



University of Pennsylvania
ScholarlyCommons

Departmental Papers (ESE)

Department of Electrical & Systems Engineering

8-2012

Time-triggered Implementations of Dynamic Controllers

Truong X. Nghiem
University of Pennsylvania

George J. Pappas
University of Pennsylvania, pappasg@seas.upenn.edu

Rajeev Alur
University of Pennsylvania, alur@cis.upenn.edu

Antoine Girard
Université Joseph Fourier

Follow this and additional works at: http://repository.upenn.edu/ese_papers

 Part of the [Controls and Control Theory Commons](#)

Recommended Citation

Truong X. Nghiem, George J. Pappas, Rajeev Alur, and Antoine Girard, "Time-triggered Implementations of Dynamic Controllers", . August 2012.

Nghiem, T. X.; Pappas, G. J.; Alur, R. & Girard, A. (2012). Time-Triggered Implementations of Dynamic Controllers. *ACM Transactions on Embedded Computing Systems (TECS)*, 11(S2), Article 58. doi: [10.1145/2331147.2331168](https://doi.org/10.1145/2331147.2331168)

© ACM, 2012. This is the author's version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. The definitive version was published in *ACM Transactions on Embedded Computing Systems*, Vol. 11, No. S2, August 2012 <http://doi.acm.org/10.1145/2331147.2331168>

This paper is posted at ScholarlyCommons. http://repository.upenn.edu/ese_papers/620
For more information, please contact repository@pobox.upenn.edu.

Time-triggered Implementations of Dynamic Controllers

Abstract

Bridging the gap between model-based design and platform-based implementation is one of the critical challenges for embedded software systems. In the context of embedded control systems that interact with an environment, a variety of errors due to quantization, delays, and scheduling policies may generate executable code that does not faithfully implement the model-based design. In this paper, we show that the performance gap between the model-level semantics of linear dynamic controllers, e.g. the proportional- integral-derivative (PID) controllers, and their implementation-level semantics can be rigorously quantified if the controller implementation is executed on a predictable time-triggered architecture. Our technical approach uses lifting techniques for periodic, time-varying linear systems in order to compute the exact error between the model semantics and the execution semantics. Explicitly computing the impact of the implementation on overall system performance allows us to compare and partially order different implementations with various scheduling or timing characteristics.

Keywords

control, dynamic controller, time-triggered, implementation, performance, PID, PI, scheduling

Disciplines

Controls and Control Theory | Electrical and Computer Engineering | Engineering

Comments

Nghiem, T. X.; Pappas, G. J.; Alur, R. & Girard, A. (2012). Time-Triggered Implementations of Dynamic Controllers. *ACM Transactions on Embedded Computing Systems (TECS)*, 11(S2), Article 58. doi: [10.1145/2331147.2331168](https://doi.org/10.1145/2331147.2331168)

© ACM, 2012. This is the author's version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. The definitive version was published in *ACM Transactions on Embedded Computing Systems*, Vol. 11, No. S2, August 2012 <http://doi.acm.org/10.1145/2331147.2331168>

Time-triggered Implementations of Dynamic Controllers

TRUONG NGHIEM, GEORGE J. PAPPAS, and RAJEEV ALUR, University of Pennsylvania and ANTOINE GIRARD, Laboratoire Jean Kuntzmann, Université Joseph Fourier

Bridging the gap between model-based design and platform-based implementation is one of the critical challenges for embedded software systems. In the context of embedded control systems that interact with an environment, a variety of errors due to quantization, delays, and scheduling policies may generate executable code that does not faithfully implement the model-based design. In this paper, we show that the performance gap between the model-level semantics of linear dynamic controllers, e.g. the proportional-integral-derivative (PID) controllers, and their implementation-level semantics can be rigorously quantified if the controller implementation is executed on a predictable time-triggered architecture. Our technical approach uses lifting techniques for periodic, time-varying linear systems in order to compute the exact error between the model semantics and the execution semantics. Explicitly computing the impact of the implementation on overall system performance allows us to compare and partially order different implementations with various scheduling or timing characteristics.

Categories and Subject Descriptors: C.3 [Computer Systems Organization]: Special-purpose and Application-based Systems—*Real-time and embedded systems*; C.4 [Computer Systems Organization]: Performance of Systems—*Modeling techniques, Performance attributes*; J.7 [Computer Applications]: Computers in other Systems—*Real time, Command and control*

General Terms: Design, Performance

Additional Key Words and Phrases: control, dynamic controller, time-triggered, implementation, performance, PID, PI, scheduling

ACM Reference Format:

Truong Nghiem, George J. Pappas, Rajeev Alur, and Antoine Girard, 2012. Time-triggered Implementations of Dynamic Controllers. *ACM Trans. Embedd. Comput. Syst.* 11, S2, Article 58 (August 2012), 23 pages. DOI: <http://dx.doi.org/10.1145/0000000.0000000>

1. INTRODUCTION

Bridging the gap between high-level modeling or programming abstractions, and implementation platforms is one of the key challenge for embedded software research [Sastry et al. 2003; Lee 2000]. The goal of our research, initiated in a recent paper [Yazarel et al. 2005], is to address this challenge in the context of implementing feedback control loops by software [Caspi and Maler 2005].

Consider a physical plant interacting with a controller that measures some plant signals and generates appropriate control signals in order to influence the behavior

This work has been supported by NSF Information Technology Research (ITR) Grant 0121431 and NSF Embedded and Hybrid Systems (EHS) Grant 0410662.

Authors' addresses: Truong Nghiem and George J. Pappas, Department of Electrical and Systems Engineering, University of Pennsylvania, USA; Rajeev Alur, Department of Computer and Information Science, University of Pennsylvania, USA; Antoine Girard, Laboratoire Jean Kuntzmann, Université Joseph Fourier, France.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.

© 2012 ACM 1539-9087/2012/08-ART58 \$15.00

DOI: <http://dx.doi.org/10.1145/0000000.0000000>

of the plant. The models of both the plant and the controller have well-defined timed semantics 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. 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 (c.f. [Buttazo 1997; Kopetz 2000]).

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. 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. As a result, 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 [Kopetz and Bauer 2003; Kopetz 2000]. In a time-triggered implementation, instead of mapping control blocks to periodic tasks, the compiler can allocate well-defined time slots to control blocks. 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.

In our formalization, given a dynamic 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. Given linear control plants, linear dynamic controllers (e.g. PID controllers), a dispatch sequence, and a timing function, we model the controller implementation naturally as a *periodic linear time-varying system* (PLTV). Compared to our previous work [Nghiem et al. 2006], in this paper we consider the application of our theoretical results in the analysis and design of controller implementations in section 4. In section 5, we also present a MATLAB implementation of our techniques and a SIMULINK-based simulation tool for time-triggered real-time control systems. More numerical examples with detailed discussion are provided in section 6 to illustrate our results.

Related Work : Programming abstractions for embedded real-time controllers include synchronous reactive programming (languages such as ESTEREL and LUSTRE [Halbwachs et al. 1991; Halbwachs 1993]), and the related *Fixed Logical Execution Time* (FLET) assumption used in the Giotto project [Henzinger et al. 2003]. Research on time-triggered architecture has focused on achieving clock synchronization, fault tolerance, real-time communication, and automotive applications (c.f. [Kopetz 2000; Kopetz and Bauer 2003]). The goal of this research has been ensuring predictable communication between components. In the context of this paper, time-triggered platform offers predefined time slots for scheduling, and we study how this can be exploited for predictable execution of control blocks.

Recently, the problem of generating code from timed and hybrid automata has been considered in [Alur et al. 2003; Hur et al. 2004; Wulf et al. 2004], but 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

[Caspi et al. 2003]. In [Alur and Chandrashekarapuram 2005], *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. Automaton-based scheduling of control systems was introduced in [Alur and Weiss 2008], however quantitative measure of the performance was not considered. 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 [Seto et al. 1996] and control-scheduling co-design [Årzén et al. 2005].

There is a rich literature on sampled control systems [Aström and Wittenmark 1997] along two main approaches. The first approach discretizes a continuous plant given implementation dependent sampling times, and a controller is designed for the discretized plant. The second approach starts with a designed continuous controller and focuses on discretizing the controller on some implementation platform [Aström and Wittenmark 1997]. 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.

2. MODELING

In this section, we model embedded control systems and provide two different semantics: model-level semantics used for control design, and implementation-level semantics used for execution on a time-triggered platform. The goal of this paper will be to compute a distance between these two semantics, thus providing a measure for the quality of the implementation.

2.1. Feedback Control Model

Consider a finite set $X = \{x_1, \dots, x_n\}$ of *plant variables*, a set $Y = \{y_1, \dots, y_p\}$ of *output variables*, a set $U = \{u_1, \dots, u_m\}$ of *control variables*, and a set $Z = \{z_1, \dots, z_q\}$ of *controller's internal 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 $f : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^n$ that defines the dynamics of variables x_i in terms of the current plant state and control inputs.
- A function $h : \mathbb{R}^n \rightarrow \mathbb{R}^p$ that expresses the observable output of the plant given the current plant state.

A controller model $\mathcal{M}_C = (\mathcal{B}_1, \dots, \mathcal{B}_r)$ consists of r control blocks, each describes the dynamics of some internal variables or computes the values of some control variables in terms of observable plant outputs, internal variables, and other control variables. We assume that the control blocks are well-defined so that given the values of the internal variables at time t_0 and the plant outputs for time $t \geq t_0$, the values of all internal variables and control variables at time $t \geq t_0$ are uniquely determined.

Note that the class of controllers considered by this model is dynamic due to the existence of the internal variables z_i that have their own dynamics. This model captures the widely used proportional-integral-derivative (PID) controllers as well as more general observer-based controllers. This is a significant extension of the class of static controllers considered in our previous work [Yazarel et al. 2005].

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, Z , 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 $x(t) = (x_1(t), \dots, x_n(t))$ denote plant trajectories in vector notation, and, similarly, $y(t) = (y_1(t), \dots, y_p(t))$, $u(t) = (u_1(t), \dots, u_m(t))$, and $z(t) = (z_1(t), \dots, z_q(t))$. Given feedback control model \mathcal{M} , we denote the *continuous-time semantics* of the feedback control model by $\llbracket \mathcal{M} \rrbracket$ and define $\llbracket \mathcal{M} \rrbracket$ as the collection of all trajectories $(x(t), y(t), u(t), z(t))$ that for all $t \geq 0$ satisfy the following differential and algebraic constraints modeling the feedback interconnection between the plant and the controller dynamics:

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

$$M_C : \begin{cases} \dot{z}(t) = g(z(t), y(t)) \\ u_1(t) = k_1(y(t), \dot{y}(t), z(t)) \\ u_j(t) = k_j(y(t), \dot{y}(t), z(t), u_1(t), \dots, u_{j-1}(t)), \quad 2 \leq j \leq m \\ z(0) = 0 \end{cases} \quad (2)$$

where $g : \mathbb{R}^q \times \mathbb{R}^p \rightarrow \mathbb{R}^q$ describes the dynamics of the internal variables z_i and k_j computes the control variable u_j . The derivative $\dot{y}(t)$ is included in k_j to allow modeling of PID controllers. The dependence among the control variables is acyclic so that the feedback law for the control variable u_j , $j > 1$, is a function of the plant outputs and their derivatives, the internal states, and the control variables u_1, \dots, u_{j-1} , while u_1 does not depend on the other control variables. We assume that the feedback composition is well-posed, meaning that for any initial plant state $x(0)$ the above equations have unique solutions, i.e. the differential equations satisfy the Lipschitz-continuous condition. Given $x(0)$, we denote the unique solution for the continuous-semantics as

$$(x(t), y(t), u(t), z(t)) = \llbracket \mathcal{M} \rrbracket(x(0)) \quad (3)$$

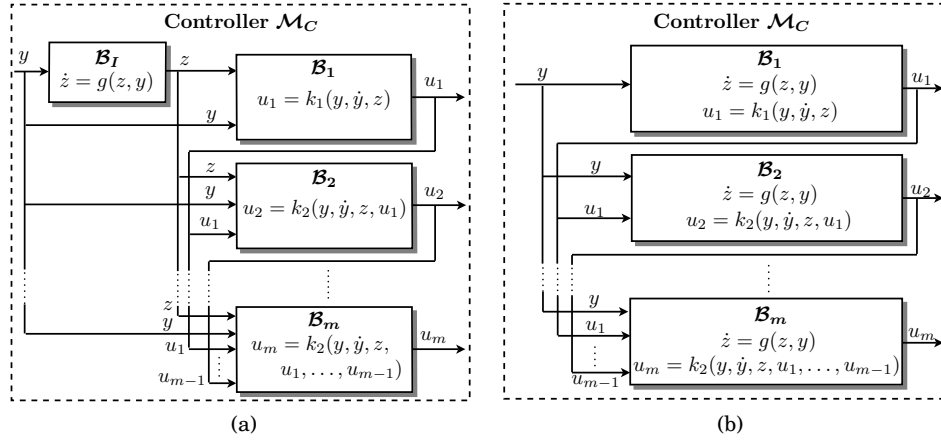
The continuous-time semantics is *implementation independent* semantics that is used for the mathematical analysis and design of controllers that achieve desired performance specifications of the output trajectories $y(t)$, using a variety of techniques from control theory. Our goal in this paper is 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 control blocks of controller $\mathcal{M}_C = (\mathcal{B}_1, \dots, \mathcal{B}_r)$ are usually defined during the process of designing the controller, using a design tool such as SIMULINK/MATLAB. Typically, the designer may choose among several structures of $\mathcal{M}_C = (\mathcal{B}_1, \dots, \mathcal{B}_r)$ to express the same model-level semantics (2). Figure 1 illustrates two possible structures. In Figure 1a, there is a block \mathcal{B}_I which computes the internal variables z_i according to their dynamics $\dot{z}(t) = g(z(t), y(t))$, and there are m blocks $\mathcal{B}_1, \dots, \mathcal{B}_m$ which compute the control variables u_j respectively. In the structure in Figure 1b, each control block \mathcal{B}_j updates its own copy of the internal variables and computes the control variable u_j . Thus, the main difference between these two structures is that the first has a separate block to compute z while the latter incorporates the computation of z into each control block \mathcal{B}_j . In this paper, we will only consider the structure in Figure 1a, however the results are readily applicable to other structures¹.

The ideal model-level semantics assumes that all control blocks of controller $\mathcal{M}_C = (\mathcal{B}_1, \dots, \mathcal{B}_r)$ are “computed” *instantaneously* and *simultaneously*. Of course, any soft-

¹See the remark at the end of section 3.3

Fig. 1: Two structures of \mathcal{M}_C

ware implementation of the controller \mathcal{M}_C will violate both assumptions. Moreover, ideal differentiation and integration computations assumed by the model-level semantics cannot be satisfied by the implementation, where approximation algorithms must be used.

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 $\{\mathcal{B}_0, \mathcal{B}_I, \mathcal{B}_1, \dots, \mathcal{B}_m\}$. 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 preemption, and when one control block completes its execution, the next block can start immediately. For example, given a controller $\mathcal{M}_C = (\mathcal{B}_I, \mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3)$ with one internal and three control blocks, possible dispatch sequences are the uniform sequence $(\mathcal{B}_I \mathcal{B}_1 \mathcal{B}_2 \mathcal{B}_3)^\omega$ or the nonuniform sequence $(\mathcal{B}_I \mathcal{B}_1 \mathcal{B}_I \mathcal{B}_2 \mathcal{B}_I \mathcal{B}_1 \mathcal{B}_I \mathcal{B}_3 \mathcal{B}_0)^\omega$ that also includes idling. Note that a dispatch sequence contains only ordering information, and is thus 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 to a fixed number of such slots. The computation of each control block \mathcal{B}_j (or \mathcal{B}_I) consists of reading the relevant plant output variables using sensors, updating the control variable u_j (or internal variables z), and finally writing the computed control value to the actuators at the end of its allotted time. The computation time of each control block is captured by a *timing function* $\tau : \{\mathcal{B}_I, \mathcal{B}_1, \dots, \mathcal{B}_m\} \rightarrow \mathbb{Z}^+$ which associates to each control block the number of time slots needed to execute it. Without loss of generality, we assume $\tau(\mathcal{B}_0) = 1$.

Informally, 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 $\llbracket \mathcal{M} \rrbracket_{(\rho, \tau, \delta)}$, 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 .

Formally, to define the implementation semantics, we note that the 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 $t_i = \sum_{k=0}^{i-1} \tau(\rho(k))\delta$ for $i \geq 1$. Except for t_0 , these are the precise timing instants when a control block completes its computation and its outputs are updated. In addition, we recursively define the time instants $\Delta_I(i)$, $i = 0, 1, 2, \dots$, as $\Delta_I(0) = 0$ and

$$\Delta_I(i+1) = \begin{cases} \Delta_I(i) + \tau(\rho(i))\delta & \text{if } \rho(i) \neq \mathcal{B}_I \\ \tau(\mathcal{B}_I)\delta & \text{if } \rho(i) = \mathcal{B}_I \end{cases} \quad (4)$$

in order to model the time elapsed since the last execution of block \mathcal{B}_I . Furthermore, the derivatives of plant outputs y are numerically computed only in blocks \mathcal{B}_j , $1 \leq j \leq m$, and are held in internal variables w . To model the time elapses between the updates of w , we define the sequence of time instants $\Delta_D(i)$, $i = 0, 1, 2, \dots$, as $\Delta_D(0) = 0$ and

$$\Delta_D(i+1) = \begin{cases} \Delta_D(i) + \tau(\rho(i))\delta & \text{if } \rho(i) \in \{\mathcal{B}_0, \mathcal{B}_I\} \\ \tau(\rho(i))\delta & \text{if } \rho(i) \notin \{\mathcal{B}_0, \mathcal{B}_I\}. \end{cases} \quad (5)$$

The implementation semantics $\llbracket \mathcal{M} \rrbracket_{(\rho, \tau, \delta)}$ can now be defined as the collection of trajectories $(x(t), y(t), u(t), z(t))$ that for all $t \geq 0$ satisfy the continuous-time plant dynamics defined by Eq. (1), and the following controller implementation constraints for every $1 \leq j \leq m$ and $i \geq 0$,

Initialization

$$z(0) = 0, \quad w(0) = 0, \quad u_j(0) = 0 \quad (6)$$

Inter-sample

$$z(t) = z(t_i), \quad w(t) = w(t_i), \quad u_j(t) = u_j(t_i) \quad \text{for } t_i < t < t_{i+1} \quad (7)$$

Controller Updates

$$u_j(t_{i+1}) = \begin{cases} u_j(t_i) & \text{if } \rho(i) \neq \mathcal{B}_j \\ k_j(y(t_i), w(t_{i+1}), z(t_i), u_1(t_i) \dots u_{j-1}(t_i)) & \text{if } \rho(i) = \mathcal{B}_j \end{cases} \quad (8)$$

Numerical Differentiation

$$w(t_{i+1}) = \begin{cases} w(t_i) & \text{if } \rho(i) \in \{\mathcal{B}_0, \mathcal{B}_I\} \\ D(y(t_i), y(t_i - \Delta_D(i)), \Delta_D(i), w(t_i)) & \text{if } \rho(i) \notin \{\mathcal{B}_0, \mathcal{B}_I\} \end{cases} \quad (9)$$

Numerical Integration

$$z(t_{i+1}) = \begin{cases} z(t_i) & \text{if } \rho(i) \neq \mathcal{B}_I \\ G(z(t_i), y(t_i), z(t_i - \Delta_I(i)), y(t_i - \Delta_I(i)), \Delta_I(i)) & \text{if } \rho(i) = \mathcal{B}_I \end{cases} \quad (10)$$

Implementation constraints (6) capture initialization, constraints (7) express the fact that during the execution of any control block all computed values remain constant, equation (8) describes the equations for updating the control variables, (9) represents the numerical computation of the derivatives of plant outputs, and (10) captures the numerical scheme for integrating the internal variables when control block \mathcal{B}_I is scheduled. The implementation constraints clearly show that $u_j(t)$, $z(t)$, and $w(t)$ are piecewise-constant signals.

Note that function G can be used to model a variety of *fixed-step* numerical integration algorithms. Different choices for this function can model different choices for well known numerical algorithms as shown in Table I. Euler is a one-step method, whereas

Table I. Function G for some well known numerical integration algorithms

Euler
$z(t_{i+1}) = z(t_i) + \Delta_I(i)g(z(t_i), y(t_i))$
Trapezoid
$z(t_{i+1}) = z(t_i) + \frac{\Delta_I(i)}{2}(g(z(t_i), y(t_i)) + g(z(t_i - \Delta_I(i)), y(t_i - \Delta_I(i))))$
Adams-Bashforth
$z(t_{i+1}) = z(t_i) + \frac{\Delta_I(i)}{2}(3g(z(t_i), y(t_i)) - g(z(t_i - \Delta_I(i)), y(t_i - \Delta_I(i))))$

Table II. Two numerical differentiation algorithms

Backward Difference
$w(t_{i+1}) = \frac{1}{\Delta_D(i)}(y(t_i) - y(t_i - \Delta_D(i)))$
Tustin's Approximation
$w(t_{i+1}) = \frac{2}{\Delta_D(i)}(y(t_i) - y(t_i - \Delta_D(i))) - w(t_i)$

Trapezoid and Adams-Bashforth are two-step methods that utilize information in two different time instants in integrating the dynamics of the internal variables. Higher order methods could easily be modeled at the expense of more variables, one for each step². Note that every time the integration block \mathcal{B}_I is executed, the numerical methods needs to integrate the internal dynamics starting from the last time block \mathcal{B}_I was executed. Therefore the dispatch sequence will have a direct effect on the size of the integration step $\Delta_I(i)$ and therefore the quality of the approximation.

Similarly, function D can be used to model the numerical computation of the derivatives of y . Two commonly used algorithms for approximating the derivatives are given in Table II.

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

$$(\tilde{x}(t), \tilde{y}(t), \tilde{u}(t), \tilde{z}(t)) = \llbracket \mathcal{M} \rrbracket_{(\rho, \tau, \delta)}(x(0)) \quad (11)$$

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

$$e_{\mathcal{M}}(\rho, \tau, \delta, x(0)) = \int_0^{+\infty} \|y(t) - \tilde{y}(t)\|_2^2 dt \quad (12)$$

We are therefore measuring the implementation error in the L_2 sense. Note that we are measuring the implementation error on the output variables of the overall closed loop system, rather than the error on the controller variables. We are therefore directly measuring the effect of controller implementation on the performance of the overall feedback interconnection.

Given a feedback control model \mathcal{M} and a set of initial plant states X_0 , we will say that the implementation $(\rho_1, \tau_1, \delta_1)$ is more accurate than the implementation

²Note that Runge-Kutta methods, even though popular for simulation, are problematic for code generation, as they require evaluations such as $g(z(t_i + \Delta_I(i)), y(t_i + \Delta_I(i)))$, which in turns requires predicting the sensed input $y(t_i + \Delta_I(i))$ in the future.

$(\rho_2, \tau_2, \delta_2)$ (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 $(\rho_2, \tau_2, \delta_2)$ for all initial states:

$$\forall x(0) \in X_0 \quad e_{\mathcal{M}}(\rho_1, \tau_1, \delta_1, x(0)) \leq e_{\mathcal{M}}(\rho_2, \tau_2, \delta_2, x(0)). \quad (13)$$

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

3. ERROR ANALYSIS

In this section we provide a method for the computation of the implementation error $e_{\mathcal{M}}(\rho, \tau, \delta, x(0))$ for an important class of plants and embedded controllers. We assume that the plant model is a linear, time-invariant (LTI) system:

$$\mathcal{M}_P : \begin{cases} \dot{x}(t) = A_p x(t) + B_p u(t) \\ y(t) = C_p x(t) \\ x(0) \in \mathbb{R}^n \end{cases} \quad (14)$$

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 will also assume that the feedback controller model \mathcal{M}_C has the following form

$$\mathcal{M}_C : \begin{cases} \dot{z}(t) = A_c z(t) + B_c y(t) \\ u(t) = K_P y(t) + K_I z(t) + K_D \dot{y}(t) + L_c u(t) \\ z(0) = 0 \end{cases} \quad (15)$$

where $A_c \in \mathbb{R}^{q \times q}$, $B_c \in \mathbb{R}^{q \times p}$, $K_P \in \mathbb{R}^{m \times p}$, $K_I \in \mathbb{R}^{m \times q}$, and $L_c \in \mathbb{R}^{m \times m}$. The controller model $\mathcal{M}_C = (\mathcal{B}_I, \mathcal{B}_1, \dots, \mathcal{B}_m)$ consists of one control block \mathcal{B}_I for integrating all the internal variables z_i , and one control block \mathcal{B}_j for computing each variable u_j . Note that the assumption that the dependence between control variables are acyclic implies that L_c is a lower triangular matrix. In the special case where $A_c = 0$ and $B_c = I$, since the internal variables z_i simply integrate the output variables y_i , we obtain the familiar equation

$$u(t) = K_P y(t) + K_I \int_0^t y(\tau) d\tau + K_D \frac{dy}{dt}(t) + L_c u(t).$$

We can thus readily see that the controllers captured in this class are the so-called proportional-integral-derivative (PID) controllers.

Without loss of generality and for the sake of simplicity, we will assume that all execution of all control blocks require one time slot. Thus, $\tau(\mathcal{B}_I) = 1$ and $\tau(\mathcal{B}_j) = 1$, $1 \leq j \leq m$, and for all $i \geq 0$, $t_i = i\delta$. If a block \mathcal{B}_j takes more than one time slot for its computation, that is $\tau(\mathcal{B}_j) = k$ for $k > 1$, any execution of \mathcal{B}_j in the dispatch sequence can be replaced by the sequence $(\mathcal{B}_Y \mathcal{B}_0 \dots \mathcal{B}_0 \mathcal{B}_j^*)$ where \mathcal{B}_0 is executed $k - 2$ times. In this sequence, \mathcal{B}_Y , with $\tau(\mathcal{B}_Y) = 1$, is a special block that saves the current values of outputs y to certain internal variables, and \mathcal{B}_j^* , with $\tau(\mathcal{B}_j^*) = 1$, is the same as \mathcal{B}_j except that it uses the values in the internal variables for y . The results in this section are still valid, although the calculation may be more complicated.

The challenge now is to compute the difference $e_{\mathcal{M}}(\rho, \tau, \delta, x(0))$ between the model-semantics and implementation-semantics for the class of plants and controllers described. Our first result along this direction is a sampling result that allows us to exactly compute the implementation error, defined over all $t \geq 0$, by using plant, output, and internal state information on the timing instants t_i .

THEOREM 3.1. *There exists a computable, symmetric, positive semi-definite matrix Q such that the implementation error defined by equation (12) is equal to*

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

where vector $\psi(t)$ is defined as $\psi(t) = [x(t)^T \ z(t)^T \ \tilde{x}(t)^T \ \tilde{u}(t)^T]^T$.

The proof of this theorem is given in the Appendix. Theorem 3.1 has effectively replaced a continuous integration with an infinite sum. It is worth noting that Theorem 3.1 holds for any controller structure \mathcal{M}_C of model (15), for example Figures 1a and 1b, as long as the values $x(t_i)$, $z(t_i)$ from the model-level semantics and the values $\tilde{x}(t_i)$, $\tilde{u}(t_i)$ from the implementation level semantics can be obtained. We will now define two discrete-time dynamical systems, that will generate these desired sequences.

3.1. Model-Level Semantics

The evolution of the closed loop feedback system for the model-level semantics is described in Equation (27). Given the timing sequence $t_i = i\delta$, we can directly generate the desired sequences $x(t_i)$, $z(t_i)$ by considering successive iterations of the following discrete-time, linear, time-invariant system:

$$\begin{bmatrix} x(t_{i+1}) \\ z(t_{i+1}) \end{bmatrix} = E \begin{bmatrix} x(t_i) \\ z(t_i) \end{bmatrix} \quad x(t_0) = x(0) \quad z(t_0) = 0 \quad (17)$$

where $E = e^{\delta \hat{A}}$ is simply the matrix exponential of $\delta \hat{A}$. Because of Theorem 3.1, the model level semantics that are relevant for the computation of the implementation error are all captured in model (17).

3.2. Implementation-Level Semantics

Our goal is now to develop a similar discrete-time model for the implementation semantics. The main idea is that the execution of any particular control block can be modeled as a discrete-time linear system. However, as the dispatch sequence switches control blocks, this results in a switching discrete-time linear system. Furthermore, since the sequence of executions of the control blocks is periodic, the implementation semantics will be captured by a periodic, linear, time-varying (PLTV) system.

Because the numerical differentiation and integration algorithms require past values of \tilde{y} , we need to introduce memory variables which capture the relevant information for the computations. Specifically, memory variables $\tilde{y}_m(t_i) = \tilde{y}(t_i - \Delta_D(t_i))$ are used for differentiation and memory variables $\tilde{z}_m(t_i) = A_c \tilde{z}(t_i) + B_c C_p \tilde{x}(t_i)$ for integration. Let us define the vector $\tilde{\vartheta}(t) = [\tilde{x}(t)^T \ \tilde{z}(t)^T \ \tilde{z}_m(t)^T \ \tilde{w}(t)^T \ \tilde{y}_m(t)^T \ \tilde{u}(t)^T]^T$ which consists of all variables of the implementation semantics. For each control block, we will develop a discrete-time model that updates $\tilde{\vartheta}$ according to the evolution of the system.

Modeling control blocks \mathcal{B}_j . Consider a control block \mathcal{B}_j , $1 \leq j \leq m$, executed during the time interval $[t_i, t_{i+1}]$. First, the derivatives $\dot{\tilde{y}}(t_i)$ are numerically computed and stored in variables $\tilde{w}(t_{i+1})$. Then the control variable u_j is updated while the other control variables \tilde{u}_k , for $k \neq j$, are not changed.

In the execution of control block \mathcal{B}_j , the values of $\tilde{y}(t_i)$ are saved to memory variables \tilde{y}_m , that is $\tilde{y}_m(t_{i+1}) = \tilde{y}(t_i) = C_p \tilde{x}(t_i)$, and \tilde{w} are updated according to the chosen

algorithm for differentiation computation:

Backward Difference

$$\tilde{w}(t_{i+1}) = \frac{1}{\Delta_D(i)} (C_P \tilde{x}(t_i) - \tilde{y}_m(t_i)) \quad (18)$$

Tustin's Approximation

$$\tilde{w}(t_{i+1}) = \frac{2}{\Delta_D(i)} (C_P \tilde{x}(t_i) - \tilde{y}_m(t_i)) - \tilde{w}(t_i) \quad (19)$$

Then the control variable \tilde{u}_j is computed as

$$\tilde{u}_j(t_{i+1}) = [K_P]_j C_P \tilde{x}(t_i) + [K_I]_j \tilde{z}(t_i) + [K_D]_j \tilde{w}(t_{i+1}) + [L_c]_j \tilde{u}(t_i)$$

where $[K_C]_j$, $[K_I]_j$, $[K_D]_j$ and $[L_c]_j$ denote the j^{th} rows of matrices K_C , K_I , K_D and L_c respectively. Note that $\tilde{u}_j(t_{i+1})$ depends on $\tilde{w}(t_{i+1})$, thus its expression in terms of the components of $\tilde{\vartheta}(t_i)$ is dependent of the chosen differentiation method. For example, if the Tustin's Approximation method is used, i.e. $\tilde{w}(t_{i+1})$ is updated according to Eq. (19), $\tilde{u}_j(t_{i+1})$ can be written as

$$\begin{aligned} \tilde{u}_j(t_{i+1}) = & \left([K_P]_j + \frac{2}{\Delta_D(i)} [K_D]_j \right) C_P \tilde{x}(t_i) + [K_I]_j \tilde{z}(t_i) \\ & - [K_D]_j \tilde{w}(t_i) - \frac{2}{\Delta_D(i)} [K_D]_j \tilde{y}_m(t_i) + [L_c]_j \tilde{u}(t_i) \end{aligned}$$

Because the other control variables \tilde{u}_k , for $k \neq j$, are not changed during the execution of \mathcal{B}_j , we can write vector $\tilde{u}(t_{i+1})$ as

$$\tilde{u}(t_{i+1}) = X_j(i) \tilde{x}(t_i) + Z_j \tilde{z}(t_i) + W_j(i) \tilde{w}(t_i) + Y_j(i) \tilde{y}_m(t_i) + (I + U_j + V_j) \tilde{u}(t_i)$$

where matrices $X_j(i)$, Z_j , $W_j(i)$, $Y_j(i)$, U_j and V_j are defined appropriately. Specifically, $X_j(i)$, Z_j , $W_j(i)$, $Y_j(i)$ and U_j are the matrices whose j^{th} rows are respectively the coefficients of $\tilde{x}(t_i)$, $\tilde{z}(t_i)$, $\tilde{w}(t_i)$, $\tilde{y}_m(t_i)$ and $\tilde{u}(t_i)$ in the expression of $\tilde{u}_j(t_{i+1})$, and all the other rows are 0. V_j is the matrix whose coefficients are all zero except the j^{th} element of its diagonal $[V_j]_{j,j} = -1$.

Having obtained the equation for $\tilde{u}(t_{i+1})$ and given that \tilde{z} and \tilde{z}_m are not changed by \mathcal{B}_j , we can now write the discrete-time system that updates $\tilde{\vartheta}$ after the execution of \mathcal{B}_j . If the Tustin's Approximation method is used, we have that

$$\tilde{\vartheta}(t_{i+1}) = \begin{bmatrix} e^{\delta A_p} & 0 & 0 & 0 & 0 & \alpha_p(\delta) \\ 0 & I & 0 & 0 & 0 & 0 \\ 0 & 0 & I & 0 & 0 & 0 \\ \frac{2}{\Delta_D(i)} C_P & 0 & 0 & -I & -\frac{2}{\Delta_D(i)} I & 0 \\ C_P & 0 & 0 & 0 & 0 & 0 \\ X_j(i) & Z_j & 0 & W_j(i) & Y_j(i) & I + U_j + V_j \end{bmatrix} \tilde{\vartheta}(t_i) = E_{\mathcal{B}_j}(i) \tilde{\vartheta}(t_i).$$

Similar equations can be obtained for other differentiation methods. Note that for $i = 0$, $\frac{1}{\Delta_D(0)} = 0$. Also observe that if the dispatch sequence ρ is periodic, the matrix $E_{\mathcal{B}_j}(i)$ is periodic after the first period of ρ .

Modeling integration block \mathcal{B}_I . Let us consider the evolution of the implementation semantics when integration block \mathcal{B}_I is executed. Since the execution of \mathcal{B}_I does not change the control variables, we have that $\tilde{u}(t_{i+1}) = \tilde{u}(t_i)$. On the time interval $[t_i, t_{i+1}]$, the plant evolves continuously according to Eq. (28) in the Appendix. By integration we thus obtain that

$$\tilde{x}(t_{i+1}) = e^{\delta A_p} \tilde{x}(t_i) + \alpha_p(\delta) \tilde{u}(t_i) \quad (20)$$

where $\alpha_p(\delta) = \int_0^\delta e^{A_p \tau} B_p d\tau$, which reduces to $\alpha_p(\delta) = (e^{\delta A_p} - I)A_p^{-1}B_p$ if A_p is invertible. The evolution of the internal variables z_i is captured by the equations describing the numerical algorithm that integrates the equations (15). Recalling that $\tilde{y}(t_i) = C_p \tilde{x}(t_i)$, we obtain following discrete-time models for the numerical integrators above

Euler

$$\tilde{z}(t_{i+1}) = (I + \Delta_I(i)A_c) \tilde{z}(t_i) + \Delta_I(i)B_c C_p \tilde{x}(t_i)$$

Trapezoid

$$\begin{aligned} \tilde{z}(t_{i+1}) &= (I + \frac{1}{2}\Delta_I(i)A_c) \tilde{z}(t_i) + \frac{1}{2}\Delta_I(i)B_c C_p \tilde{x}(t_i) + \frac{1}{2}\Delta_I(i)\tilde{z}_m(t_i) \\ \tilde{z}_m(t_{i+1}) &= A_c \tilde{z}(t_i) + B_c C_p \tilde{x}(t_i) \end{aligned}$$

Adams-Bashforth

$$\begin{aligned} \tilde{z}(t_{i+1}) &= (I + \frac{3}{2}\Delta_I(i)A_c) \tilde{z}(t_i) + \frac{3}{2}\Delta_I(i)B_c C_p \tilde{x}(t_i) - \frac{1}{2}\Delta_I(i)\tilde{z}_m(t_i) \\ \tilde{z}_m(t_{i+1}) &= A_c \tilde{z}(t_i) + B_c C_p \tilde{x}(t_i) \end{aligned}$$

Note that more precise, higher order (multi-step) methods can be easily considered by using more memory variables.

Collecting all the above equations allows us to construct a discrete-time linear system that models the evolution of the implementation when the integration block is executed. For example, for the two-step Adams-Bashforth scheme, the discrete-time model is

$$\tilde{\vartheta}(t_{i+1}) = E_{B_I}(i)\tilde{\vartheta}(t_i) = \begin{bmatrix} e^{\delta A_p} & 0 & 0 & 0 & 0 & \alpha_p(\delta) \\ \frac{3\Delta_I(i)}{2}B_c C_p & I + \frac{3\Delta_I(i)}{2}A_c & -\frac{\Delta_I(i)}{2} & 0 & 0 & 0 \\ B_c C_p & A_c & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & I & 0 & 0 \\ 0 & 0 & 0 & 0 & I & 0 \\ 0 & 0 & 0 & 0 & 0 & I \end{bmatrix} \tilde{\vartheta}(t_i)$$

Note that matrix $E_{B_I}(i)$ is not fixed but depends on $\Delta_I(i)$. However, since we assume that ρ is periodic, $\Delta_I(i)$ is periodic after the first period of the dispatch sequence. For Euler, Trapezoid and other numerical schemes we can obtain similar discrete-time models. Note that for the first order Euler method, there is no need for the extended $\tilde{z}_m(t_i)$ variables.

Modeling idle block \mathcal{B}_0 . The execution of the idle block \mathcal{B}_0 does not affect either the internal or control variables. Therefore we have that $\tilde{u}(t_{i+1}) = \tilde{u}(t_i)$, $\tilde{w}(t_{i+1}) = \tilde{w}(t_i)$, $\tilde{y}_m(t_{i+1}) = \tilde{y}_m(t_i)$, $\tilde{z}(t_{i+1}) = \tilde{z}(t_i)$, and $\tilde{z}_m(t_{i+1}) = \tilde{z}_m(t_i)$ in case a higher order integration method is used. On the time interval $[t_i, t_{i+1}]$, the plant still evolves continuously according to equation (20). Therefore, the value of the variables are modified by the execution of the control block \mathcal{B}_0 according to the following discrete-time system

$$\tilde{\vartheta}(t_{i+1}) = E_{\mathcal{B}_0}\tilde{\vartheta}(t_i) = \begin{bmatrix} e^{\delta A_p} & 0 & 0 & 0 & 0 & \alpha_p(\delta) \\ 0 & I & 0 & 0 & 0 & 0 \\ 0 & 0 & I & 0 & 0 & 0 \\ 0 & 0 & 0 & I & 0 & 0 \\ 0 & 0 & 0 & 0 & I & 0 \\ 0 & 0 & 0 & 0 & 0 & I \end{bmatrix} \tilde{\vartheta}(t_i)$$

Note that unlike matrices $E_{\mathcal{B}_j}$ and E_{B_I} , $E_{\mathcal{B}_0}$ is fixed.

From dispatch sequences to PLTV systems. Let us consider a dispatch sequence ρ defining the order in which the control blocks have to be executed. The sequences $\tilde{x}(t_i)$, $\tilde{z}(t_i)$,

$\tilde{z}_m(t_i)$, $\tilde{w}(t_i)$, $\tilde{y}_m(t_i)$, and $\tilde{u}(t_i)$, for $i \geq 0$, can be determined from the following discrete-time dynamical system:

$$\tilde{\vartheta}(t_{i+1}) = E(i)\tilde{\vartheta}(t_i) \quad (21)$$

where $E(i) = E_{\rho(i)}(i)$. Therefore the implementation values at instants t_i are captured by a discrete-time linear system that is time-varying. Furthermore, since we assume that ρ is periodic (let n_ρ denote its period), this dynamical system is a PLTV system.

3.3. Error Computation

Having modeled both the model-level and implementation level semantics at the instants t_i , we define a system which describes both the discretized model-level semantics and the discrete-time implementation level semantics. Its state is the vector $\bar{\psi}(t)$ defined as

$$\bar{\psi}(t)^T = [x(t)^T \ z(t)^T \ \tilde{x}(t)^T \ \tilde{z}(t)^T \ \tilde{z}_m(t)^T \ \tilde{w}(t)^T \ \tilde{y}_m(t)^T \ \tilde{u}(t)^T].$$

Then,

$$\bar{\psi}(t_{i+1}) = \bar{E}(i) \bar{\psi}(t_i) = \begin{bmatrix} E & 0 \\ 0 & E(i) \end{bmatrix} \bar{\psi}(t_i) \quad (22)$$

Note that $\bar{E}(i)$ is periodic after the first period of the dispatch sequence. Thus the composite system is also a PLTV system of period n_ρ . Using so-called lifting techniques for the analysis of PLTV systems (see for instance [S.Bittanti and P.Colaneri 2000]), which transform a periodic time-varying system into a higher order linear time-invariant system, we can state the main result of this paper.

THEOREM 3.2. *The implementation error is exactly equal to*

$$e_{\mathcal{M}}(\rho, \tau, \delta, x(0)) = \bar{\psi}(0)^T \hat{\mathcal{O}} \bar{\psi}(0) \quad (23)$$

where $\hat{\mathcal{O}} = \hat{G}_0^T \hat{Q} \hat{G}_0 + \hat{E}_0^T \mathcal{O} \hat{E}_0$, and \mathcal{O} is solution to the following Lyapunov equation

$$\mathcal{O} = \hat{E}^T \mathcal{O} \hat{E} + \hat{G}^T \hat{Q} \hat{G} \quad (24)$$

for implementation-dependent matrices \hat{E}_0 , \hat{G}_0 , \hat{E} , \hat{G} , \hat{Q} (see proof). $\bar{\psi}(0)$ can be computed by $\bar{\psi}(0) = Hx(0)$ where $H = [I \ 0 \ I \ 0 \ 0 \ 0 \ 0 \ 0]^T$.

Theorem 3.2 states that the implementation error can be computed *exactly* for this class of systems and controllers, and furthermore, the L_2 -norm error is a global quadratic function of the initial state x_0 . The proof of this theorem is given in the Appendix.

In implementing the controller, we are only interested in implementations that follow the ideal system, in the sense that the corresponding implementation error $e_{\mathcal{M}}(\rho, \tau, \delta, x(0))$ is finite for all $x(0) \in \mathbb{R}^n$. From the Lyapunov equation (24) in Theorem 3.2, it is straightforward to see that this can only be achieved if matrix \hat{E} is stable [Rugh 1996]. We thus have the following corollary.

COROLLARY 3.3. *The implementation error $e_{\mathcal{M}}(\rho, \tau, \delta, x(0))$ for a given implementation (ρ, τ, δ) is finite for all $x(0) \in \mathbb{R}^n$ if and only if matrix \hat{E} is stable, that is all eigenvalues of \hat{E} are inside the unit circle on the complex plane.*

Remark: Although the results in this section were developed for a specific structure of controller \mathcal{M}_C (Figure 1a), they are readily applicable to other structures of \mathcal{M}_C . Indeed, Theorem 3.1 is independent of the controller structure. Similar to section 3.2,

for each control block in the new structure, a discrete-time system that models the system evolution during its execution can be obtained. This allows us to write the PLTV system (22) of the implementation and to apply Theorem 3.2 as well as Corollary 3.3.

4. APPLICATIONS IN ANALYSIS AND DESIGN OF IMPLEMENTATIONS

In section 3, we presented the main result of this paper: given a feedback control model \mathcal{M} , an initial state $x(0)$ and an implementation (ρ, τ, δ) , we can compute exactly the implementation error, thus can assess the quality of the implementation. Moreover, the L_2 -norm error is a quadratic function of $x(0)$. In this section, we discuss some applications of this result in the analysis and design of controller implementations.

4.1. Analysis of Implementations

For a specific implementation (ρ, τ, δ) of a control model \mathcal{M} , Theorem 3.2 provides a quantitative measure of its quality by allowing us to compute the implementation error as a quadratic function of the initial state. Intuitively, the smaller the error is, the better the implementation is. However, in practice, a precise initial state is usually not known but a bounded set of initial states X_0 . In this case, we are more interested in the upper bound of the implementation error for X_0 , which can be computed by solving the quadratic program:

$$\begin{aligned} & \text{maximize} && x^T H^T \hat{O} H x \\ & \text{subject to} && x \in X_0. \end{aligned}$$

If X_0 is a convex set, this quadratic program can be solved efficiently using convex optimization techniques [Boyd and Vandenberghe 2004]. Alternatively, the spectral norm $\|H^T \hat{O} H\|_2$ can be used as a measure of the implementation quality, especially when X_0 is not provided or is unbounded.

Theorem 3.2 also provides a criterion for comparing two different time-triggered implementations of an embedded controller. Let $(\rho_1, \tau_1, \delta_1)$ and $(\rho_2, \tau_2, \delta_2)$ be two implementations that satisfy the condition in Corollary 3.3 so that both implementation errors are finite. Given a set of initial states X_0 , we have $(\rho_1, \tau_1, \delta_1) \preceq_{\mathcal{M}} (\rho_2, \tau_2, \delta_2)$ if for all $x(0) \in X_0$, $x(0)^T H^T \hat{O}_1 H x(0) \leq x(0)^T H^T \hat{O}_2 H x(0)$, or equivalently $0 \leq x(0)^T H^T (\hat{O}_2 - \hat{O}_1) H x(0)$. This is equivalent to checking whether

$$\min_{x(0) \in X_0} x(0)^T H^T (\hat{O}_2 - \hat{O}_1) H x(0) \geq 0,$$

which can be performed using convex programming ([Boyd and Vandenberghe 2004]) if X_0 is a convex set. On the other hand, if $X_0 = \mathbb{R}^n$, then checking whether $0 \leq x(0)^T H^T (\hat{O}_2 - \hat{O}_1) H x(0)$ for all $x(0) \in \mathbb{R}^n$, that is $(\rho_1, \tau_1, \delta_1)$ is *globally* better than $(\rho_2, \tau_2, \delta_2)$, reduces to checking whether $H^T (\hat{O}_2 - \hat{O}_1) H$ is a positive semi-definite matrix.

4.2. Design of Dispatch Sequences

In designing a time-triggered implementation of a given control model \mathcal{M} , the main problem is choosing a dispatch sequence ρ that gives good performance while conforming to certain requirements, e.g. the minimum ratio of idling time. Automaton-based schedulers ([Alur and Weiss 2008]) offer a flexible and dynamic approach to this scheduling problem. However, in this paper, we only consider static periodic dispatch sequences ρ in the form of a finite string that repeats. If we use $\|H^T \hat{O} H\|_2$ as the performance measure of an implementation, the scheduling problem is equivalent to

solving the minimization problem:

$$\begin{aligned} & \text{minimize} && \|H^T \hat{O}(\rho)H\|_2 \\ & \text{subject to} && \rho \in \Pi, \end{aligned}$$

where Π is the set of feasible dispatch sequences and $\hat{O}(\rho)$ depends on ρ . An obvious approach to solving this minimization is the brute-force search algorithm: compute the performance measure for every $\rho \in \Pi$ up to a certain length, then choose the best one. Although it is appropriate for small models \mathcal{M} , the brute-force algorithm does not scale well with the size of the system. In order to speed up the search, certain heuristics could be used to reduce the search space. For instance when one part of the plant has faster dynamics than the rest, we can restrict our interest to dispatch sequences that schedule the control computation for this part more often than the others. Another useful heuristics is to restrict the frequency of the idling block \mathcal{B}_0 in ρ to the minimum requirement because in general, scheduling more executions of \mathcal{B}_0 in the dispatch sequence results in worse performance. Better techniques for designing dispatch sequences or improving the brute-force search is a subject for future research.

5. COMPUTATIONAL TOOLS

5.1. Analysis Tool

We implemented the analysis method discussed in section 3 in MATLAB. Given the model $\mathcal{M} = \langle \mathcal{M}_P, \mathcal{M}_C \rangle$, the implementation specifics (ρ, τ, δ) , and an initial state $x(0)$, the MATLAB program computes the implementation error $e_{\mathcal{M}}(\rho, \tau, \delta, x(0))$. One difficulty in implementing this algorithm is the computation of matrix Q given in equation (30), which involves integrals of exponentials of (possibly singular) matrices. Although numerical integration can be used to compute Q , it will fail if A is singular since in such case, $\alpha(t) = \int_0^t e^{A\tau} B d\tau$ cannot be simplified to $(e^{At} - I)A^{-1}B$. Symbolic computation (e.g. with MATHEMATICA) works when A is singular but it is inefficient, especially for large matrices. The technique used in our program to compute matrix Q employs the results from [Van Loan 1978]. The algorithm is summarized below.

If A is nonsingular then

 Compute

$$\exp\left(\delta \begin{bmatrix} -A^T & C^T C \\ 0 & A \end{bmatrix}\right) = \begin{bmatrix} R_{12} & R_{22} \\ 0 & R_{13} \end{bmatrix}, \quad \exp\left(\delta \begin{bmatrix} A & C^T C \\ 0 & 0 \end{bmatrix}\right) = \begin{bmatrix} R_{13} & R_{23} \\ 0 & R_{14} \end{bmatrix}$$

 Compute

$$M = \int_0^\delta e^{A^T t} C^T C e^{At} dt = R_{13}^T R_{22} \\ N = \int_0^\delta e^{A^T t} C^T C dt = R_{23}, \quad P = \int_0^\delta C^T C dt = \delta C^T C$$

 Compute

$$Q_{1,1} = M \\ Q_{1,2} = Q_{2,1}^T = (M - N)A^{-1}B \\ Q_{2,2} = B^T (A^{-1})^T (M - N - N^T + P)A^{-1}B$$

Else

 Compute

$$\exp\left(\delta \begin{bmatrix} -A^T & I & 0 & 0 \\ 0 & -A^T & C^T C & 0 \\ 0 & 0 & A & B \\ 0 & 0 & 0 & 0 \end{bmatrix}\right) = \begin{bmatrix} R_{11} & R_{21} & R_{31} & R_{41} \\ 0 & R_{12} & R_{22} & R_{32} \\ 0 & 0 & R_{13} & R_{23} \\ 0 & 0 & 0 & R_{14} \end{bmatrix}$$

 Compute

$$Q_{1,1} = R_{13}^T R_{22} \\ Q_{1,2} = Q_{2,1}^T = R_{13}^T R_{32} \\ Q_{2,2} = (B^T R_{13}^T R_{41}) + (B^T R_{13}^T R_{41})^T$$

End If

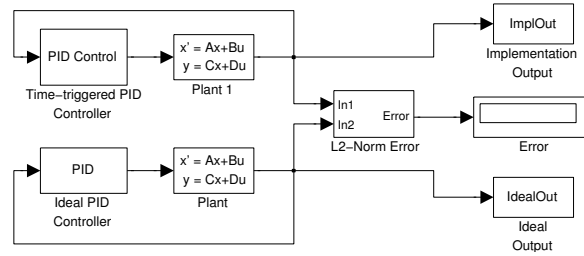


Fig. 2: SIMULINK Model For Error Computation

$$\mathbf{Return} \ Q = \begin{bmatrix} Q_{1,1} & Q_{1,2} \\ Q_{2,1} & Q_{2,2} \end{bmatrix}$$

Note that the computation does not involve integration, but uses exponential of constant matrices instead [Moler and Loan 2003]. Therefore, algorithmically, the construction of the implementation-dependent matrices \hat{E}_0 , \hat{G}_0 , \hat{E} , \hat{G} , \hat{Q} requires straightforward matrix operations (sums, products, exponentiation) on matrices, whereas solving Lyapunov equations is polynomial in the size of the matrices. The largest matrices in Theorem 3.2 are matrices \hat{G} and \hat{G}_0 which have $n_\rho \times (2n + 5p + m)$ rows and $2n + 5p + m$ columns.

5.2. Simulation Tool

We also developed a MATLAB/SIMULINK-based simulator for time-triggered real-time control systems. The simulator is based on the TRUETIME library [Henriksson et al. 2003] and can be used to simulate general time-triggered systems, not limited to the controller implementation considered in this paper. TRUETIME is a MATLAB toolbox which facilitates simulation of multitasking real-time kernel executing control tasks, modeled as ordinary SIMULINK blocks [Henriksson et al. 2003]. The main component of TRUETIME is a computer kernel block which executes user-defined tasks written in MATLAB code or C++ code. The tasks can generally do anything, such as control algorithms, network communication, and I/O interface. The computer kernel block supports most important features of an actual real-time kernel: interrupts and interrupt handlers, priorities and task scheduling policies (both predefined and user-defined scheduling policies can be used), periodic and aperiodic tasks, synchronization mechanisms (e.g. monitors, events, semaphores), timing control, I/O (A/D, D/A, and network), to name a few. TRUETIME also supports simulation of communication networks with various medium access control protocols (CSMA/CD, CSMA/CA, Round Robin, FDMA, TDMA) and user-defined transmission rate [Henriksson et al. 2003]. Therefore, TRUETIME is capable of simulating not only dynamic real-time control systems but networked control loops as well.

Our simulator is an extension of TRUETIME to time-triggered real-time systems. In general, any time-triggered control task, implemented in MATLAB code or C++ code, can be simulated. The scheduling can be periodic (specified as a periodic dispatch sequence) or aperiodic (e.g. specified by a finite state machine). In addition, for convenience, the commonly used PID controller is predefined with various integration and differentiation algorithms (see section 2.3). Given a plant model \mathcal{M}_P , a controller model \mathcal{M}_C and its time-triggered implementation (ρ, τ, δ) , for a particular initial state $x(0)$, we can construct the SIMULINK model shown in Figure 2 using our simulation library and simulate it in order to visualize the discrepancies between the output variables $y(t)$ and $\tilde{y}(t)$. All output plots in the following examples were generated by this simulator.

Table III. Implementation errors for various time-triggered platforms.

No.	δ	Integration	Differentiation	ρ	$\ H^T \hat{O}H\ _2$	$e_{\mathcal{M}}$
1	0.001	Euler	Backward Diff.	$(\mathcal{B}_1 \mathcal{B}_1 \mathcal{B}_2)^\omega$	21.9183	10.0058
2	0.001	Euler	Backward Diff.	$(\mathcal{B}_1 \mathcal{B}_2 \mathcal{B}_1)^\omega$	0.0394	0.5241
3	0.001	Euler	Backward Diff.	$(\mathcal{B}_1 \mathcal{B}_2 \mathcal{B}_1 \mathcal{B}_1)^\omega$	N/A	∞
4	0.001	Euler	Backward Diff.	$(\mathcal{B}_1 \mathcal{B}_2 \mathcal{B}_1 \mathcal{B}_1 \mathcal{B}_1)^\omega$	0.0640	0.6336
5	0.00075	Euler	Backward Diff.	$(\mathcal{B}_1 \mathcal{B}_1 \mathcal{B}_2)^\omega$	0.8523	1.9457
6	0.0005	Euler	Backward Diff.	$(\mathcal{B}_1 \mathcal{B}_2 \mathcal{B}_1 \mathcal{B}_1)^\omega$	0.0281	0.3704

6. NUMERICAL EXAMPLES

6.1. Example 1: PID Controller

Consider the plant

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \\ \dot{x}_3(t) \\ \dot{x}_4(t) \end{bmatrix} = \begin{bmatrix} -1020 & -156.3 & 0 & 0 \\ 128 & 0 & 0 & 0 \\ 0 & 0 & -10.2 & -2.002 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \end{bmatrix} + \begin{bmatrix} 8 & 0 \\ 0 & 0 \\ 0 & 0.5 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} u_1(t) \\ u_2(t) \end{bmatrix}$$

$$\begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix} = \begin{bmatrix} 0 & 4.8828 & 0 & 0 \\ 0 & 0 & 0 & 0.4 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \end{bmatrix}$$

which consists of two separate subsystems: one subsystem (captured by variables x_1, x_2) has much faster dynamics than the other (captured by variables x_3, x_4). A PID controller designed for this plant is

$$\begin{bmatrix} u_1(t) \\ u_2(t) \end{bmatrix} = K_P \begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix} + K_I \begin{bmatrix} \int_0^t y_1(\tau) d\tau \\ \int_0^t y_2(\tau) d\tau \end{bmatrix} + K_D \begin{bmatrix} dy_1(t)/dt \\ dy_2(t)/dt \end{bmatrix}$$

where

$$K_P = \begin{bmatrix} -116 & 0 \\ 0 & -250 \end{bmatrix} \quad K_I = \begin{bmatrix} -480 & 0 \\ 0 & -30 \end{bmatrix} \quad K_D = \begin{bmatrix} -0.2 & 0 \\ 0 & -20 \end{bmatrix}$$

Control variable u_1 regulates the faster subsystem, whereas u_2 regulates the slower subsystem. Table III summarizes the performance results for various dispatch sequences. For each implementation, the spectral norm $\|H^T \hat{O}H\|_2$ and the implementation error for initial state $x_1(0) = x_2(0) = x_3(0) = x_4(0) = 2$ are computed. Figure 3 shows the evolution of output $y_1(t)$ for various implementation choices. From Table III, it is observable that though the ideal closed loop system is stable, implementation $(\rho_3, \tau_3, \delta_3)$ destabilizes the plant. The uniform dispatch sequence ρ_1 produces a large error, while by slightly changing the dispatch sequence, implementation $(\rho_2, \tau_2, \delta_2)$ outperforms $(\rho_1, \tau_1, \delta_1)$ by a large factor. In fact, $(\rho_2, \tau_2, \delta_2)$ gives better performance than other implementations, even those on faster platforms. Thus, scheduling can have great effect on the overall performance of the system. Since the dynamics of subsystem 1 are faster than those of subsystem 2, it is predictable that allocating more time slots to control block \mathcal{B}_1 will result in better performance, as illustrated by implementation $(\rho_4, \tau_4, \delta_4)$. The above observations show that the performance of a controller can be considerably improved without changing the platform, however care should be taken when choosing the dispatch sequence. Furthermore, even on a slightly faster computer, the performance of $(\rho_5, \tau_5, \delta_5)$ may be worse than $(\rho_4, \tau_4, \delta_4)$ executing on a slower computer but with better scheduling. Nonetheless, for sufficiently faster platform $(\rho_6, \tau_6, \delta_6)$, the performance is improved greatly.

On a platform with $\delta = 0.001$ secs and Euler method for integration and Backward Difference method for differentiation, we can use the brute-force algorithm to search for the best periodic dispatch sequence ρ with n_ρ up to 8, subject to a minimum idling

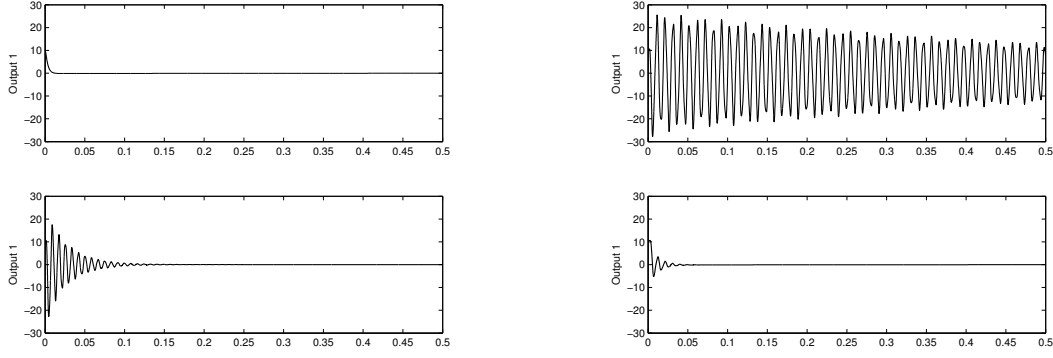


Fig. 3: Plots of y_1 of example 1 for the ideal semantics $\llbracket \mathcal{M} \rrbracket$ (top left), and implementations $\llbracket \mathcal{M} \rrbracket_{(\rho_1, \tau_1, \delta_1)}$ (top right), $\llbracket \mathcal{M} \rrbracket_{(\rho_5, \tau_5, \delta_5)}$ (bottom left), $\llbracket \mathcal{M} \rrbracket_{(\rho_2, \tau_2, \delta_2)}$ (bottom right).

Table IV. Best dispatch sequences for various idling time requirements.

Idling Time	Best ρ	$\ H^T \hat{O} H\ _2$
$\geq 0\%$	$(\mathcal{B}_1 \mathcal{B}_1 \mathcal{B}_1 \mathcal{B}_1 \mathcal{B}_I \mathcal{B}_2)^\omega$	0.0180
$\geq 10\%$	$(\mathcal{B}_1 \mathcal{B}_1 \mathcal{B}_1 \mathcal{B}_1 \mathcal{B}_I \mathcal{B}_0 \mathcal{B}_2)^\omega$	0.0235
$\geq 20\%$	$(\mathcal{B}_1 \mathcal{B}_0 \mathcal{B}_1 \mathcal{B}_1 \mathcal{B}_I \mathcal{B}_0 \mathcal{B}_2)^\omega$	0.0314
$\geq 50\%$	$(\mathcal{B}_2 \mathcal{B}_1 \mathcal{B}_I \mathcal{B}_0 \mathcal{B}_1 \mathcal{B}_0 \mathcal{B}_0 \mathcal{B}_0)^\omega$	0.0852

time requirement. Table IV summarizes the search results for various idling time requirements. These results show that a carefully chosen dispatch sequence can perform very well, even with 50% idling time. Using the heuristics that \mathcal{B}_1 , which computes the control output u_1 for the faster subsystem, should be scheduled more frequently than \mathcal{B}_2 , we managed to speed up the brute-force algorithm by up to two times.

6.2. Example 2: State-feedback Control with Observer

Given plant model (14), a state-feedback controller $u = Kx$ can be designed so that the closed-loop system meets the specifications (by placing its poles at desired positions on the complex plane). However, since state information $x(t)$ is generally not directly available, a state observer (or state estimator) is usually required to provide an estimate of $x(t)$, given input $u(t)$ and output $y(t)$. A Luenberger observer has the form

$$\dot{z}(t) = A_p z(t) + B_p u(t) + L(y(t) - C_p z(t)) \quad (25)$$

where L is designed so that $z(t)$ converges to $x(t)$ sufficiently fast as $t \rightarrow \infty$ [Antsaklis and Michel 1997]. Putting all together, we obtain the following model of controller \mathcal{M}_C :

$$\begin{cases} \dot{z}(t) = (A_p + B_p K - LC_p)z(t) + Ly(t) \\ u(t) = Kz(t) \\ z(0) = 0 \end{cases} \quad (26)$$

which is of the form (15).

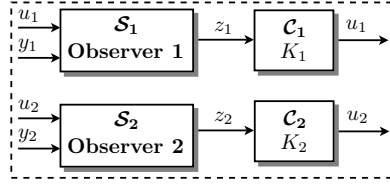


Fig. 4: Controller structure in example 2

Now consider two plants to be controlled using state-feedback with observers. The first plant has model:

$$\dot{x}(t) = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 2 & -1 \end{bmatrix} x(t) + \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 0 & 0 \end{bmatrix} u(t)$$

$$y(t) = [1 \ 0 \ 0] x(t)$$

with poles 0, 1, -2. The second plant has model:

$$\dot{x}(t) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & -2 & 3 \end{bmatrix} x(t) + \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} u(t)$$

$$y(t) = [1 \ 0 \ 0 \ 0] x(t)$$

with poles 2.414, $0.5 \pm 0.866i$, -0.414. Note that both plants are unstable, and the latter is “more unstable” in the sense that its poles are further to the right of the imaginary axis on the complex plane. State-feedback controllers and observers were designed for the first plant:

$$K_1 = \begin{bmatrix} 2 & -1 & -2 \\ -2 & 0 & 0.5 \end{bmatrix}, \quad L_1 = [8 \ 21 \ 22]^T$$

and for the second plant:

$$K_2 = \begin{bmatrix} -1.8799 & 4.7722 & 4.9236 & 2.2896 \\ -1.3906 & -5.1201 & -4.3603 & -7.1201 \end{bmatrix}, \quad L_2 = [23 \ 217 \ 1106 \ 3533]^T.$$

Assume that the state-feedback controllers are implemented in one computer, connected to the two plants through a time-triggered communication network. In each time-slot, the computer can only read from or write to one plant. Thus, a natural block structure of the controller is given in Figure 4. Block \mathcal{S}_1 is the observer which gives an estimate of state z_1 of the first plant. Block \mathcal{C}_1 computes control variable u_1 for this plant from the estimated z_1 . Similarly, \mathcal{S}_2 estimates state z_2 and \mathcal{C}_2 computes control variable u_2 for the second plant. Though this structure is different from that considered in section 3, similar analysis can be carried out to obtain the implementation error. Note that theorems 3.1 and 3.2 still hold for this case, provided the relevant matrices, for example $E(i)$, \hat{E} and \hat{G} , are appropriately computed. Table V shows some computational results for various implementations of the system, when $x(0) = [1 \ 1 \ 1]^T$ for the first plant and $x(0) = [0 \ 0.5 \ 0.5 \ 0.5]^T$ for the second plant. The simple dispatch sequence $(\mathcal{S}_1 \mathcal{C}_1 \mathcal{S}_2 \mathcal{C}_2)^\omega$ destabilizes the system, while by giving more computation time for observer blocks (\mathcal{S}_1 and \mathcal{S}_2), implementation $(\rho_2, \tau_2, \delta_2)$ results in a relatively small error. Moreover, by allocating more time-slots to observer \mathcal{S}_2 , implementation $(\rho_3, \tau_3, \delta_3)$

Table V. Implementation errors for Example 2

No.	δ	Integration	ρ	$e_{\mathcal{M}}$
1	0.002	Euler	$(\mathcal{S}_1 \mathcal{C}_1 \mathcal{S}_2 \mathcal{C}_2)^\omega$	∞
2	0.002	Euler	$(\mathcal{S}_1 \mathcal{S}_2 \mathcal{S}_1 \mathcal{S}_2 \mathcal{C}_1 \mathcal{C}_2)^\omega$	1.119
3	0.002	Euler	$(\mathcal{S}_1 \mathcal{S}_2 \mathcal{S}_2 \mathcal{S}_2 \mathcal{C}_1 \mathcal{C}_2)^\omega$	0.7651
4	0.002	Euler	$(\mathcal{S}_1 \mathcal{S}_2 \mathcal{S}_1 \mathcal{S}_1 \mathcal{C}_1 \mathcal{C}_2)^\omega$	∞
5	0.003	Euler	$(\mathcal{S}_1 \mathcal{S}_2 \mathcal{S}_1 \mathcal{S}_2 \mathcal{S}_1 \mathcal{S}_2 \mathcal{S}_2 \mathcal{S}_2 \mathcal{C}_1 \mathcal{C}_2 \mathcal{B}_0 \mathcal{B}_0)^\omega$	6.865
6	0.003	Euler	$(\mathcal{S}_1 \mathcal{S}_2 \mathcal{S}_1 \mathcal{S}_2 \mathcal{S}_1 \mathcal{S}_2 \mathcal{S}_1 \mathcal{S}_2 \mathcal{S}_2 \mathcal{C}_1 \mathcal{C}_2 \mathcal{B}_0 \mathcal{B}_0)^\omega$	∞
7	0.003	Euler	$(\mathcal{S}_1 \mathcal{S}_2 \mathcal{S}_1 \mathcal{S}_2 \mathcal{S}_1 \mathcal{S}_2 \mathcal{S}_1 \mathcal{S}_2 \mathcal{S}_1 \mathcal{C}_1 \mathcal{C}_2 \mathcal{B}_0 \mathcal{B}_0)^\omega$	∞
8	0.003	Euler	$(\mathcal{S}_1 \mathcal{S}_2 \mathcal{S}_1 \mathcal{S}_2 \mathcal{S}_1 \mathcal{S}_2 \mathcal{S}_1 \mathcal{S}_1 \mathcal{S}_1 \mathcal{C}_1 \mathcal{C}_2 \mathcal{B}_0 \mathcal{B}_0)^\omega$	∞

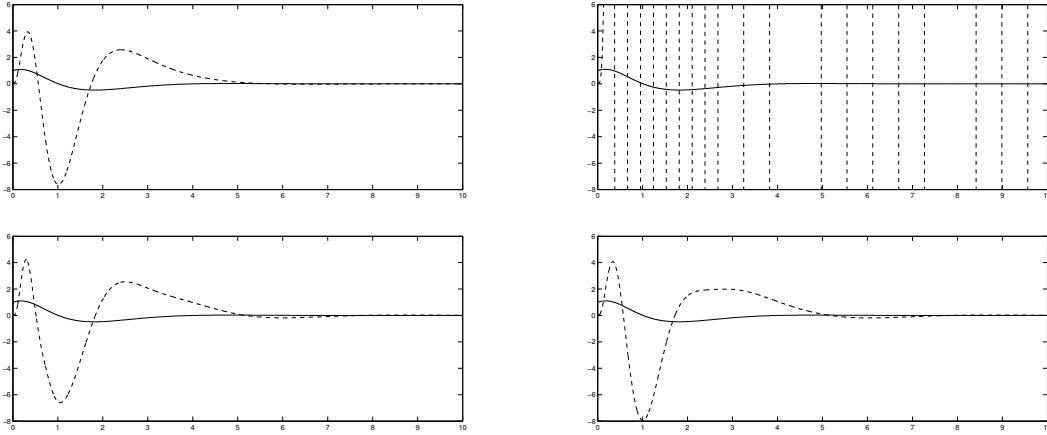


Fig. 5: Plots of outputs y_1 (solid lines) and y_2 (dashed lines) of example 2 for the ideal semantics $\llbracket \mathcal{M} \rrbracket$ (top left), and implementations $\llbracket \mathcal{M} \rrbracket_{(\rho_1, \tau_1, \delta_1)}$ (top right), $\llbracket \mathcal{M} \rrbracket_{(\rho_2, \tau_2, \delta_2)}$ (bottom left), $\llbracket \mathcal{M} \rrbracket_{(\rho_3, \tau_3, \delta_3)}$ (bottom right).

gives better performance, which is expected since the second subsystem has faster dynamics than the first one. Devoting most computation time to a wrong block may make the system unstable, as shown in implementation $(\rho_4, \tau_4, \delta_4)$. Therefore, scheduling can have great impact on the overall performance of the system. This fact is illustrated again in implementations 5 through 8, where $(\rho_5, \tau_5, \delta_5)$ which focuses on observer \mathcal{S}_2 stabilizes the system while other schedules destabilize it. Figure 5 shows the outputs for various implementation choices.

7. CONCLUSIONS

In this paper, we continue our efforts aimed at understanding and quantifying the gap between model-level timed semantics of embedded controllers and their implementation on time-triggered platforms. For linear plant models and linear dynamic controllers (e.g. PID controllers), we have presented a method to exactly compute the L_2 -error of the deviation of the plant's output in the implementation semantics from that in the continuous time semantics. This method gives a criterion to measure the quality of implementations and to compare different implementations. It can also be used in designing schedules for a time-triggered platform.

Future research includes the extension of our framework to larger classes of plant models, including nonlinear and hybrid systems, as well as more general nonlinear controllers. Whereas exact computation of the error may not be feasible in these gener-

eral settings, computable error bounds for appropriate norms will still enable the comparison of different implementations. 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. Finally, more efficient algorithms for designing schedules is also a subject of future research.

APPENDIX

PROOF OF THEOREM 3.1. First, note 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 $[t_i, t_{i+1})$, using equations (14), (15) and notice that $\dot{y}(t) = C_P \dot{x}(t)$, the evolution of the closed loop system for the model-level, continuous time semantics can be described by the differential equations:

$$\begin{bmatrix} \dot{x}(t) \\ \dot{z}(t) \end{bmatrix} = \begin{bmatrix} M(A_p + B_p(I - L_c)^{-1} K_P C_p) & M B_p(I - L_c)^{-1} K_I \\ B_c C_p & A_c \end{bmatrix} \begin{bmatrix} x(t) \\ z(t) \end{bmatrix} = \hat{A} \begin{bmatrix} x(t) \\ z(t) \end{bmatrix} \quad (27)$$

where $M = (I - B_p(I - L_c)^{-1} K_D C_p)^{-1}$. On the same interval $[t_i, t_{i+1})$, the computed control $\tilde{u}(t) = \tilde{u}(t_i)$ stays constant until the next instant t_{i+1} , therefore the evolution of the implementation semantics is given by

$$\dot{\tilde{x}}(t) = A_p \tilde{x}(t) + B_p \tilde{u}(t_i). \quad (28)$$

We now define the following linear system with output $y_e(t) = y(t) - \tilde{y}(t)$

$$\begin{cases} \dot{\varphi}(t) = A\varphi(t) + B\tilde{u}(t_i), \\ y_e(t) = C\varphi(t) \end{cases} \quad (29)$$

where

$$A = \begin{bmatrix} \hat{A} & 0 \\ 0 & A_p \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ B_p \end{bmatrix}, \quad C = [C_p \quad 0 \quad -C_p], \quad \varphi(t) = \begin{bmatrix} x(t) \\ z(t) \\ \tilde{x}(t) \end{bmatrix}$$

By explicitly solving the differential equation (29), we obtain

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

where $\alpha(t) = \int_0^t e^{A\tau} B d\tau$. On the interval $[t_i, t_{i+1})$, we can check that

$$\begin{aligned} \mathcal{I}_i &= \int_{t_i}^{t_{i+1}} \|(y(t) - \tilde{y}(t))\|_2^2 dt = \int_{t_i}^{t_{i+1}} y_e(t)^T y_e(t) dt \\ &= \varphi(t_i)^T \left(\int_0^\delta e^{A^T t} C^T C e^{At} dt \right) \varphi(t_i) + \varphi(t_i)^T \left(\int_0^\delta e^{A^T t} C^T C \alpha(t) dt \right) \tilde{u}(t_i) \\ &\quad + \tilde{u}(t_i)^T \left(\int_0^\delta \alpha(t)^T C^T C e^{At} dt \right) \varphi(t_i) + \tilde{u}(t_i)^T \left(\int_0^\delta \alpha(t)^T C^T C \alpha(t) dt \right) \tilde{u}(t_i) \end{aligned}$$

Let us define the following matrices

$$\begin{aligned} Q_{1,1} &= \int_0^\delta e^{A^T t} C^T C e^{At} dt \\ Q_{1,2} &= Q_{2,1}^T = \int_0^\delta e^{A^T t} C^T C \alpha(t) dt \\ Q_{2,2} &= \int_0^\delta \alpha(t)^T C^T C \alpha(t) dt \end{aligned}$$

Then, we have

$$\mathcal{I}_i = \begin{bmatrix} \varphi(t_i) \\ \tilde{u}(t_i) \end{bmatrix}^T \begin{bmatrix} Q_{1,1} & Q_{1,2} \\ Q_{2,1} & Q_{2,2} \end{bmatrix} \begin{bmatrix} \varphi(t_i) \\ \tilde{u}(t_i) \end{bmatrix} = \psi(t_i)^T Q \psi(t_i) \quad (30)$$

Note that matrix Q is constant and computable. In case matrix A is invertible, it is straightforward to see that $\alpha(t) = (e^{At} - I) A^{-1} B$.

The desired result is directly obtained from (30). \square

PROOF OF THEOREM 3.2. Define the matrices

$$\hat{E}_0 = \bar{E}(n_\rho - 1) \bar{E}(n_\rho - 2) \dots \bar{E}(1) \bar{E}(0),$$

$$\hat{G}_0 = \begin{bmatrix} I \\ \bar{E}(0) \\ \bar{E}(1) \bar{E}(0) \\ \vdots \\ \bar{E}(n_\rho - 2) \dots \bar{E}(0) \end{bmatrix},$$

and

$$\hat{E} = \bar{E}(2n_\rho - 1) \bar{E}(2n_\rho - 2) \dots \bar{E}(n_\rho + 1) \bar{E}(n_\rho),$$

$$\hat{G} = \begin{bmatrix} I \\ \bar{E}(n_\rho) \\ \bar{E}(n_\rho + 1) \bar{E}(n_\rho) \\ \vdots \\ \bar{E}(2n_\rho - 2) \dots \bar{E}(n_\rho) \end{bmatrix}$$

Using Eq. (22), we have that,

$$\begin{cases} \bar{\psi}(t_{n_\rho}) &= \hat{E}_0 \bar{\psi}(0) \\ \begin{bmatrix} \bar{\psi}(0) \\ \vdots \\ \bar{\psi}(t_{n_\rho-1}) \end{bmatrix} &= \hat{G}_0 \bar{\psi}(0) \end{cases} \quad (31)$$

and for all $l \geq 1$,

$$\begin{cases} \bar{\psi}(t_{(l+1)n_\rho}) &= \hat{E} \bar{\psi}(t_{ln_\rho}) \\ \begin{bmatrix} \bar{\psi}(t_{ln_\rho}) \\ \vdots \\ \bar{\psi}(t_{ln_\rho+n_\rho-1}) \end{bmatrix} &= \hat{G} \bar{\psi}(t_{ln_\rho}) \end{cases} \quad (32)$$

Note that we can write vector $\psi(t)$, defined in Theorem 3.1, as $\psi(t) = F \bar{\psi}(t)$ where

$$F = \begin{bmatrix} I & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & I & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & I & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & I \end{bmatrix}.$$

Theorem 3.1 yields

$$\begin{aligned}
e_{\mathcal{M}}(\rho, \tau, \delta, x(0)) &= \sum_{l=0}^{l=+\infty} \sum_{i=l n_{\rho}}^{i=l n_{\rho} + n_{\rho} - 1} \psi(t_i)^T Q \psi(t_i) \\
&= \sum_{l=0}^{l=+\infty} \sum_{i=l n_{\rho}}^{i=l n_{\rho} + n_{\rho} - 1} \bar{\psi}(t_i)^T F^T Q F \bar{\psi}(t_i) \\
&= \sum_{l=0}^{l=+\infty} \left(\begin{bmatrix} \bar{\psi}(t_{l n_{\rho}}) \\ \vdots \\ \bar{\psi}(t_{l n_{\rho} + n_{\rho} - 1}) \end{bmatrix}^T \begin{bmatrix} F^T Q F & & \\ & \ddots & \\ & & F^T Q F \end{bmatrix} \begin{bmatrix} \bar{\psi}(t_{l n_{\rho}}) \\ \vdots \\ \bar{\psi}(t_{l n_{\rho} + n_{\rho} - 1}) \end{bmatrix} \right) \quad (33)
\end{aligned}$$

Define \hat{Q} to be a block diagonal matrix composed of n_{ρ} blocks equal to $F^T Q F$:

$$\hat{Q} = \begin{bmatrix} F^T Q F & & \\ & \ddots & \\ & & F^T Q F \end{bmatrix}.$$

From Eq. (31), Eq. (32) and Eq. (33) we have:

$$\begin{aligned}
e_{\mathcal{M}}(\rho, \tau, \delta, x(0)) &= \bar{\psi}(0)^T \hat{G}_0^T \hat{Q} \hat{G}_0 \bar{\psi}(0) + \sum_{l=1}^{l=+\infty} \bar{\psi}(t_{l n_{\rho}})^T \hat{G}^T \hat{Q} \hat{G} \bar{\psi}(t_{l n_{\rho}}) \\
&= \bar{\psi}(0)^T \hat{G}_0^T \hat{Q} \hat{G}_0 \bar{\psi}(0) + \bar{\psi}(0)^T \hat{E}_0^T \left(\sum_{l=0}^{l=+\infty} (\hat{E}^l)^T \hat{G}^T \hat{Q} \hat{G} \hat{E}^l \right) \hat{E}_0 \bar{\psi}(0) \\
&= \bar{\psi}(0)^T \left[\hat{G}_0^T \hat{Q} \hat{G}_0 + \hat{E}_0^T \mathcal{O} \hat{E}_0 \right] \bar{\psi}(0) \\
&= \bar{\psi}(0)^T \hat{\mathcal{O}} \bar{\psi}(0)
\end{aligned}$$

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 [Rugh 1996], \mathcal{O} is the solution of the Lyapunov equation (24). Moreover, from the continuous time and the implementation semantics, we have $\bar{\psi}(0) = Hx(0)$. \square

REFERENCES

- R. Alur and A. Chandrashekarapuram. 2005. Dispatch sequences for embedded control models. In *Proceedings of the 11th IEEE Real-time and Embedded Technology and Applications*. 508–518.
- R. Alur, F. Ivancic, J. Kim, I. Lee, and O. Sokolsky. 2003. Generating embedded software from hierarchical hybrid models. In *Proceedings of the ACM Conference on Languages, Compilers, and Tools for Embedded Systems*. 171–182.
- Rajeev Alur and Gera Weiss. 2008. Regular Specifications of Resource Requirements for Embedded Control Software. In *RTAS '08: Proceedings of the 2008 IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE Computer Society, Washington, DC, USA, 159–168. DOI: <http://dx.doi.org/10.1109/RTAS.2008.13>
- P. Antsaklis and A. Michel. 1997. *Linear Systems*. McGraw Hill.
- Karl-Erik Årzén, Anton Cervin, and Dan Henriksson. 2005. Implementation-Aware Embedded Control Systems. In *Handbook of Networked and Embedded Control Systems*. Birkhäuser.

- K.J. Aström and B. Wittenmark. 1997. *Computer-controlled systems: Theory and Design*. Prentice Hall.
- Stephen Boyd and Lieven Vandenbergh. 2004. *Convex Optimization*. Cambridge University Press.
- G.C. Buttazo. 1997. *Hard real-time computing systems: Predictable scheduling algorithms and applications*. Kluwer Academic Publishers.
- P. Caspi, A. Curic, A. Maignan, C. Sofronis, and S. Tripakis. 2003. Translating discrete-time Simulink to Lustre. In *Proceedings of Third International Conference on Embedded Software (LNCS 2855)*. 84–99.
- Paul Caspi and Oded Maler. 2005. From Control Loops to Real-Time Programs. In *Handbook of Networked and Embedded Control Systems*. Birkhäuser.
- N. Halbwachs. 1993. *Synchronous Programming of Reactive Systems*. Kluwer Academic Publishers.
- N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. 1991. The synchronous dataflow programming language Lustre. *Proc. IEEE* 79 (1991), 1305–1320.
- Dan Henriksson, Anton Cervin, and Karl-Erik Årzén. 2003. TrueTime: Real-time Control System Simulation with MATLAB/Simulink. In *Proceedings of the Nordic MATLAB Conference*. Copenhagen, Denmark.
- T.A. Henzinger, B. Horowitz, and C.M. Kirsch. 2003. Giotto: A time-triggered language for embedded programming. *Proc. IEEE* 91, 1 (2003), 84–99.
- Y. Hur, J. Kim, I. Lee, and J. Choi. 2004. Sound code generation from communicating hybrid models. In *Hybrid Systems: Computation and Control, Proceedings of the 7th International Workshop (LNCS 2993)*. 432–447.
- H. Kopetz. 2000. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers.
- H. Kopetz and G. Bauer. 2003. The time triggered architecture. *Proc. IEEE* 91, 1 (2003), 112–126.
- E.A. Lee. 2000. What’s ahead for embedded software. *IEEE Computer* (September 2000), 18–26.
- Cleve Moler and Charles Van Loan. 2003. Nineteen Dubious Ways to Compute the Exponential of a Matrix, Twenty-Five Years Later. *SIAM Rev.* 45, 1 (2003), 3–49. DOI: <http://dx.doi.org/10.1137/S00361445024180>
- Truong Nghiem, George J. Pappas, Rajeev Alur, and Antoine Girard. 2006. Time-triggered implementations of dynamic controllers. In *EMSOFT ’06: Proceedings of the 6th ACM & IEEE International conference on Embedded software*. ACM, New York, NY, USA, 2–11. DOI: <http://dx.doi.org/10.1145/1176887.1176890>
- Wilson J. Rugh. 1996. *Linear System Theory*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.
- S. Sastry, J. Sztipanovits, R. Bajcsy, and H. Gill. 2003. Modeling and design of embedded software. *Proc. IEEE* 91, 1 (2003).
- S.Bittanti and P.Colaneri. 2000. Invariant representations of discrete-time periodic systems - a survey. *Automatica* 36, 12 (2000), 1777–1793.
- D. Seto, J. Lehoczky, L. Sha, and K. Shin. 1996. On task schedulability in real-time control systems. In *Proceedings of the IEEE Real-Time Systems Symposium*.
- C. Van Loan. 1978. Computing integrals involving the matrix exponential. *Automatic Control, IEEE Transactions on* 23, 3 (1978), 395–404. 0018-9286.
- M. De Wulf, L. Doyen, and J.-F. Raskin. 2004. Almost ASAP semantics: From timed models to timed implementations. In *Hybrid Systems: Computation and Control, Proceedings of the 7th International Workshop (LNCS 2993)*. 296–310.
- H. Yazarel, A. Girard, G. J. Pappas, and R. Alur. 2005. Quantifying the Gap between Embedded Control Models and Time-Triggered Implementations.. In *Proceedings of the 26th IEEE Real-Time Systems Symposium (RTSS)*. 111–120.