[\![\mathcal{M}]\!]_{(\rho,\tau,\delta)}$, to be the set of trajectories obtained by executing the control blocks of controller $\mathcal{M}_C$ according to the dispatch sequence ρ, where the number of slots of length δ for each control block are chosen according to the timing function τ.

Dispatch sequence ρ, timing function τ and time slot length δ result in the following sequence of timing instants t_i: $t_0 = 0$ and $t_i = \sum_{k=0}^{i} \tau(\rho(k))\delta$ for $i \geq 1$. These are the precise timing instants when a control block completes its computation and its outputs are updated.

The implementation semantics $[\![\mathcal{M}]\!]_{(\rho,\tau,\delta)}$ can now be precisely defined as the collection of trajectories $(x(t), y(t), u(t))$ that for all $t \geq 0$ satisfy the plant dynamics that stays the same as in Equation (1), and the following controller implementation constraints for every $1 \leq j \leq m$ and $i \geq 0$,

$$u_j(t) = u_j(t_i) \quad \text{for } t_i < t < t_{i+1} \tag{4}$$

$$u_j(0) = 0 \tag{5}$$

$$u_1(t_{i+1}) = \begin{cases} u_1(t_i) & \text{if } \rho(i) \neq 1 \\ k_1(y(t_i)) & \text{if } \rho(i) = 1 \end{cases} \tag{6}$$

For $2 \leq j \leq m$,

$$u_j(t_{i+1}) = \begin{cases} u_j(t_i), & \text{if } \rho(i) \neq j \\ k_j(y(t_i), u_1(t_i) \dots u_{j-1}(t_i)), & \text{if } \rho(i) = j \end{cases} \tag{7}$$

These constraints say that the function $u(t)$ is piecewise-constant. If $\rho(i) = j$, the control output u_j is updated by evaluating k_j.

Given $x(0)$, we denote the unique solutions for the implementation-semantics as

$$(x(t), y(t), u(t)) = [\![\mathcal{M}]\!]_{(\rho,\tau,\delta)}(x(0)) \tag{8}$$

The main goal of this paper is to quantify the quality of the controller implementation for a particular dispatch sequence ρ, timing function τ and time slot length δ. Having defined both the ideal platform-independent semantics, and the platform-dependent semantics, we can directly define the error of the implementation as a function of the initial plant state $x(0)$ simply as

$$\begin{aligned} (x(t), y(t), u(t)) &= [\![\mathcal{M}]\!]_C(x(0)) \\ (\tilde{x}(t), \tilde{y}(t), \tilde{u}(t)) &= [\![\mathcal{M}]\!]_{(\rho,\tau,\delta)}(x(0)) \\ e_{\mathcal{M}}(\rho, \tau, \delta, x(0)) &= \int_0^{+\infty} \|y(t) - \tilde{y}(t)\|_2^2 dt \end{aligned} \tag{9}$$

We are therefore measuring the implementation error in the L_2 sense. The challenge is now to compute the L_2 norm of the implementation error as a function of $x(0)$, given implementation specifics (ρ, τ, δ).

Given a feedback control model $\mathcal{M}$, we will say that the implementation $(\rho_1, \tau_1, \delta_1)$ is more accurate than the implementation $(\rho_2, \tau_2, \delta_2)$ (noted $(\rho_1, \tau_1, \delta_1) \preceq_{\mathcal{M}} (\rho_2, \tau_2, \delta_2)$) if the implementation error of $(\rho_1, \tau_1, \delta_1)$ is smaller than the one of $(\rho_2, \tau_2, \delta_2)$ independently of the initial state of the plant:

$$\forall x(0) \in \mathbb{R}^n,\ e_{\mathcal{M}}(\rho_1, \tau_1, \delta_1, x(0)) \leq e_{\mathcal{M}}(\rho_2, \tau_2, \delta_2, x(0)). \quad (10)$$

Note that the relation $\preceq_{\mathcal{M}}$ is a preorder on the set of implementations.

3 Analysis

The goal of this part is to provide a method for the computation of the implementation error $e_{\mathcal{M}}(\rho, \tau, \delta, x(0))$. We assume that the plant model is a linear time invariant (LTI) systems (i.e. $f(x,u) = A_p x + B_p u$ and $h(x) = C_p x$):

$$\mathcal{M}_P : \left\{ \begin{array}{lcl} \dot{x}(t) & = & A_p x(t) + B_p u(t), \\ y(t) & = & C_p x(t), \\ x(0) & \in & \mathbb{R}^n \end{array} \right. \quad (11)$$

where $A_p \in \mathbb{R}^{n\times n}$, $B_p \in \mathbb{R}^{n\times m}$ and $C_p \in \mathbb{R}^{p\times n}$.

We assume that the feedback controller model $\mathcal{M}_C = (\mathcal{B}_1, \dots \mathcal{B}_m)$ consists of one control linear block $\mathcal{B}_i$ for every variable u_i (i.e. $k_1(y)$ is a linear combination of $y_1, \dots, y_p$ and $k_j(y, u_1 \dots u_{j-1})$ is a linear combination of $y_1, \dots, y_p$ and $u_1, \dots, u_{j-1}$). Then, the feedback control law given in a vector form is given by,

$$\mathcal{M}_C :\ u(t) = K_c y(t) + L_c u(t) \quad (12)$$

where $K_c \in \mathbb{R}^{m\times p}$, and $L_c \in \mathbb{R}^{m\times m}$. Note that the assumption that the dependence between control variables are acyclic implies that L_c is lower triangular.

For the sake of simplicity, we will assume that all the execution of all the control blocks require the same time. Thus, for all $1 \leq j \leq m$, $\tau(\mathcal{B}_i) = 1$ and for all $i \geq 0$, $t_i = i\delta$. Let $\mathcal{M} = \langle \mathcal{M}_P, \mathcal{M}_C \rangle$ and

$$\begin{array}{rcl} (x(t), u(t), z(t)) & = & [\![\mathcal{M}]\!]_C(x(0)) \quad (13) \\ (\tilde{x}(t), \tilde{u}(t), \tilde{z}(t)) & = & [\![\mathcal{M}]\!]_{(\rho,\tau,\delta)}(x(0)). \quad (14) \end{array}$$

3.1 Discretization of the implementation error

In this section, we discretize the integral defining the error due to implementation as given in Equation (9). We show that this integral can be computed from the sequences of values $x(t_i)$, $\tilde{x}(t_i)$, $\tilde{u}(t_i)$, for $i \geq 0$. Let us define the vector

$$\psi(t) = \left[\begin{array}{c} x(t) \\ \tilde{x}(t) \\ \tilde{u}(t) \end{array} \right] \quad (15)$$

Theorem 3.1 *There exists a positive symmetric matrix Q such that the implementation error defined by equation (9) is:*

$$e_{\mathcal{M}}(\rho, \tau, \delta, x(0)) = \sum_{i=0}^{i=+\infty} \psi(t_i)^T Q \psi(t_i) \quad (16)$$

Proof : First, let us remark that $e_{\mathcal{M}}(\rho, \tau, \delta, x(0)) = \sum_{i=0}^{i=+\infty} \mathcal{I}_i$ where $\mathcal{I}_i = \int_{t_i}^{t_{i+1}} \|y(t) - \tilde{y}(t)\|_2^2 dt$. On the interval $[t_i, t_{i+1})$, the evolution of the closed loop system in the continuous time semantics can be described by the following differential equation:

$$\dot{x}(t) = \left[A_p + B_p(I - L_c)^{-1} K_c C_p\right] x(t). \quad (17)$$

On the same interval, the evolution of the plant in the implementation semantics is given by

$$\dot{\tilde{x}}(t) = A_p \tilde{x}(t) + B_p \tilde{u}(t_i). \quad (18)$$

Let us define

$$A = \left[\begin{array}{cc} A_p + B_p(I-L_c)^{-1} K_c C_p & 0 \\ 0 & A_p \end{array} \right], B = \left[\begin{array}{c} 0 \\ B_p \end{array} \right]$$

$$C = [C_p\ \ -C_p],\ \varphi(t) = \left[\begin{array}{c} x(t) \\ \tilde{x}(t) \end{array} \right].$$

The output deviation due to implementation is $y_e(t) = y(t) - \tilde{y}(t)$. On $[t_i, t_{i+1})$, $y_e(t)$ is the output of the LTI system

$$\left\{ \begin{array}{lcl} \dot{\varphi}(t) & = & A\varphi(t) + B\tilde{u}(t_i), \\ y_e(t) & = & C\varphi(t) \end{array} \right. \quad (19)$$

By solving the differential equation, we have that

$$y_e(t) = C\left[e^{A(t-t_i)}(\varphi(t_i) + A^{-1}B\tilde{u}(t_i)) - A^{-1}B\tilde{u}(t_i)\right]$$

We can check that

$$\begin{array}{rcl} \mathcal{I}_i = & \int_{t_i}^{t_{i+1}} y_e(t)^T y_e(t) dt \\ = & (\varphi(t_i) + A^{-1}B\tilde{u}(t_i))^T M (\varphi(t_i) + A^{-1}B\tilde{u}(t_i)) \\ & -2(\varphi(t_i) + A^{-1}B\tilde{u}(t_i))^T N A^{-1} B\tilde{u}(t_i) \\ & +\tilde{u}(t_i)^T B^T (A^{-1})^T P A^{-1} B \tilde{u}(t_i) \end{array}$$

where

$$M = \int_0^\delta e^{A^T t} C^T C e^{At} dt,\ N = \int_0^\delta e^{A^T t} C^T C dt,$$
$$P = \int_0^\delta C^T C dt.$$

Let us define the following matrices

$$\begin{array}{rl} Q_{1,1} = & M, \\ Q_{1,2} = & Q_{2,1}^T = (M - N)A^{-1}B, \\ Q_{2,2} = & B^T (A^{-1})^T (M - N - N^T + P) A^{-1} B. \end{array}$$

Then, we have

$$\mathcal{I}_i = \left[\begin{array}{c} \varphi(t_i) \\ \tilde{u}(t_i) \end{array} \right]^T \left[\begin{array}{cc} Q_{1,1} & Q_{1,2} \\ Q_{2,1} & Q_{2,2} \end{array} \right] \left[\begin{array}{c} \varphi(t_i) \\ \tilde{u}(t_i) \end{array} \right].$$

This leads to the expected result. ■

In the following, we show that the sequences $x(t_i)$, $\tilde{x}(t_i)$ and $\tilde{u}(t_i)$ characterizing the implementation error can be computed from discrete time linear systems derived from the continuous time semantics and the implementation semantics of the feedback model.

3.2 Discrete time linear system from the continuous time semantics

From the differential equation (17), the sequences $x(t_i)$ are the successive iterations of the following discrete time LTI system:

$$x(t_{i+1}) = E\ x(t_i),\ E = e^{\delta[A_p+B_p(I-L_c)^{-1}K_cC_p]}. \quad (20)$$

3.3 Discrete time linear system from the implementation semantics

In this section, we show that the sequences $\tilde{x}(t_i)$ and $\tilde{u}(t_i)$ are the output sequences of a discrete time periodic linear time-varying (PLTV) system. The main idea is that during the execution of a particular control block, the feedback system evolves like a particular linear system. Since the sequence of executions of the control blocks is periodic, then the system evolves globally like a PLTV system.

3.3.1 Control blocks as linear systems

First, let us consider the control block $\mathcal{B}_0$ defined in section 2. Assume execution happens during the time interval $[t_i, t_{i+1}]$. The execution of $\mathcal{B}_0$ does not affect the control variables, hence $\tilde{u}(t_{i+1}) = \tilde{u}(t_i)$. On the time interval $[t_i, t_{i+1}]$, the plant evolves continuously according to equation (18). Hence, we have

$$\tilde{x}(t_{i+1}) = e^{\delta A_p}\tilde{x}(t_i) + (e^{\delta A_p} - I)A_p^{-1}B_p u(t_i). \quad (21)$$

Therefore, the value of the variables are modified by the execution of the control block $\mathcal{B}_0$ according to the following linear system:

$$\begin{bmatrix} \tilde{x}(t_{i+1}) \\ \tilde{u}(t_{i+1}) \end{bmatrix} = E_0 \begin{bmatrix} \tilde{x}(t_i) \\ \tilde{u}(t_i) \end{bmatrix},$$
$$E_0 = \begin{bmatrix} e^{\delta A_p} & (e^{\delta A_p} - I)A_p^{-1}B_p \\ 0 & I \end{bmatrix} \quad (22)$$

Let us now consider the control block $\mathcal{B}_j$, $1 \leq j \leq m$, executed during the time interval $[t_i, t_{i+1}]$. For $k \neq j$, the execution of $\mathcal{B}_j$ does not modify the value of the variables $\tilde{u}_k$. Thus, the value of the variable $\tilde{u}_j$ is updated according to

$$\begin{aligned} \tilde{u}_k(t_{i+1}) &= \tilde{u}_k(t_i), \text{ if } k \neq j \\ \tilde{u}_j(t_{i+1}) &= [K_c]_j C_p \tilde{x}(t_i) + [L_c]_j \tilde{u}(t_i) \end{aligned}$$

where $[K_c]_j$ and $[L_c]_j$ denote the jth lines of the matrices K_c and L_c. Let U_j be the matrix whose lines $[U_j]_k$ are

$$\begin{aligned} [U_j]_k &= 0, \quad \text{if } k \neq j \\ [U_j]_j &= [K_c]_j C_p \end{aligned}$$

Let V_j be the matrix whose lines $[V_j]_k$ are

$$\begin{aligned} [V_j]_k &= 0, \quad \text{if } k \neq j \\ [V_j]_j &= [L_c]_j \end{aligned}$$

Let W_j be the matrix whose coefficients are all zero except the jth element of its diagonal $[W_j]_{j,j} = -1$. Then,

$$\begin{aligned} \tilde{u}(t_{i+1}) &= U_j\tilde{x}(t_i) + V_j\tilde{u}(t_i) + (I + W_j)\tilde{u}(t_i) \\ \tilde{u}(t_{i+1}) &= U_j\tilde{x}(t_i) + (I + V_j + W_j)\tilde{u}(t_i). \end{aligned}$$

Thus, the value of the variables are modified by the execution of the control block $\mathcal{B}_j$ according to the following linear system:

$$\begin{bmatrix} \tilde{x}(t_{i+1}) \\ \tilde{u}(t_{i+1}) \end{bmatrix} = E_j \begin{bmatrix} \tilde{x}(t_i) \\ \tilde{u}(t_i) \end{bmatrix},$$
$$E_j = \begin{bmatrix} e^{\delta A_P} & (e^{\delta A_P} - I)A_P^{-1}B_P \\ U_j & I + V_j + W_j \end{bmatrix} \quad (23)$$

3.3.2 From dispatch sequences to PLTV systems

For the computation of the error due to implementation, we need the values of the variables $\tilde{x}(t)$ and $\tilde{u}(t)$ at the time values t_i for $i \geq 0$. For each control block, the value of the variables of the system at the end of the execution is a linear combination of these values at the beginning of the execution. Let us consider a dispatch sequence ρ defining the order in which the control blocks have to be executed. Hence, the sequences $\tilde{x}(t_i)$ and $\tilde{u}(t_i)$, for $i \geq 0$, can be determined from the following discrete time dynamical system:

$$\begin{bmatrix} \tilde{x}(t_{i+1}) \\ \tilde{u}(t_{i+1}) \end{bmatrix} = E_{\rho(i)} \begin{bmatrix} \tilde{x}(t_i) \\ \tilde{u}(t_i) \end{bmatrix} \quad (24)$$

Henceforth, we will assume that ρ is periodic, and let n_ρ denote its period. Then, this dynamical system is a PLTV system.

3.4 Computation of the Implementation Error

In this section, we propose a method to compute the error due to implementation. First, we define a system which describes both the evolutions defined by the continuous time semantics and the implementation semantics and whose state is the vector $\psi(t)$ defined by equation (15):

$$\psi(t) = \begin{bmatrix} x(t) \\ \tilde{x}(t) \\ \tilde{u}(t) \end{bmatrix} \quad (25)$$

Then,

$$\psi(t_{i+1}) = \bar{E}_{\rho(i)}\ \psi(t_i), \quad \bar{E}_{\rho(i)} = \begin{bmatrix} E & 0 \\ 0 & E_{\rho(i)} \end{bmatrix}. \quad (26)$$

This system is also a PLTV system of period n_ρ. A classical technique for the analysis of PLTV systems is the lifting technique (see for instance [25]). It consists in a transformation which allows rewriting the PLTV system as a LTI system. Let us define the following matrices

$$\begin{array}{rcl} \hat{E} & = & \bar{E}_{\rho(n_\rho-1)}\bar{E}_{\rho(n_\rho-2)}\dots\bar{E}_{\rho(1)}\bar{E}_{\rho(0)} \\ \hat{G} & = & \left[\begin{array}{c} I \\ \bar{E}_{\rho(0)} \\ \bar{E}_{\rho(1)}\bar{E}_{\rho(0)} \\ \vdots \\ \bar{E}_{\rho(n_\rho-2)}\dots\bar{E}_{\rho(0)} \end{array}\right] \end{array}$$

Then, we have, for all $l \geq 0$,

$$\left\{\begin{array}{rcl} \psi(t_{(l+1)n_\rho}) & = & \hat{E}\,\psi(t_{ln_\rho}) \\ \left[\begin{array}{c} \psi(t_{ln_\rho}) \\ \vdots \\ \psi(t_{ln_\rho+n_\rho-1}) \end{array}\right] & = & \hat{G}\,\psi(t_{ln_\rho}) \end{array}\right. \tag{27}$$

Let us remark that this new system is a LTI system. We define the block diagonal matrix $\hat{Q}$ composed of n_ρ blocks equal to the matrix Q given by Theorem 3.1:

$$\hat{Q} = \left[\begin{array}{ccc} Q & & \\ & \ddots & \\ & & Q \end{array}\right]$$

We can now state the main result of this paper.

Theorem 3.2 *The error due to the implementation can be computed by*

$$e_{\mathcal{M}}(\rho,\tau,\delta,x(0)) = x(0)^T H^T \mathcal{O} H x(0) \tag{28}$$

where

$$H = \left[\begin{array}{c} I \\ I \\ 0 \end{array}\right] \tag{29}$$

and $\mathcal{O}$ is the solution of the Lyapunov equation

$$\mathcal{O} = \hat{E}^T\mathcal{O}\hat{E} + \hat{G}^T\hat{Q}\hat{G}. \tag{30}$$

Proof : From theorem 3.1, we have

$$\begin{array}{rcl} e_{\mathcal{M}}(\rho,\tau,\delta,x(0)) & = & \displaystyle\sum_{l=0}^{l=+\infty}\sum_{i=lp}^{i=lp+p-1}\psi(t_i)^T Q\,\psi(t_i) \\ & = & \displaystyle\sum_{l=0}^{l=+\infty}\psi(t_{lp})^T\hat{G}^T\hat{Q}\hat{G}\,\psi(t_{lp}) \\ & = & \displaystyle\sum_{l=0}^{l=+\infty}\psi(0)^T(\hat{E}^l)^T\hat{G}^T\hat{Q}\hat{G}\hat{E}^l\,\psi(0) \\ & = & \psi(0)^T\mathcal{O}\,\psi(0) \end{array}$$

where

$$\mathcal{O} = \sum_{l=0}^{l=+\infty}(\hat{E}^l)^T\hat{G}^T\hat{Q}\hat{G}\hat{E}^l.$$

From the theory of LTI systems $\mathcal{O}$ is the solution of the Lyapunov equation (30). Moreover, from the continuous time and the implementation semantics, we have $\psi(0) = Hx(0)$. ■

The above Theorem provides a criterion for the comparison of two implementations of an embedded controller. Let $\mathcal{M}$ be a feedback control model, let (ρ_1,τ_1,δ_1) and (ρ_2,τ_2,δ_2) be two implementations. Then, from Theorem 3.2, there exist two symmetric matrices $\mathcal{O}_1$ and $\mathcal{O}_2$ such that for all $x(0) \in \mathbb{R}^n$,

$$\begin{array}{rcl} e_{\mathcal{M}}(\rho_1,\tau_1,\delta_1,x(0)) & = & x(0)^T H^T\mathcal{O}_1 Hx(0) \\ e_{\mathcal{M}}(\rho_2,\tau_2,\delta_2,x(0)) & = & x(0)^T H^T\mathcal{O}_2 Hx(0) \end{array}$$

Corollary 3.1 $(\rho_1,\tau_1,\delta_1) \preceq_{\mathcal{M}} (\rho_2,\tau_2,\delta_2)$ *if and only if the symmetric matrix $H^T(\mathcal{O}_2 - \mathcal{O}_1)H$ is positive definite.*

Proof : $(\rho_1,\tau_1,\delta_1) \preceq_{\mathcal{M}} (\rho_2,\tau_2,\delta_2)$ if and only if for all $x(0) \in \mathbb{R}^n$, $x(0)^T H^T\mathcal{O}_1 Hx(0) \leq x(0)^T H^T\mathcal{O}_2 Hx(0)$ or equivalently $0 \leq x(0)^T H^T(\mathcal{O}_2 - \mathcal{O}_1)Hx(0)$. The latest inequality holds for all $x(0)$ if and only if $H^T(\mathcal{O}_2-\mathcal{O}_1)H$ is positive definite. ■

Remark 3.3 *The rigorous analysis of the complexity of the whole procedure is quite difficult and tedious. However, let us remark that most of the computations are matrix operations (sums, products, linear matrix equations) whose complexity in time is polynomial in the size of the matrices. Furthermore, the largest matrix considered during the procedure is the matrix $\hat{G}$ which has $n_\rho \times (2n+m)$ lines and $2n+m$ columns. Therefore, it is clear that the complexity of the whole procedure is polynomial in the size of the feedback control model (i.e. $n+m$) and in the period of the dispatch sequence n_ρ.*

4 Computational Examples

In this section, we present numerical examples that illustrate the analysis method for comparing feedback controller implementations. The analysis method has been implemented as a MATLAB script. The inputs to the script are the continuous plant model $\mathcal{M}_P$ defined as in (11), the continuous controller model $\mathcal{M}_C$ defined as in (12), implementation specifics (ρ,τ,δ), for a periodic ρ, defined as in Section 2. Then the script computes the implementation error by the method presented in Section 3. We omit the details on the design process of continuous controllers in the examples, since the focus of the paper is not the design of the continuous controllers but the analysis of the implementations of the pre-designed feedback controllers.

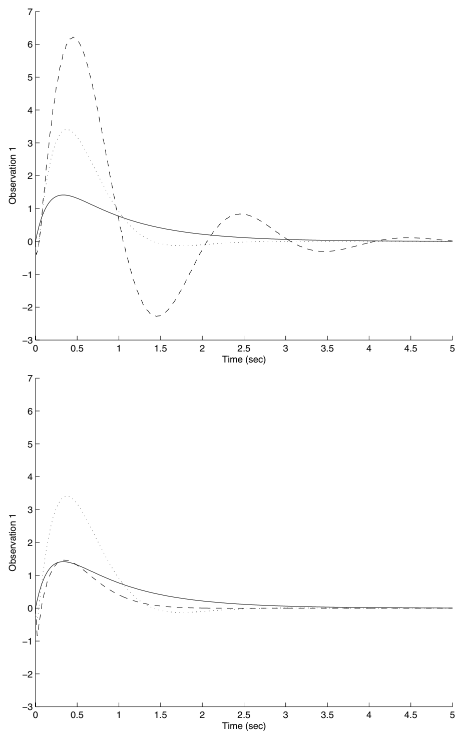

Figure 2. Plots of the first output of the plant for the semantics $[\![\mathcal{M}]\!]_C$ (solid line, top and bottom figures), $[\![\mathcal{M}]\!]_{(\rho_1,\tau_1,\delta_1)}$ (dotted line, top and bottom figures), $[\![\mathcal{M}]\!]_{(\rho_2,\tau_2,\delta_2)}$ (dashed line, top figure), $[\![\mathcal{M}]\!]_{(\rho_3,\tau_3,\delta_3)}$ (dashed line, bottom figure).

Example 4.1 Consider the 2-input 2-ouput continuous LTI plant model $\mathcal{M}_P$ given in the form of (11) with,

$$A_p = \begin{bmatrix} 0.65 & 0.065 \\ 0 & 13 \end{bmatrix},\ B_p = \begin{bmatrix} 10.4000 & 0 \\ -10.4000 & 10.4000 \end{bmatrix}, \quad C_p = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

A linear time-invariant controller model $\mathcal{M}_C = (\mathcal{B}_1, \mathcal{B}_2)$ designed by a control engineer to regulate the plant $\mathcal{M}_P$ is given in the matrix form as in (12) with $L_c = 0$ and

$$K_c = \begin{bmatrix} -1.4000 & 0.9000 \\ 0.5000 & -1.6000 \end{bmatrix}$$

Control block $\mathcal{B}_1$ updates controller input u_1 and keeps u_2 unchanged. Control block $\mathcal{B}_2$ updates control input u_2 and keeps u_1 unchanged. We consider three implementations of the controller $(\rho_1, \tau_1, \delta_1)$, $(\rho_2, \tau_2, \delta_2)$ and $(\rho_3, \tau_3, \delta_3)$ given by

$$\begin{aligned}
&\rho_1 = (\mathcal{B}_2\mathcal{B}_1)^\omega, && \tau_1(\mathcal{B}_i) = 1,\ \delta_1 = 0.01sec \\
&\rho_2 = (\mathcal{B}_2\mathcal{B}_1\mathcal{B}_1\mathcal{B}_1)^\omega, && \tau_2(\mathcal{B}_i) = 1,\ \delta_2 = 0.01sec \\
&\rho_3 = (\mathcal{B}_1\mathcal{B}_2\mathcal{B}_2\mathcal{B}_2\mathcal{B}_2\mathcal{B}_2)^\omega, && \tau_3(\mathcal{B}_i) = 1,\ \delta_3 = 0.01sec
\end{aligned}$$

Using the method presented in Section 3, we compute the matrices $\mathcal{O}_1, \mathcal{O}_2$ and $\mathcal{O}_3$ defined in (30). The computation time required to evaluate these matrices is 0.08 seconds. Using Corollary 3.1, we can order the implementations with the relation $\preceq_{\mathcal{M}}$: $(\rho_3, \tau_3, \delta_3)$ is more accurate than $(\rho_1, \tau_1, \delta_1)$ which is more accurate than $(\rho_2, \tau_2, \delta_2)$. For the initial value of the state of the plant $x(0) = [3\ -3]^T$, the implementation errors are

$$\begin{aligned}
e_{\mathcal{M}}(\rho_1, \tau_1, \delta_1, x(0)) &= 2.6013 \\
e_{\mathcal{M}}(\rho_2, \tau_2, \delta_2, x(0)) &= 7.0562 \\
e_{\mathcal{M}}(\rho_3, \tau_3, \delta_3, x(0)) &= 0.8458
\end{aligned}$$

In Figure 2, the evolutions of the first output of the plant for the three implementations are represented.

In this example, the dynamics of the second state of the plant is much faster than the dynamics of the first one. More attention is needed for the second state, since the propagation of errors due to implementation is much faster for this state. The implementation $(\rho_3, \tau_3, \delta_3)$ is more accurate since it gives more attention on the computation of the control block $\mathcal{B}_2$ which corresponds to the faster dynamics.

Example 4.2 The second example we consider in this paper is a model of vibration control of a multivariable smart structural system which is defined as a sensor-controller-actuator system which can automatically adjust to changes in the environment. We adopt the 2-input 4-output continuous model of such a system presented in [6], where the vibration of a two dimensional distributed cantilever plate is controlled using two PVDF sensors and two PZT actuators. The experimentally identified continuous LTI model $\mathcal{M}_P$ of the smart structural system is given in the form of (11) with,

$$A_p = \begin{bmatrix} -0.757 & -0.225 & -83.5 & -3.86 \\ -0.224 & -0.793 & -1.60 & -97.4 \\ 141 & -82.0 & -0.752 & -0.260 \\ -68.2 & 109 & -0.193 & -0.798 \end{bmatrix},$$

$$B_p = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0.914 & -0.209 \\ -0.189 & 0.456 \end{bmatrix},\ C_p = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

A continuous LTI controller model $\mathcal{M}_C = (\mathcal{B}_1, \mathcal{B}_2)$ designed by a control engineer to regulate the plant $\mathcal{M}_P$ is given in the matrix form as in (12) with $L_c = 0$ and

$$K = \begin{bmatrix} -6 & -0.1 & 20 & -5 \\ -0.35 & -0.3 & -10 & 0.1 \end{bmatrix}$$

We note that, the plant is naturally a stable plant in the sense that, it is stable even though no controller is used. However, the settling time is very slow and the controller is used to stabilize the plant faster.

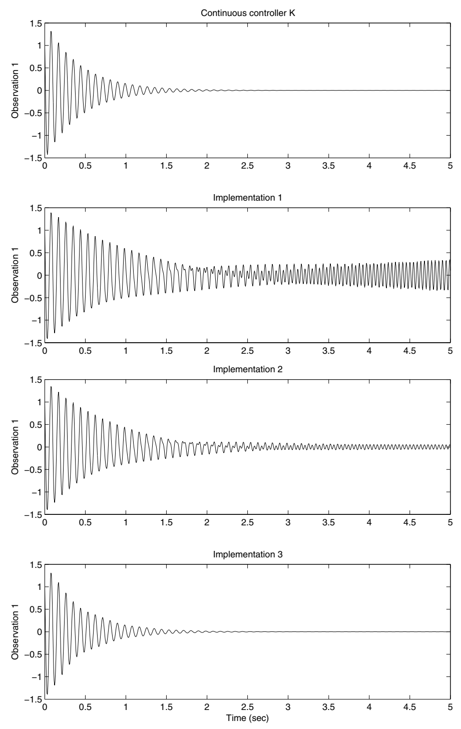

Figure 3. Plots of the first output of the plant model for the semantics $[\![\mathcal{M}]\!]_C$ (top figure), $[\![\mathcal{M}]\!]_{(\rho_1,\tau_1,\delta_1)}$ (second to top figure), $[\![\mathcal{M}]\!]_{(\rho_2,\tau_2,\delta_2)}$ (second to bottom figure), $[\![\mathcal{M}]\!]_{(\rho_3,\tau_3,\delta_3)}$ (bottom figure).

We first consider three implementations $(\rho_1, \tau_1, \delta_1)$, $(\rho_2, \tau_2, \delta_2)$ and $(\rho_3, \tau_3, \delta_3)$ given by

$$\begin{array}{ll} \rho_1 = (\mathcal{B}_1\mathcal{B}_2)^\omega, & \tau_1(\mathcal{B}_i) = 1, \\ \rho_2 = (\mathcal{B}_1\mathcal{B}_1\mathcal{B}_2)^\omega, & \tau_2(\mathcal{B}_i) = 1, \\ \rho_3 = (\mathcal{B}_1\mathcal{B}_1\mathcal{B}_1\mathcal{B}_1\mathcal{B}_1\mathcal{B}_1\mathcal{B}_1\mathcal{B}_1\mathcal{B}_1\mathcal{B}_2)^\omega, & \tau_3(\mathcal{B}_i) = 1, \end{array}$$

and $\delta_1 = \delta_2 = \delta_3 = 0.005sec$.

Using a simple test on the eigenstructure of some matrices defined by the method presented in section 3, we can determine that the first implementation is unstable. Hence, for all initial values of the plant $x(0) \in \mathbb{R}^4$, $e_{\mathcal{M}}(\rho_1, \tau, \delta, x(0)) = \infty$. It is worth noting that the implementation $(\rho_1, \tau_1, \delta_1)$ whose dispatch sequence alternates control blocks $\mathcal{B}_1$ and $\mathcal{B}_2$ uniformly (which seems a reasonable choice) destabilizes the plant $\mathcal{M}_p$ which was initially stable. This illustrates the critical fact that a bad implementation of the feedback controller can have dramatic consequences on the behavior of the closed loop system.

We also computed the matrices $\mathcal{O}_2$ and $\mathcal{O}_3$ defined in (30). The computation time required to evaluate these matrices is 0.12 seconds. Using Corollary 3.1, we can order the implementations with the relation $\preceq_{\mathcal{M}}$: $(\rho_3, \tau_3, \delta_3)$ is more accurate than $(\rho_2, \tau_2, \delta_2)$. Then, for the initial value of the state of the plant $x(0) = [1\ 1\ 1\ 1]^T$, the implementation errors are

$$\begin{aligned} e_{\mathcal{M}}(\rho_1, \tau_1, \delta_1, x(0)) &= \infty \\ e_{\mathcal{M}}(\rho_2, \tau_2, \delta_2, x(0)) &= 0.5401 \\ e_{\mathcal{M}}(\rho_3, \tau_3, \delta_3, x(0)) &= 0.1061 \end{aligned}$$

In Figure 3, the evolutions of the first output of the plant model for the three implementations are represented. Now let us consider the implementations consisting of the same dispatch sequences and the same timing functions but running on a faster platform:

$$\begin{array}{ll} \rho_1 = (\mathcal{B}_1\mathcal{B}_2)^\omega, & \tau_1(\mathcal{B}_i) = 1, \\ \rho_2 = (\mathcal{B}_1\mathcal{B}_1\mathcal{B}_2)^\omega, & \tau_2(\mathcal{B}_i) = 1, \\ \rho_3 = (\mathcal{B}_1\mathcal{B}_1\mathcal{B}_1\mathcal{B}_1\mathcal{B}_1\mathcal{B}_1\mathcal{B}_1\mathcal{B}_1\mathcal{B}_1\mathcal{B}_2)^\omega, & \tau_3(\mathcal{B}_i) = 1, \end{array}$$

and $\delta_1' = \delta_2' = \delta_3' = 0.002sec$.

Let us remark that the implementation $(\rho_1, \tau_1, \delta_1)$ is now stable. More generally, we can check experimentally that for the same dispatch sequence and the same timing function, the faster the platform the more accurate the implementation. Actually, the six implementations can be ordered according to the preorder $\preceq_{\mathcal{M}}$:

$$\begin{aligned} &(\rho_3, \tau_3, \delta_3') \preceq_{\mathcal{M}} (\rho_3, \tau_3, \delta_3) \preceq_{\mathcal{M}} (\rho_2, \tau_2, \delta_2') \\ &\preceq_{\mathcal{M}} (\rho_1, \tau_1, \delta_1') \preceq_{\mathcal{M}} (\rho_2, \tau_2, \delta_2) \preceq_{\mathcal{M}} (\rho_1, \tau_1, \delta_1) \end{aligned}$$

For the initial value of the state of the plant $x(0) = [1\ 1\ 1\ 1]^T$, the implementation errors are

$$\begin{aligned} e_{\mathcal{M}}(\rho_1, \tau_1, \delta_1', x(0)) &= 0.1744 \\ e_{\mathcal{M}}(\rho_2, \tau_2, \delta_2', x(0)) &= 0.1368 \\ e_{\mathcal{M}}(\rho_3, \tau_3, \delta_3', x(0)) &= 0.0695 \end{aligned}$$

An important point to note is that, the performances of the implementations $(\rho_1, \tau_1, \delta_1')$ and $(\rho_2, \tau_2, \delta_2')$ are not as good as the one of $(\rho_3, \tau_3, \delta_3)$ even though the latter runs on a platform more than twice slower. This illustrates the fact that a rigorous analysis of the error due to the implementation of a feedback controller is of great interests since the performance of a controller can be considerably increased without changing the platform.

5 Discussions and Conclusions

In this paper, we formulated the problem of quantifying the error due to the implementation of embedded controllers on time-triggered platforms. For linear models and controllers, we presented a method to exactly compute the L_2-error of the deviation of the output of the plant in the implementation semantics from the output of the plant in the continuous time semantics. This method relies on standard tools of periodic linear time-varying systems, and gives a criterion to compare implementations independently of the initial value of the state of the plant.

Future research includes the extension of our framework to larger classes of plant models, including uncertain linear, nonlinear and hybrid systems, as well as more general nonlinear and dynamic controllers. Whereas exact computation of the error may not be feasible in these general settings, computable error bounds for appropriate norms will still enable the comparison of different implementations. Finally, our approach may enable us to develop a scheduling framework on time-triggered platforms in order to reduce implementation error, while potentially achieving a decomposition between dispatch sequences and timing functions.

6 Acknowledgements

This research is partially supported by the National Science Foundation Information Technology Research grant CCR 0410662. The second author is partially supported by the Région Rhône-Alpes (Projet CalCel). We also thank Nader Motee for valuable discussions on control theory.

References

[1] R. Alur and A. Chandrashekharapuram. Dispatch sequences for embedded control models. In *Proceedings of the 11th IEEE Real-time and Embedded Technology and Applications*, pages 508–518, 2005.

[2] R. Alur, F. Ivancic, J. Kim, I. Lee, and O. Sokolsky. Generating embedded software from hierarchical hybrid models. In *Proceedings of the ACM Conference on Languages, Compilers, and Tools for Embedded Systems*, pages 171–182, 2003.

[3] K. Aström and B. Wittenmark. *Computer-controlled systems: Theory and Design*. Prentice Hall, 1997.

[4] G. Berry and G. Gonthier. The synchronous programming language ESTEREL: design, semantics, implementation. Technical Report 842, INRIA, 1988.

[5] R. Brockett. Stabilization of motor networks. In *Proceedings of the 1995 IEEE Conference on Decision and Control*, pages 1484–1488, Dec. 1995.

[6] R. Butler and V. Rao. A state space modeling and control method for multivariable smart structural systems. *Smart Materials and Structures*, 5(4):386–399, 1996.

[7] G. Buttazo. *Hard real-time computing systems: Predictable scheduling algorithms and applications*. Kluwer Academic Publishers, 1997.

[8] P. Caspi, A. Curic, A. Maignan, C. Sofronis, and S. Tripakis. Translating discrete-time Simulink to Lustre. In *Proceedings of Third International Conference on Embedded Software*, LNCS 2855, pages 84–99, 2003.

[9] P. Caspi and O. Maler. From control loops to real-time programs. *Handbook of Networked and Embedded Control Systems (to appear)*, 2005.

[10] J. Eker, J. Janneck, E. Lee, J. Liu, X. Liu, J. Luvig, S. Neuendorffer, S. Sachs, and Y. Xiong. Taming heterogeneity–the Ptolemy approach. *Proceedings of the IEEE*, 91(1):127–144, 2003.

[11] N. Halbwachs. *Synchronous Programming of Reactive Systems*. Kluwer Academic Publishers, 1993.

[12] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous dataflow programming language Lustre. *Proceedings of the IEEE*, 79:1305–1320, 1991.

[13] T. Henzinger, B. Horowitz, and C. Kirsch. Giotto: A time-triggered language for embedded programming. *Proceedings of the IEEE*, 91(1):84–99, 2003.

[14] T. Henzinger and C. Kirsch. The embedded machine: Predictable, portable, real-time code. In *Proceedings of the ACM Conference on Programming Language Design and Implementation*, pages 315–326, 2002.

[15] D. Hristu and K. Morgansen. Limited communication control. *Systems and control letters*, 37(4):193–205, July 1999.

[16] Y. Hur, J. Kim, I. Lee, and J. Choi. Sound code generation from communicating hybrid models. In *Hybrid Systems: Computation and Control, Proceedings of the 7th International Workshop*, LNCS 2993, pages 432–447, 2004.

[17] H. Ishii and B. Francis. Stabilization with control networks. *Automatica*, 38:1745–1751, 2002.

[18] G. Karsai, J. Sztipanovits, A. Ledeczi, and T. Bapty. Model-integrated development of embedded software. *Proceedings of the IEEE*, 91(1):145–164, 2003.

[19] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, 2000.

[20] H. Kopetz and G. Bauer. The time triggered architecture. *Proceedings of the IEEE*, 91(1):112–126, 2003.

[21] E. Lee. What's ahead for embedded software. *IEEE Computer*, pages 18–26, September 2000.

[22] L. Palopoli, C. Pinello, A. L. Sangiovanni-Vincentelli, L. El-Ghaoui, and A. Bicchi. Synthesis of robust control systems under resource constraints. In M. Greenstreet and C. Tomlin, editors, *Hybrid Systems: Computation and Control*, volume LNCS 2289 of *Lecture Notes in Computer Science*, pages 337–350. Springer-Verlag, Heidelberg, Germany, 2002.

[23] A. Sangiovanni-Vincetelli, L. Carloni, F. D. Bernardinis, and M. Sgori. Benefits and challenges for platform-based design. In *Proceedings of the 41th ACM Design Automation Conference*, pages 409–414, 2004.

[24] S. Sastry, J. Sztipanovits, R. Bajcsy, and H. Gill. Modeling and design of embedded software. *Proceedings of the IEEE*, 91(1), 2003.

[25] S.Bittanti and P.Colaneri. Invariant representations of discrete-time periodic systems - a survey. *Automatica*, 36(12):1777–1793, 2000.

[26] D. Seto, J. Lehoczky, L. Sha, and K. Shin. On task schedulability in real-time control systems. In *Procedings of the IEEE Real-Time Systems Symposium*, 1996.

[27] M. D. Wulf, L. Doyen, and J. Raskin. Almost ASAP semantics: From timed models to timed implementations. In *Hybrid Systems: Computation and Control, Proceedings of the 7th International Workshop*, LNCS 2993, pages 296–310, 2004.