Time-triggered Implementations of Dynamic Controllers*

Truong Nghiem
Dept. Elec. Sys. Eng.
University of Pennsylvania
Philadelphia, USA
nghiem@seas.upenn.edu

George J. Pappas
Dept. Elec. Sys. Eng.
University of Pennsylvania
Philadelphia, USA
pappasg@seas.upenn.edu

Antoine Girard
VERIMAG
Grenoble, France
Antoine.Girard@imag.fr

Rajeev Alur
Dept. Comp. Info. Sci.
University of Pennsylvania
Philadelphia, USA
alur@cis.upenn.edu

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 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*

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

EMSOFT'06, October 22–25, 2006, Seoul, Korea.

General Terms

Design, Performance

Keywords

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

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 [15, 14]. The goal of our research, initiated in a recent paper [19], is to address this challenge in the context of implementing feedback control loops by software [7].

Consider a physical plant interacting with a 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 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. [5, 12]).

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 [13, 12]. 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, proportional-integral-derivative (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 [19], in this paper we consider more general dynamic controllers (PID). This in turn requires us to consider the error dynamics of the numerical integration and differentiation algorithms in our error computation. Using the lifting technique for transforming a PLTV to a linear time-invariant linear system, we can compute the error with respect to the model semantics measured by L_2-norms. Since different implementations correspond to different PLTVs, this gives us a way of comparing implementations quantitatively, as illustrated via examples in Section 4.

Related Work : Programming abstractions for embedded real-time controllers include synchronous reactive programming (languages such as ESTEREL and LUSTRE [9, 8]), and the related *Fixed Logical Execution Time* (FLET) assumption used in the Giotto project [10]. Research on time-triggered architecture has focused on achieving clock synchronization, fault tolerance, real-time communication, and automotive applications (c.f. [12, 13]). The goal of this research has been ensuring predictable communication between components. In the context of this paper, time-triggered platform offers pre-defined 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 [2, 11, 18], 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 [6]. 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. 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 [17] and control-scheduling co-design [3].

There is a rich literature on sampled control systems [4] 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 [4]. 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 $X = \{x_1, \ldots, x_n\}$ of *plant variables*, a set $Y = \{y_1, \ldots, y_p\}$ of *output variables*, a set $U = \{u_1, \ldots, u_m\}$ of *control variables*, and a set $Z = \{z_1, \ldots, z_q\}$ of *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 $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ that defines the dynamics of variables x_i in terms of the current plant state and control inputs.
- A function $h : \mathbb{R}^n \to \mathbb{R}^p$ that expresses the observable output of the plant given the current plant state.

A controller model $\mathcal{M}_C = (\mathcal{B}_I, \mathcal{B}_1, \ldots, \mathcal{B}_m)$, consists of

- A control block $\mathcal{B}_I$ that describes the function $g : \mathbb{R}^q \times \mathbb{R}^p \to \mathbb{R}^q$ that expresses the dynamics of the internal variables z_i in terms of the current internal variables and plant outputs.
- A finite set of control blocks $(\mathcal{B}_1, \ldots, \mathcal{B}_m)$, one control block $\mathcal{B}_j$ for every control variable $u_j \in U$. Every control block $\mathcal{B}_j$ describes a function
$$k_j : \mathbb{R}^p \times \mathbb{R}^p \times \mathbb{R}^q \times \mathbb{R}^{j-1} \to \mathbb{R}$$
expressing the feedback control law for the control variable u_j as a function of observable plant outputs and their derivatives, internal variables, and other control variables.

We assume that the dependence among the control variables is acyclic so that the feedback law for the control variable u_j is a function of the plant outputs and their derivatives, the internal states, and the control variables $u_1, \ldots, u_{j-1}$.

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. In addition, the derivatives of the plant outputs are included in the computation of control blocks $\mathcal{B}_j$. 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 [19].

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 $x(t) = (x_1(t), \ldots, x_n(t))$ denote plant trajectories in vector notation, and, similarly, $y(t) = (y_1(t), \ldots, y_p(t))$, $u(t) = (u_1(t), \ldots, u_m(t))$, and $z(t) = (z_1(t), \ldots, z_q(t))$. Given feedback control model $\mathcal{M}$, we denote the *continuous-time semantics* of the feedback control model by $[\![\mathcal{M}]\!]$ and define $[\![\mathcal{M}]\!]$ 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:

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

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

In this paper, we assume that the feedback composition 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 solution for the continuous-semantics as

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

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 ideal model-level semantics assumes that all control blocks of controller $\mathcal{M}_C = (\mathcal{B}_I, \mathcal{B}_1, \ldots, \mathcal{B}_m)$ are "computed" *instantaneously* and *simultaneously*. Of course, any software implementation of the controller $\mathcal{M}_C$ will violate both assumptions. In addition, the continuous-time semantics assumes that the internal variables, described by the differential equation $\dot{z}(t) = g(z(t), y(t))$ of control block $\mathcal{B}_I$, are ideally integrated, with no error. Clearly, any numerical method for approximately solving this differential equation will introduce errors depending on the integration step size. Moreover, ideal differentiation computation assumed by the model-level semantics cannot be satisfied by the implementation, where an approximation algorithm 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 $\{\mathcal{B}_0, \mathcal{B}_I, \mathcal{B}_1, \ldots, \mathcal{B}_m\}$. 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 $\mathcal{M}_C = (\mathcal{B}_I, \mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3)$ 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 $\mathcal{B}_i$ (or $\mathcal{B}_I$) consists of reading the relevant plant output variables using sensors, updating the control variable u_i (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* $\tau : \{\mathcal{B}_I, \mathcal{B}_1, \ldots, \mathcal{B}_m\} \to \mathbb{Z}^+$ 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 $[\![\mathcal{M}]\!]_{(\rho,\tau,\delta)}$, 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.

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 $\Delta_I(i)$, where $\Delta_I(0) = 0$ and

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

in order to model the time elapsed since the last execution of the integration block $\mathcal{B}_I$. Similarly, we define the sequence of time instants $\Delta_D(i)$, where $\Delta_D(0) = 0$ and $j = 1, 2, \ldots, m$,

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

to model the time elapses between executions of blocks $\mathcal{B}_j$, $1 \leq j \leq m$, which are essential to the differentiation computation. The derivatives of plant outputs y are numerically computed in the controller and are hold in internal variables w. The implementation semantics $[\![\mathcal{M}]\!]_{(\rho,\tau,\delta)}$ 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

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

Inter-sample

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

Controller Updates

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

Numerical Differentiation

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

Numerical Integration

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

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 the function $G(z(t_i), y(t_i), z(t_i - \Delta_I(i)), y(t_i - \Delta_I(i)), \Delta_I(i))$ 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 1. Euler is a one-step method, whereas 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 step[1]. 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, the function $D(y(t_i), y(t_i - \Delta_D(i)), \Delta_D(i), w(t_i))$ can be used to model the numerical computation of the derivatives of y. Some commonly used algorithms for approximating the derivatives are

[1]Note 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.

Backward Difference

$$w(t_{i+1}) = \frac{1}{\Delta_D(i)}\big(y(t_i) - y(t_i - \Delta_D(i))\big)$$

Tustin's Approximation

$$w(t_{i+1}) = \frac{2}{\Delta_D(i)}\big(y(t_i) - y(t_i - \Delta_D(i))\big) - w(t_i)$$

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

$$(x(t), y(t), u(t), z(t)) = [\![\mathcal{M}]\!]_{(\rho,\tau,\delta)}(x(0)) \tag{11}$$

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

$$\begin{aligned} (x(t), y(t), u(t), z(t)) &= [\![\mathcal{M}]\!](x(0)) \\ (\tilde{x}(t), \tilde{y}(t), \tilde{u}(t), \tilde{z}(t)) &= [\![\mathcal{M}]\!]_{(\rho,\tau,\delta)}(x(0)) \\ e_{\mathcal{M}}(\rho, \tau, \delta, x(0)) &= \int_0^{+\infty} \|y(t) - \tilde{y}(t)\|_2^2 dt \end{aligned} \tag{12}$$

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 $(\rho_2, \tau_2, \delta_2)$ (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:

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

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:

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

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

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 1: Function G for some well known numerical integration algorithms

following structure

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

where $B_C \in \mathbb{R}^{q \times p}$, $K_P \in \mathbb{R}^{m \times p}$, $K_I \in \mathbb{R}^{m \times q}$, and $L_C \in \mathbb{R}^{m \times m}$. The controller model $\mathcal{M}_C = (\mathcal{B}_I, \mathcal{B}_1, \ldots \mathcal{B}_m)$ consists of one control block $\mathcal{B}_I$ for integrating all the internal variables z_i, and one control block $\mathcal{B}_i$ for every variable u_i (i.e. u_1 is a linear combination of $y_1, \ldots, y_p$, $z_1, \ldots, z_q$, $dy_1/dt, \ldots, dy_p/dt$, and u_j is a linear combination of $y_1, \ldots, y_p$, $z_1, \ldots, z_q$, $dy_1/dt, \ldots, dy_p/dt$ and $u_1, \ldots, u_{j-1}$). Note that the assumption that the dependence between control variables are acyclic implies that L_C is a lower triangular matrix. Since the internal variables z_i simply integrate the output variables y_i, we obtain that

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

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 the execution of all the control blocks require the same time. Thus, $\tau(\mathcal{B}_I) = 1$, and for all $1 \leq j \leq m$, $\tau(\mathcal{B}_j) = 1$, and for all $i \geq 0$, $t_i = i\delta$. The challenge 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 1. *There exists a computable, symmetric, positive semi-definite matrix Q such that the implementation error defined by equation* (12) *is equal to*

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

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

Theorem 1 has effectively replaced a continuous integration with an infinite sum. The computation of matrix Q is described in the proof in the Appendix. Given Q, what we need for all t_i are the values of $x(t_i)$, $z(t_i)$ from the model-level semantics, and the values of $\tilde{x}(t_i)$ and $\tilde{u}(t_i)$ from the implementation level semantics. We will now define two discrete-time dynamical systems, that will generate these desired sequences.

Model-Level Semantics

Using equations (14), (15) and notice that $\dot{y}(t) = C_P \dot{x}(t)$, the evolution of the closed loop feedback system for the model-level semantics can be described as :

$$\begin{bmatrix} \dot{x}(t) \\ \dot{z}(t) \end{bmatrix} = \hat{A} \begin{bmatrix} x(t) \\ z(t) \end{bmatrix} = \begin{bmatrix} M\left(A_p + B_p(I - L_c)^{-1} K_P C_p\right) & M B_p (I - L_c)^{-1} K_I \\ B_c C_p & 0 \end{bmatrix} \begin{bmatrix} x(t) \\ z(t) \end{bmatrix} \tag{17}$$

where $M = \left(I - B_p(I - L_c)^{-1} K_D C_p\right)^{-1}$. 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:

$$\begin{bmatrix} x(t_{i+1}) \\ z(t_{i+1}) \end{bmatrix} = E \begin{bmatrix} x(t_i) \\ z(t_i) \end{bmatrix} \qquad x(t_0) = x(0) \quad z(t_0) = 0 \tag{18}$$

where $E = e^{\delta \hat{A}}$ is simply the matrix exponential of $\delta \hat{A}$. Thanks to Theorem 1, the model level semantics that are relevant for the computation of the implementation error are all captured in model (18).

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.

Differentiation Computation

Since each control variable u_j depends on the derivatives $\dot{y}(t)$ of plant outputs, we assume that in the execution of each control block $\mathcal{B}_j$, the derivatives $\dot{\tilde{y}}(t_i) = \tilde{w}(t_{i+1})$ are numerically computed and are then used in updating control variable u_j. Because in general, the numerical differentiation algorithm requires the old values $\tilde{y}(t_i - \Delta_D(t_i))$, we extend the system by adding the memory variables $\tilde{y}_m(t_i) = \tilde{y}(t_i - \Delta_D(t_i))$ which capture the needed information for the differentiation computation at t_i. Consider the computation

of control block $\rho(i)$ in time interval $[t_i, t_{i+1}]$. If $\rho(i) = \mathcal{B}_0$ or $\rho(i) = \mathcal{B}_I$, the values of variables $\tilde{w}$ and $\tilde{y}_m$ are not changed, that is $\tilde{w}(t_{i+1}) = \tilde{w}(t_i)$ and $\tilde{y}_m(t_{i+1}) = \tilde{y}_m(t_i)$. On the other hand, in the execution of control block $\mathcal{B}_j$, $1 \leq j \leq m$, the values of $\tilde{y}(t_i)$ are saved to the memory variables $\tilde{y}_m$, that is $\tilde{y}_m(t_{i+1}) = \tilde{y}(t_i) = C_P\tilde{x}(t_i)$, and the variables $\tilde{w}$ are updated according to the chosen algorithm for differentiation computation:

Backward Difference

$$\tilde{w}(t_{i+1}) = \tfrac{1}{\Delta_D(i)}\left(C_P\tilde{x}(t_i) - \tilde{y}_m(t_i)\right) \tag{19}$$

Tustin's Approximation

$$\tilde{w}(t_{i+1}) = \tfrac{2}{\Delta_D(i)}\left(C_P\tilde{x}(t_i) - \tilde{y}_m(t_i)\right) - \tilde{w}(t_i) \tag{20}$$

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 equation (34). By integration we thus obtain that

$$\tilde{x}(t_{i+1}) = e^{\delta A_p}\tilde{x}(t_i) + \alpha_p(\delta)\tilde{u}(t_i) \tag{21}$$

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. Since $\mathcal{B}_I$ is the integration block, the evolution of the internal variables z_i is captured by the equations describing the numerical algorithm that integrates the equations (15). Since the Trapezoid and Adams-Bashforth methods are two-step methods that depend on $\tilde{y}(t_i - \Delta_I(i))$, we can extend the system so that all the relevant information is available at t_i by defining the memory variables $\tilde{z}_m(t_i) = \tilde{y}(t_i - \Delta_I(i))$. Recalling that $\tilde{y}(t_i) = C_p\tilde{x}(t_i)$, we obtain following discrete-time models for the numerical integrators above

Euler

$$\tilde{z}(t_{i+1}) = \tilde{z}(t_i) + \Delta_I(i)B_cC_p\tilde{x}(t_i) \tag{22}$$

Trapezoid

$$\tilde{z}(t_{i+1}) = \tilde{z}(t_i) + \tfrac{\Delta_I(i)}{2}B_c(C_p\tilde{x}(t_i) + \tilde{z}_m(t_i)) \tag{23}$$
$$\tilde{z}_m(t_{i+1}) = C_p\tilde{x}(t_i) \tag{24}$$

Adams-Bashforth

$$\tilde{z}(t_{i+1}) = \tilde{z}(t_i) + \tfrac{\Delta_I(i)}{2}B_c(C_p3\tilde{x}(t_i) - \tilde{z}_m(t_i)) \tag{25}$$
$$\tilde{z}_m(t_{i+1}) = C_p\tilde{x}(t_i) \tag{26}$$

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

Let us define the vector

$$\tilde{\vartheta}(t) = \begin{bmatrix}\tilde{x}(t) & \tilde{z}(t) & \tilde{z}_m(t) & \tilde{w}(t) & \tilde{y}_m(t) & \tilde{u}(t)\end{bmatrix}^T$$

which consists the variables of the implementation semantics. 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

$$\tilde{\vartheta}(t_{i+1}) = E_{\mathcal{B}_I}(i)\tilde{\vartheta}(t_i) = \begin{bmatrix} e^{\delta A_p} & 0 & 0 & 0\ 0 & \alpha_p(\delta) \\ \frac{3\Delta_I(i)}{2}B_cC_p & I & -\frac{\Delta_I(i)}{2}B_c & 0\ 0 & 0 \\ C_p & 0 & 0 & 0\ 0 & 0 \\ 0 & 0 & 0 & I\ 0 & 0 \\ 0 & 0 & 0 & 0\ I & 0 \\ 0 & 0 & 0 & 0\ 0 & I \end{bmatrix}\tilde{\vartheta}(t_i) \tag{27}$$

Note that matrix $E_{\mathcal{B}_I}(i)$ 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 (21). 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

$$\tilde{\vartheta}(t_{i+1}) = E_{\mathcal{B}_0}\tilde{\vartheta}(t_i) = \begin{bmatrix} e^{\delta A_p} & 0\ 0\ 0\ 0 & \alpha_p(\delta) \\ 0 & I\ 0\ 0\ 0 & 0 \\ 0 & 0\ I\ 0\ 0 & 0 \\ 0 & 0\ 0\ I\ 0 & 0 \\ 0 & 0\ 0\ 0\ I & 0 \\ 0 & 0\ 0\ 0\ 0 & I \end{bmatrix}\tilde{\vartheta}(t_i) \tag{28}$$

Note that unlike matrix $E_{\mathcal{B}_I}$, matrix $E_{\mathcal{B}_0}$ is fixed.

Modeling control blocks $\mathcal{B}_j$

Let us now consider the control block $\mathcal{B}_j$, $1 \leq j \leq m$, executed during the time interval $[t_i, t_{i+1}]$. Clearly $\tilde{z}(t_{i+1}) = \tilde{z}(t_i)$, and $\tilde{z}_m(t_{i+1}) = \tilde{z}_m(t_i)$ in case a higher order method is used. For $k \neq j$, the execution of $\mathcal{B}_j$ does not modify the value of the variable $\tilde{u}_k$. Thus, the value of the variable $\tilde{u}_j$ is updated according to

$$\begin{aligned}\tilde{u}_k(t_{i+1}) &= \tilde{u}_k(t_i),\ \text{if } k \neq j \\ \tilde{u}_j(t_{i+1}) &= [K_P]_jC_p\tilde{x}(t_i) + [K_I]_j\tilde{z}(t_i) + [K_D]_j\tilde{w}(t_{i+1}) \\ &\quad + [L_C]_j\tilde{u}(t_i)\end{aligned}$$

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 $\tilde{w}(t_{i+1})$ which is computed from current values at instant t_i by some formula determined by the differentiation algorithm. For example, for the Tustin's Approximation method (Eq. (20)), $\tilde{u}_j(t_{i+1})$ can be written as

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

Let $U_j(i)$ be the matrix whose rows $[U_j(i)]_k$ are 0 if $k \neq j$ and $[U_j(i)]_j$ is the coefficient of $\tilde{x}(t_i)$ in the equation of $\tilde{u}_j(t_{i+1})$. Note that $U_j(i)$ depends on $\Delta_D(i)$ and the differentiation method used. Similarly let V_j be the constant matrix whose rows $[V_j]_k$ are 0 if $k \neq j$ and $[V_j]_j = [K_I]_j$, matrix $W_j(i)$ be the matrix whose rows $[W_j(i)]_k$ are 0 if $k \neq j$ and $[W_j(i)]_j$ is the coefficient of $\tilde{w}(t_i)$ in the equation of $\tilde{u}_j(t_{i+1})$, $X_j(i)$ be the matrix whose rows $[X_j(i)]_k$ are 0 if $k \neq j$ and $[X_j(i)]_j$ is the coefficient of $\tilde{y}_m(t_i)$, and Y_j be the matrix such that

$[Y_j]_k$ are 0 if $k \neq j$ and $[Y_j]_j = [L_C]_j$. Also, let Z_j be the matrix whose coefficients are all zero except the j^{th} element of its diagonal $[Z_j]_{j,j} = -1$. Then,

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

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

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

for the Tustin's Approximation method. Note that for $i = 0$, $\Delta_D(0) = 0$ and $\frac{1}{\Delta_D(0)}$ is replaced by $\frac{1}{\Delta_D(0)} = 0$. Also observe that like matrix $E_{\mathcal{B}_I}(i)$, matrix $E_{\mathcal{B}_j}(i)$ is not fixed but periodic after the first period of the dispatch sequence.

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:

$$\tilde{\vartheta}(t_{i+1}) = E(i)\tilde{\vartheta}(t_i) \tag{29}$$

where $E(i) = E_{\rho(i)}(i)$ when $\rho(i) \neq \mathcal{B}_I$, and $E(i) = E_{\mathcal{B}_I}(i)$ when $\rho(i) = \mathcal{B}_I$. 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.

3.1 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

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

Then,

$$\bar{\psi}(t_{i+1}) = \bar{E}(i)\ \bar{\psi}(t_i) = \begin{bmatrix} E & 0 \\ 0 & E(i) \end{bmatrix} \bar{\psi}(t_i) \tag{30}$$

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 [16]), 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 2. *The implementation error is exactly equal to*

$$e_{\mathcal{M}}(\rho, \tau, \delta, x(0)) = x(0)^T H^T \hat{\mathcal{O}} H x(0) \tag{31}$$

where $H = \begin{bmatrix} I & 0 & I & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T$, *and* $\hat{\mathcal{O}}$, $\mathcal{O}$ *are solutions to the following nested matrix Lyapunov equations*

$$\hat{\mathcal{O}} = \hat{G}_0^T \hat{Q} \hat{G}_0 + \hat{E}_0^T \mathcal{O} \hat{E}_0 \tag{32}$$

$$\mathcal{O} = \hat{E}^T \mathcal{O} \hat{E} + \hat{G}^T \hat{Q} \hat{G}. \tag{33}$$

for implementation-dependent matrices $\hat{E}_0$, $\hat{G}_0$, $\hat{E}$, $\hat{G}$, $\hat{Q}$ *(see proof in Appendix).*

Theorem 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. 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 E and $E(i)$ in (30), whereas solving Lyapunov equations is polynomial in the size of the matrices. The largest matrices in Theorem 2 are matrices $\hat{G}$ and $\hat{G}_0$ which have $n_\rho \times (2n + 5p + m)$ rows and $2n + 5p + m$ columns.

Theorem 2 provides a criterion for comparing two different time-triggered implementations of an embedded controller. Let $\mathcal{M}$ be a feedback control model, and let $(\rho_1, \tau_1, \delta_1)$ and $(\rho_2, \tau_2, \delta_2)$ be two implementations. Then, from Theorem 2, there exist two symmetric matrices $\mathcal{O}_1$ and $\mathcal{O}_2$ such that for all $x(0) \in \mathbb{R}^n$,

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

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 $x(0) \in X_0$ we have that $x(0)^T H^T \hat{\mathcal{O}}_1 H x(0) \leq x(0)^T H^T \hat{\mathcal{O}}_2 H x(0)$ or equivalently $0 \leq x(0)^T H^T (\hat{\mathcal{O}}_2 - \hat{\mathcal{O}}_1) H x(0)$. This is equivalent to checking whether

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

If X_0 is a convex polytope (or more general convex set), then this can be performed using convex programming. On the other hand, if $X_0 = \mathbb{R}^n$, then checking whether $0 \leq x(0)^T H^T (\hat{\mathcal{O}}_2 - \hat{\mathcal{O}}_1) H x(0)$ for all $x(0) \in \mathbb{R}^n$ reduces to $H^T (\hat{\mathcal{O}}_2 - \hat{\mathcal{O}}_1) H$ being a positive semi-definite matrix.

4. COMPUTATIONAL EXAMPLE

Our method has been implemented in MATLAB/SIMULINK to automate the computation of the implementation errors and to simulate the time-triggered implementation considered in this paper. Given $\mathcal{M} = \langle \mathcal{M}_P, \mathcal{M}_C \rangle$ and implementation specifics (ρ, τ, δ), for a particular initial state $x(0)$, we can simulate the SIMULINK model shown in Figure 1 in order to visualize the discrepancies between the output variables $y(t)$ and $\tilde{y}(t)$. We now illustrate the performance of various implementations for control system

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

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

Note that the plant consists of two separate subsystems, one subsystem (captured by variables x_1, x_2) being much faster than another (captured by variables x_3, x_4). The

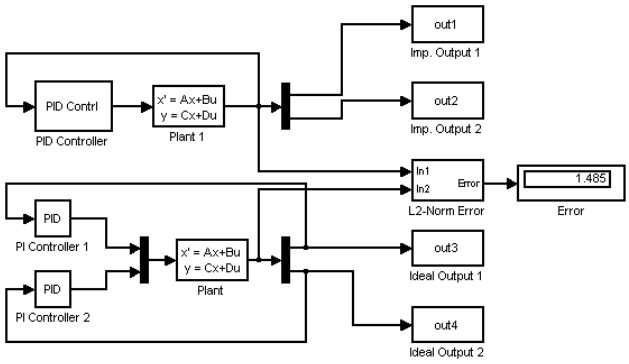

Figure 1: Simulink Model For Error Computation

PID controller that has been 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. For initial state $x_1(0) = x_2(0) = x_3(0) = x_4(0) = 2$, Table 2 summarizes the implementation errors for various dispatch sequences and different numerical integration and differentiation algorithms. For comparison, we include in Table 2 the corresponding results for a PI controller with the same parameters K_P and K_I. Figure 2 shows the evolution of output $y_1(t)$ for various implementation choices. From Table 2, it is observable that the implementation errors for the PI controller are generally less than the corresponding errors for the PID controller. This shows the effect of the derivative part on the performance of the software implementation. Though the ideal closed loop system is stable, implementation $(\rho_1, \tau_1, \delta_1)$ destabilizes the plant (for the PI controller), or produces a large error (for the PID controller). By using a better numerical integration algorithm, $(\rho_2, \tau_2, \delta_2)$ avoids instability and reduces the implementation error. It is worth noting that scheduling can have great effect on the overall performance of the system, as clearly illustrated when comparing implementations $(\rho_1, \tau_1, \delta_1)$, $(\rho_3, \tau_3, \delta_3)$, $(\rho_4, \tau_4, \delta_4)$, and $(\rho_5, \tau_5, \delta_5)$. With the same numerical integration and differentiation methods and the same time slot length δ, but a slight change in the dispatch sequence, implementation $(\rho_3, \tau_3, \delta_3)$ outperforms $(\rho_1, \tau_1, \delta_1)$. Moreover, for the PID controller case, it produces a smaller error than those of other implementations, even those on much faster platforms. Implementation $(\rho_4, \tau_4, \delta_4)$ destabilizes the plant with the PID controller while, by allocating more time slots to control block $\mathcal{B}_1$, implementation $(\rho_5, \tau_5, \delta_5)$ destabilizes the plant with the PI controller but greatly improves the performance when the PID controller is used. The above observations show that the performance of a controller can be considerably increased without changing the platform, however care should be taken when choosing the dispatch sequence. Furthermore, even on a slightly faster computer, the performance of $(\rho_7, \tau_7, \delta_7)$ may be worse than $(\rho_6, \tau_6, \delta_6)$ executing on a slower computer but with better dispatch sequence. Nonetheless for sufficiently faster platform $(\rho_8, \tau_8, \delta_8)$, the performance is improved greatly.

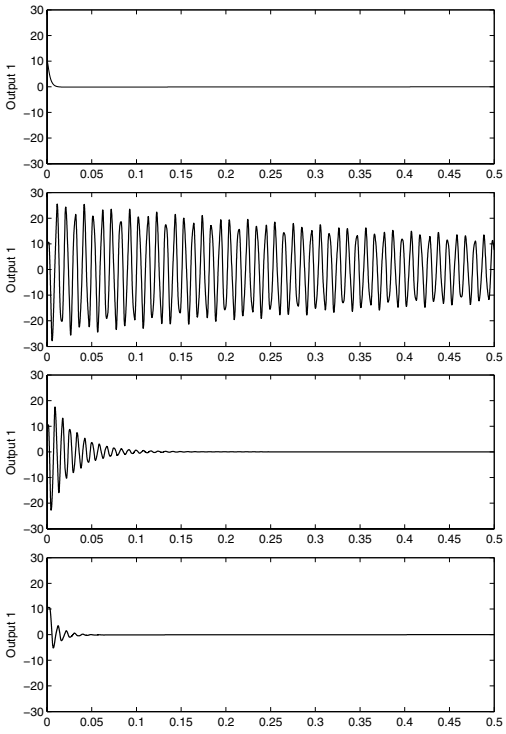

Figure 2: Plots of y_1 for the ideal semantics $[\![\mathcal{M}]\!]$ (top), and implementations $[\![\mathcal{M}]\!]_{(\rho_1,\tau_1,\delta_1)}$ (second to top), $[\![\mathcal{M}]\!]_{(\rho_7,\tau_7,\delta_7)}$ (second to bottom), $[\![\mathcal{M}]\!]_{(\rho_3,\tau_3,\delta_3)}$ (bottom).

5. 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 dynamic PID controllers, we have 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 gives a criterion to compare different implementations.

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 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. 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*

(ρ, τ, δ)	δ	Integration	Differentiation	ρ	$e_{\mathcal{M}}$ (PI)	$e_{\mathcal{M}}$ (PID)
$(\rho_1, \tau_1, \delta_1)$	0.001	Euler	Backward Diff.	$(\mathcal{B}_I\mathcal{B}_1\mathcal{B}_2)^\omega$	∞	10.0058
$(\rho_2, \tau_2, \delta_2)$	0.001	Trapezoid	Backward Diff.	$(\mathcal{B}_I\mathcal{B}_1\mathcal{B}_2)^\omega$	5.3074	7.6896
$(\rho_3, \tau_3, \delta_3)$	0.001	Euler	Backward Diff.	$(\mathcal{B}_I\mathcal{B}_2\mathcal{B}_1)^\omega$	8.8155	0.5241
$(\rho_4, \tau_4, \delta_4)$	0.001	Euler	Backward Diff.	$(\mathcal{B}_I\mathcal{B}_2\mathcal{B}_1\mathcal{B}_1)^\omega$	1.1461	∞
$(\rho_5, \tau_5, \delta_5)$	0.001	Euler	Backward Diff.	$(\mathcal{B}_I\mathcal{B}_2\mathcal{B}_1\mathcal{B}_1\mathcal{B}_1\mathcal{B}_1)^\omega$	∞	0.63357
$(\rho_6, \tau_6, \delta_6)$	0.001	Trapezoid	Backward Diff.	$(\mathcal{B}_I\mathcal{B}_1\mathcal{B}_2\mathcal{B}_1\mathcal{B}_1\mathcal{B}_1\mathcal{B}_1\mathcal{B}_1\mathcal{B}_1\mathcal{B}_1)^\omega$	0.61896	1.6412
$(\rho_7, \tau_7, \delta_7)$	0.00075	Trapezoid	Backward Diff.	$(\mathcal{B}_I\mathcal{B}_1\mathcal{B}_2)^\omega$	0.75237	1.9263
$(\rho_8, \tau_8, \delta_8)$	0.0005	Trapezoid	Backward Diff.	$(\mathcal{B}_I\mathcal{B}_1\mathcal{B}_2\mathcal{B}_1\mathcal{B}_1\mathcal{B}_1\mathcal{B}_1\mathcal{B}_1\mathcal{B}_1\mathcal{B}_1)^\omega$	0.19384	0.51015

Table 2: Implementation errors for various time-triggered platforms

on Languages, Compilers, and Tools for Embedded Systems, pages 171–182, 2003.

[3] K.-E. Årzén, A. Cervin, and D. Henriksson. Implementation-aware embedded control systems. In *Handbook of Networked and Embedded Control Systems*. Birkhäuser, Jan. 2005.

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

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

[6] 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.

[7] P. Caspi and O. Maler. From control loops to real-time programs. In *Handbook of Networked and Embedded Control Systems*. Birkhäuser, Jan. 2005.

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

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

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

[11] 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.

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

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

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

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

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

[17] 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.

[18] 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.

[19] H. Yazarel, A. Girard, G. J. Pappas, and R. Alur. Quantifying the gap between embedded control models and time-triggered implementations. In *Proceedings of the 26th IEEE Real-Time Systems Symposium (RTSS)*, pages 111–120, 2005.

APPENDIX

PROOF OF THEOREM 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 $[t_i, t_{i+1})$, the evolution of the closed loop system for the model-level, continuous time semantics can be described by the differential equations (17). For the implementation, on the same interval $[t_i, t_{i+1})$, the input $\tilde{u}(t) = \tilde{u}(t_i)$ since the computed control stays constant until the next computation, and therefore the evolution of the implementation semantics is given by

$$\dot{\tilde{x}}(t) = A_p \tilde{x}(t) + B_p \tilde{u}(t_i). \tag{34}$$

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

$$\begin{bmatrix} \dot{x}(t) \\ \dot{z}(t) \\ \dot{\tilde{x}}(t) \end{bmatrix} = \begin{bmatrix} \hat{A} & 0 \\ 0 & A_p \end{bmatrix} \begin{bmatrix} x(t) \\ z(t) \\ \tilde{x}(t) \end{bmatrix} + \begin{bmatrix} 0 \\ B_p \end{bmatrix} \tilde{u}(t_i)$$

$$y_e(t) = \begin{bmatrix} C_p & 0 & -C_p \end{bmatrix} \begin{bmatrix} x(t) \\ z(t) \\ \tilde{x}(t) \end{bmatrix}$$

Letting $\varphi(t) = \begin{bmatrix} x(t) \\ z(t) \\ \tilde{x}(t) \end{bmatrix}$, the above equation is a linear system with constant input $\tilde{u}(t_i)$, having the form

$$\left\{ \begin{array}{rcl} \dot{\varphi}(t) & = & A\varphi(t) + B\tilde{u}(t_i), \\ y_e(t) & = & C\varphi(t) \end{array} \right. \tag{35}$$

By explicitly solving the differential equation, we obtain

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

The integral term can be written as

$$\int_{t_i}^{t} e^{A(t-\tau)} B \, d\tau = \int_{0}^{t-t_i} e^{A\tau} B \, d\tau = \alpha(t - t_i)$$

where $\alpha(\cdot)$ is a matrix-valued function mapping $t \in \mathbb{R}$ to a matrix $\alpha(t) \in \mathbb{R}^{(2n+p)\times m}$. On the interval $[t_i, t_{i+1}]$, we can check that

$$\begin{aligned}\mathcal{I}_i &= \int_{t_i}^{t_{i+1}} \|(y(t) - \tilde{y}(t))\|_2^2 \, dt = \int_{t_i}^{t_{i+1}} y_e(t)^T y_e(t) \, dt \\ &= \varphi(t_i)^T \left(\int_0^\delta e^{A^T t} C^T C e^{At} \, dt \right) \varphi(t_i) \\ &\quad + \varphi(t_i)^T \left(\int_0^\delta e^{A^T t} C^T C \alpha(t) \, dt \right) \tilde{u}(t_i) \\ &\quad + \tilde{u}(t_i)^T \left(\int_0^\delta \alpha(t)^T C^T C e^{At} \, dt \right) \varphi(t_i) \\ &\quad + \tilde{u}(t_i)^T \left(\int_0^\delta \alpha(t)^T C^T C \alpha(t) \, dt \right) \tilde{u}(t_i)\end{aligned} \tag{36}$$

Let us define the following matrices

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

Then, we have

$$\mathcal{I}_i = \begin{bmatrix} x(t_i) \\ z(t_i) \\ \tilde{x}(t_i) \\ \tilde{u}(t_i) \end{bmatrix}^T \begin{bmatrix} Q_{1,1} & Q_{1,2} \\ Q_{2,1} & Q_{2,2} \end{bmatrix} \begin{bmatrix} x(t_i) \\ z(t_i) \\ \tilde{x}(t_i) \\ \tilde{u}(t_i) \end{bmatrix} = \psi(t_i)^T Q \psi(t_i) \tag{37}$$

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

The desired result is directly obtained from (37). □

Proof of Theorem 2. Consider the following matrices

$$\hat{E}_0 = \bar{E}(n_\rho - 1)\bar{E}(n_\rho - 2) \ldots \bar{E}(1)\bar{E}(0) \tag{38}$$

$$\hat{G}_0 = \begin{bmatrix} I \\ \bar{E}(0) \\ \bar{E}(1)\bar{E}(0) \\ \vdots \\ \bar{E}(n_\rho - 2) \ldots \bar{E}(0) \end{bmatrix} \tag{39}$$

$$\hat{E} = \bar{E}(2n_\rho - 1)\bar{E}(2n_\rho - 2) \ldots \bar{E}(n_\rho + 1)\bar{E}(n_\rho) \tag{40}$$

$$\hat{G} = \begin{bmatrix} I \\ \bar{E}(n_\rho) \\ \bar{E}(n_\rho + 1)\bar{E}(n_\rho) \\ \vdots \\ \bar{E}(2n_\rho - 2) \ldots \bar{E}(n_\rho) \end{bmatrix} \tag{41}$$

Then, we have,

$$\begin{cases} \bar{\psi}(t_{n_\rho}) &= \hat{E}_0 \, \bar{\psi}(0) \\ \begin{bmatrix} \bar{\psi}(0) \\ \vdots \\ \bar{\psi}(t_{n_\rho - 1}) \end{bmatrix} &= \hat{G}_0 \, \bar{\psi}(0) \end{cases} \tag{42}$$

and for all $l \geq 1$,

$$\begin{cases} \bar{\psi}(t_{(l+1)n_\rho}) &= \hat{E} \, \bar{\psi}(t_{ln_\rho}) \\ \begin{bmatrix} \bar{\psi}(t_{ln_\rho}) \\ \vdots \\ \bar{\psi}(t_{ln_\rho + n_\rho - 1}) \end{bmatrix} &= \hat{G} \, \bar{\psi}(t_{ln_\rho}) \end{cases} \tag{43}$$

Note that we can write vector $\psi(t)$, defined in Theorem 1, as $\psi(t) = F\bar{\psi}(t)$ where

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

We define the block diagonal matrix $\hat{Q}$ composed of n_ρ blocks equal to $F^T Q F$ where matrix Q is given by Theorem 1:

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

From Theorem 1, we have

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

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 $\mathcal{O}$ is the solution of the Lyapunov equation (33). Moreover, from the continuous time and the implementation semantics, we have $\bar{\psi}(0) = Hx(0)$. □