# On Self-Triggered Full-Information H-infinity Controllers

Michael Lemmon[1*], Thidapat Chantem[2], Xiaobo Sharon Hu[2], and Matthew Zyskowski[1]

[1] University of Notre Dame, Department of Electrical Engineering, Notre Dame, IN 46556, USA. lemmon,mzyskows@nd.edu,

[2] University of Notre Dame, Department of Computer Science and Engineering, Notre Dame, IN 46556, USA. Sharon.Hu.8,tchantem@nd.edu

**Abstract.** A self-triggered control task is one in which the task determines its next release time. It has been conjectured that self-triggering can relax the requirements on a real-time scheduler while maintaining application (i.e. control system) performance. This paper presents preliminary results supporting that conjecture for a self-triggered real-time system implementing full-information $\mathcal{H}_\infty$ controllers. Release times are selected to enforce upper bounds on the induced $\mathcal{L}_2$ gain of a linear feedback control system. These release times are treated as requests by the system scheduler, which then assigns actual release times using Buttazzo's elastic scheduling algorithm. Preliminary experimental results from a Matlab stateflow simulink model demonstrated a remarkable robustness to scheduling delays induced by real-time schedulers. These results show that self-triggered controllers are indeed able to maintain acceptable levels of application performance during prolonged periods of processor overloading.

## 1 Introduction

Computer controlled systems are often implemented using periodic real-time tasks. This approach can lead to significant over-provisioning of the real-time system since task periods are determined by the worst case time interval assuring closed loop system stability. In recent years, a number of researchers have proposed **aperiodic** task models in which tasks are either *event-triggered* [1] or *self-triggered* [2] controllers. Event-triggered control systems are systems whose control tasks are triggered by some asynchronous "event" within the control loop. These events are usually generated when an error signal crosses a specified threshold. The notion of event-triggered feedback [1] has appeared under a variety of names, such as interrupt-based feedback [3], Lebesgue sampling [4], asynchronous sampling [5], or state-triggered feedback [6].

Except for relay or pulse-width modulated feedback, event-triggered feedback can be impractical since it requires integrating an analog event detector into the

* The authors gratefully acknowledge the partial finanical support of the National Science Foundation (NSF-CNS-0410771)

physical plant. A more pragmatic approach for implementing aperiodic feedback is found in the **self-triggered** task model of Velasco et al. [2]. In self-triggered systems, the control task determines its next release time based on samples of the state gathered at the current release time. Self-triggered task models, therefore, can be implemented in existing computer controlled system without the need for any special analog event-detectors.

This paper presents experimental results examining the performance of a self-triggered control system. Our system's control tasks select sampling periods in a way that guarantees the closed-loop system's induced $\mathcal{L}_2$ gain satisfies a specified bound. We then consider a real-time system that schedules multiple self-triggered control tasks using traditional earliest-deadline-first (EDF) scheduling and Buttazzo's elastic scheduling algorithm [7]. Our implementation of Buttazzo's scheduler relies on a utilization constraint similar to that originally suggested by Chantem et al. [8]. Preliminary simulation results for a Simulink/Stateflow model of a real-time system controlling three inverted pendulums showed that the control system's performance under self-triggering was remarkably insensitive to the type of scheduler used by the real-time system. While preliminary, these results strongly suggest that self-triggering can provide a valuable way of ensuring control system performance in cases where the scheduler is unable to provide hard real-time guarantees on job completion.

## 2 Prior Work on Sample Period Selection

This section briefly reviews some of the prior work on sample period selection. Sample period selection for aperiodic real-time systems requires a detailed analysis of the system's intersample behavior. This usually involves studying a candidate Lyapunov function as was done in Zheng et al. [9] for a class of nonlinear sampled-data systems. Nesic et al. [10] used input-to-state stability (ISS) techniques to bound the intersample behavior of nonlinear systems [10]. This approach was used by Tabuada et al. [6] to estimate sampling periods for a class of nonlinear event-triggered control systems.

All of the aforecited works selected sampling periods to preserve some measure of the control system's stability, whether this is asymptotic stability or input-to-state stability. Applications, however, also need to ensure some minimum level of control system performance. Early work concerning the co-design of control systems and real-time systems viewed this as a schedulability problem in which sampling periods were selected to solve the following optimization problem,

$$\begin{array}{ll} \text{minimize:} & \text{Penalty on Control Performance} \\ \text{with respect to:} & \text{Sampling Periods} \\ \text{subject to:} & \text{closed loop stability} \\ & \text{task set schedulability} \end{array} \tag{1}$$

Early statements of this problem may be found in Seto et al. [11] with more recent studies in [12] and [13]. The penalty function used in the above problem

is often a performance index for an infinite-horizon optimal control. The problem we face here, however, is that such performance indices [5] are rarely monotone functions of the sampling period. So it can be very difficult to identify "optimal" sampling periods for the above problem.

This paper uses Lyapunov techniques to select sampling periods that bound the intersample behavior of the system. This is similar to the approaches in [10], [9], and [6]. But rather than simply assuring closed loop stability, we select sampling periods to adjust the induced $\mathcal{L}_2$ gain of the system. This approach allows us to make the system responsive to variations in the intensity of the input disturbances driving the system. In the presence of high-intensity disturbances, for example, the system can reduce its gain to keep its output signal below some specified threshold. In the presence of low-intensity disturbances, it can then relax that gain and still ensure that the output remains below the same specified threshold. The variations in disturbance intensity are therefore mirrored in variations of the system gain which in turn result in large variations in the sampling period. This means that during periods of low disturbance intensity, the average sampling period can be much longer than during periods of high disturbance intensity. The bounds on intersample behavior ensuring a specified $\mathcal{L}_2$ gain are discussed in section 4. Because our controllers enforce a specified induced $\mathcal{L}_2$ gain, we confine our attention to linear full-information $\mathcal{H}_\infty$ controllers for which we can generate tight bounds on the system's intersample behavior.

## 3 System Model

The real-time system considered in this paper consists of $N$ dynamical systems (called plants) that are controlled by $N$ tasks running on a single processor. Each task samples (S) the system state, computes a state feedback control, and outputs that control to the plant through a zero-order hold (H). The state $x_i : \Re \to \Re^n$, of the $i$th plant satisfies the initial value problem,

$$\begin{aligned} \dot{x}_i(t) &= A_i x_i(t) + B_{1i} u_i(t) + B_{2i} w_i(t) \\ x_i(0) &= x_{i0} \end{aligned} \tag{2}$$

for $t \geq 0$ and $i \in \mathcal{N} = \{1, \ldots, N\}$. The function $w_i : \Re \to \Re^n$ is an uncontrolled and bounded disturbance. The function $u_i : \Re \to \Re^m$ is the control input generated by the $i$th real-time control task. $A_i$, $B_{1i}$, and $B_{2i}$ are appropriately dimensioned matrices.

The $i$th plant's control, $u_i$, is generated by task $i \in \mathcal{N}$. Task $i$ is associated with a sequence of **release times**, $\{r_i[j]\}_{j=1}^{\infty}$. The time $r_i[j] \in \Re$ is that time when the $j$th **job** of task $i$ is available for execution. The task set is said to be **synchronous** if $r_i[0]$ is the same for all $i \in \mathcal{N}$. The **period** for the $j$th job of task $i$ is denoted as

$$T_i[j] = r_i[j+1] - r_i[j] \tag{3}$$

If $T_i[j]$ is constant for all $j = 1, \ldots, \infty$, then the task is said to be periodic. A task that is not periodic is said to be **sporadic**.

The task set is associated with a **scheduling function**, $\sigma : \Re \rightarrow \mathcal{N}$. This function takes the value $\sigma(t) = i \in \mathcal{N}$ at time $t$ when task $i$ is executing at that time. The **finishing time** for job $j$ of task $i$ is denoted as $f_i[j] \in \Re$ and is formally defined as

$$f_i[j] = \max \left\{ t \in \Re \ : \ \begin{array}{l} i = \sigma(t^+), i \neq \sigma(t^-) \\ r_i[j] \leq t \leq r_i[j+1] \end{array} \right\} \tag{4}$$

where $\sigma(t^+) = \lim_{\tau \uparrow t} \sigma(\tau)$ and $\sigma(t^-) = \lim_{\tau \downarrow t} \sigma(\tau)$ are the left and right hand limits of $\sigma$ at $t$, respectively.

The $i$th task's **worst-case execution time** (WCET) is denoted as $C_i \in \Re$. The task's **relative deadline** is denoted as $D_i \in \Re$. A task set is said to be **schedulable** if there exist sequences of release times $\{r_i[j]\}_{j=1}^{\infty}$ and a scheduling function $\sigma$ such that

$$D_i \geq f_i[j] - r_i[j] \geq C_i \tag{5}$$

for all $i \in \mathcal{N}$ and $j = 1, \ldots, \infty$.

The $i$th task computes the control $u_i$ for the $i$th plant. This control is assumed to be a state feedback control law of the form

$$u_i(t) = -k^T x(r_i[j]) \tag{6}$$

for $t \in [f_i[j], f_i[j+1])$ where $j = 1, \ldots, \infty$. Note that the control output is constant between finishing times and the value of that constant is determined by the system state at the job's release time, $r_i[j]$.

## 4 Sample Period Selection for Induced $\mathcal{L}_2$ Gain

This section states the paper's main result concerning sample period selection enforcing a specified bound on the closed-loop system's induced $\mathcal{L}_2$ gain. We confine our attention to the behavior of a single plant between consecutive release times. We therefore drop the task index, $i$, without a loss of generality.

We assume, for the purpose of analytic simplicity, that the control $u_i(t)$ satisfies equation 6 for all $t$ between consecutive *release times* rather than consecutive *finishing times*. This simplification is done for analytical convenience at the expense of some loss in generality.

The following theorem provides conditions which guarantee that the induced $\mathcal{L}_2$ gain from the plant's disturbance $w$ to its state be less than a specified positive number, $\gamma$.

**Theorem 1.** *Let $G$ denote the sampled-data control system given by equations 2 and 6. Assume the control gain is $k^T = -B_1^T P$ where $P$ is a positive symmetric matrix that satisfies the algebraic Riccati equation,*

$$0 = A^T P + PA + I - P \left( B_1 B_1^T - \frac{1}{\gamma^2} B_2 B_2^T \right) P \tag{7}$$

*for some* $\gamma > 0$.

*Let* $x_r$ *denote the system's state at release time* $r[j]$. *If the system state* $x(t)$ *satisfies*

$$\begin{bmatrix} x(t) \\ x_r \end{bmatrix}^T \begin{bmatrix} -I + PB_1B_1^TP & -PB_1B_1^TP \\ -PB_1B_1^TP & 0 \end{bmatrix} \begin{bmatrix} x(t) \\ x_r \end{bmatrix} \leq -\|x(t)\|^2 \tag{8}$$

*for all* $t \in [r[j], r[j+1])$ *and* $j = 1, \ldots, \infty$, *then the induced* $\mathcal{L}_2$ *gain of* $G$ *is less than* $\gamma$.

*Proof.* The directional derivative of $V = x^TPx$ is

$$\dot{V} = \frac{\partial V}{\partial x}\left(Ax - B_1B_1^TPx_r + B_2w\right)$$

Using the standard completing the square argument and the Riccati equation 7, we rewrite the above equation as

$$\begin{aligned} \dot{V} &= \overline{x}^TX\overline{x} - \frac{1}{\gamma^2}x^TPB_2B_2^TPx + 2w^TB_2^TPx \\ &= \overline{x}^TX\overline{x} - \left\|\gamma w - \frac{1}{\gamma}B_2^TPx\right\|^2 + \gamma^2\|w\|^2 \\ &\leq \overline{x}^TX\overline{x} + \gamma^2\|w\|^2 \end{aligned}$$

where

$$X = \begin{bmatrix} -I + PB_1B_1^TP & -PB_1B_1^TP \\ -PB_1B_1^TP & 0 \end{bmatrix} \quad , \qquad \overline{x} = \begin{bmatrix} x \\ x_r \end{bmatrix} \tag{9}$$

Note that if

$$\overline{x}^TX\overline{x} \leq -\|x\|^2$$

then the above inequalities imply that

$$\dot{V}(x) \leq -\|x\|^2 + \gamma^2\|w\|^2$$

which is sufficient to ensure that the induced $\mathcal{L}_2$ gain of $G$ is less than $\gamma$ $\diamondsuit$

**Remark:** The matrix $X$ in equation 9 can be viewed as a collection of rank-one perturbations of the block diagonal matrix $\text{diag}(-I, 0)$. We can use this observation to show that $\overline{x}^TX\overline{x} \leq \lambda_1(k^T(x(t) - x_r))^2$ where $\lambda_1$ is the largest eigenvalue of $X$. So if we can ensure that $\lambda_1(k^T(x(t) - x_r))^2 < \|x(t)\|^2$, then we can again guarantee that the conditions in theorem 1 are satisfied. This is a more conservative condition than the one in theorem 1 and it is similar to the switching condition used in [6]. Therefore by using an analysis similar to that in [6] we can show that the "sampling period" is bounded below by a positive constant. In general, however, this lower bound can be an extremely conservative estimate of the sampling period.

After the task's release, we're interested in approximating the interval over which we can guarantee the condition in theorem 1. This time interval can be taken as an estimate of the task's next release time, $T^r$, which we treat as the task period. A reasonable approximation for this period is obtained by integrating the differential equation

$$\begin{aligned} \dot{x}(t) &= Ax - B_1 B_1^T P x_r \\ x(r) &= x_r \end{aligned} \tag{10}$$

Let $e^{At}$ be the transition matrix for $A$, so we can easily see that

$$x(t) = \left[ e^{At} \left( I + \int_0^t e^{-As} B_1^T ds \right) \right] x_r = \Phi(t) x_r \tag{11}$$

Because $A$ and $B_1$ are known, we can evaluate the matrix function $\Phi(t)$.

## 5 Schedulability with Deadlines less than Periods

The preceding section suggests that if our task retriggers itself so equation 8 is always satisfied, then our sampled-data system can guarantee the system's closed loop gain is less than $\gamma$. This may only happen, however, if the released tasks can meet their real-time deadlines. Earliest deadline first (EDF) schedulers are frequently used in periodically triggered real-time control systems. In the self-triggered system, however, release times will vary depending upon the system's current state. As a result we need to consider a scheduler that can adjust its task periods while assuring EDF schedulability and the "minimum" task period required by the application.

The **elastic task model** of Buttazzo et al., [7], is a popular method of adjusting task periods. The elastic task model uses a mechanical analogy to develop an algorithm for adjusting task periods. This analogy views tasks as being interconnected by "springs". The length of the spring represents the task's utilization and the spring constant represents that task's resistance to changing its utilization. Buttazzo's elastic task model was extended by Caccamo et al. [14] to handle uncertainties in computation time. A later paper [15] showed how to modify Buttazzo's algorithm to handle additional resource constraints. Hu et al [16] showed that Buttazzo's elastic scheduling algorithm can be viewed as minimizing a task set's summed squared utilization subject to the Liu-Layland EDF schedulability condition [17].

In our system, task deadlines will always be significantly less than task periods. This is needed to ensure a short time delay between the state sampling and the release of the control signal. Such task sets are schedulable under EDF if and only if

$$\sum_{i=1}^{N} \left( \left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1 \right) C_i \leq L \tag{12}$$

for all $L \in \mathcal{D}$ where

$$\mathcal{D} = \left\{ d_{i,k} \ : \ \begin{array}{l} d_{i,k} = kT_i + D_i \\ i \in \mathcal{N}, k \geq 0 \end{array} \right\}$$

This condition is proven by Baruah et al. [18] using a processor demand analysis.

Processor demand analysis requires that the total processor demand of all released tasks in an interval is less than or equal to the total processing power available in that interval. For task sets in which the deadline is less than the period, the $i$th task's processor demand over time interval $[0, L]$ is

$$C_i(0, L) = \left( \left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1 \right) C_i.$$

The condition in equation 12 is simply the sum of all demands, $C_i(0, L)$, over $i$ which must be checked for all possible future releases of the tasks.

In our application, release times are recomputed each time the task is called and so the task set is really not periodic. As a result we only need to check equation 12 over a subset of $\mathcal{D}$; namely those times that are less than or equal to the next release time. This idea was used in Chantem et al. [8] to propose a heuristic generalization of Buttazzo's elastic scheduling algorithm. The following theorem introduces a schedulability condition similar to that used in [8] which can be directly used with Buttazzo's algorithm.

**Theorem 2.** *Consider a task set in which all tasks are released at time* 0*. Assume that the task set is sorted in order of non-decreasing relative deadlines* ($D_i \leq D_{i+1}$) *and let* $\{\underline{T}_j\}_{j=1}^N$ *be a set of bounds on the task period that are generated recursively from*

$$\left\lfloor \frac{D_2 - D_1}{\underline{T}_1} \right\rfloor = \left\lfloor \frac{D_2}{C_1} - \sum_{i=1}^{2} \frac{C_i}{C_1} \right\rfloor \tag{13}$$

$$\left\lfloor \frac{D_{j+1} - D_j}{\underline{T}_j} \right\rfloor = \left\lfloor \frac{D_{j+1}}{C_j} - \sum_{i=1}^{j+1} \frac{C_i}{C_j} - \sum_{i=1}^{j-1} \left\lfloor \frac{D_{j+1} - D_i}{\underline{T}_i} \right\rfloor \frac{C_i}{C_j} \right\rfloor \tag{14}$$

*for* $j = 1, \ldots, N$*.*

*Let* $i^* = \arg\min_i \{D_i + T_i\}$ *then the task set will miss no deadlines over the interval from* $[0, D_{i^*} + T_{i^*}]$ *if* $T_j \geq \underline{T}_j$ *for all* $j = 1, \ldots, N$ *and*

$$\sum_{i=1}^{N} U_i \leq 1 - \frac{1}{\underline{T}_{i^*}} \sum_{i=1}^{N} C_i \tag{15}$$

*Proof.* To prove this theorem we need to demonstrate that the processor demand satisfies

$$\sum_{i=1}^{N} \left( \left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1 \right) C_i \leq L$$

over intervals

$$L \in \{D_1, \cdots, D_N, \min_i\{T_i + D_i\}\}$$

Essentially this means that the processor demand is satisfied between the consecutive release times.

When $L = D_i$, the processor demand can be written as a triangular system of algebraic equations

$$0 \leq D_1 - C_1 \tag{16}$$

$$\left\lfloor \frac{D_2 - D_1}{T_1} \right\rfloor C_1 \leq D_2 - \sum_{i=1}^{2} C_i \tag{17}$$

$$\left\lfloor \frac{D_3 - D_1}{T_1} \right\rfloor C_1 + \left\lfloor \frac{D_3 - D_2}{T_2} \right\rfloor C_2 \leq D_3 - \sum_{i=1}^{3} C_i \tag{18}$$

$$\cdots \leq \cdots \tag{19}$$

in which the $j$th term has the form,

$$\sum_{i=1}^{j-1} \left\lfloor \frac{D_j - D_i}{T_i} \right\rfloor C_i \leq D_j - \sum_{i=1}^{j} C_i$$

This is a triangular system of equations that we can solve recursively for $\underline{T}_i$. In particular the second equation (eqn. 17) yields equation 13. Applying this in a recursive manner yields equation 14. So if these equations are satisified then we can guarantee the processor demand is sufficient for time intervals equal to the task deadlines.

Now consider $L = T_j + D_j$ for any arbitrary $j$ and assume that $T_i \geq \underline{T}_i$ for all $i$. Then clearly,

$$\begin{aligned} D_j &\geq \sum_{i=1}^{N} \left( \left\lfloor \frac{D_j - D_i}{T_i} \right\rfloor + 1 \right) C_i \\ &\geq \sum_{i=1}^{N} \left( \left\lfloor \frac{T_j + D_j - D_i}{T_i} \right\rfloor - \left\lfloor \frac{T_j}{T_i} \right\rfloor - 1 + 1 \right) C_i \end{aligned}$$

where we used the fact that $\lfloor x + y \rfloor \leq \lfloor x \rfloor + \lfloor y \rfloor + 1$. We can now rearrange our last inequality to obtain

$$D_j + \sum_{i=1}^{N} C_i + \sum_{i=1}^{N} \left\lfloor \frac{T_j}{T_i} \right\rfloor C_i \geq \sum_{i=1}^{N} \left( \left\lfloor \frac{T_j + D_j - D_i}{T_i} \right\rfloor + 1 \right) C_i$$

The lefthand side of the above inequality can be bounded as

$$D_j + \sum_{i=1}^{N} C_i + \sum_{i=1}^{N} \left\lfloor \frac{T_j}{T_i} \right\rfloor C_i \leq D_j + \sum_{i=1}^{N} C_i + T_j \sum_{i=1}^{N} \frac{C_i}{T_i}$$

By the assumption in equation 15 we can see that

$$D_j + \sum_{i=1}^{N} C_i + \sum_{i=1}^{N} \left\lfloor \frac{T_j}{T_i} \right\rfloor C_i \leq D_j + \sum_{i=1}^{N} C_i - \sum_{i=1}^{N} C_i + T_j = D_j + T_j$$

So if the above condition holds we can ensure that

$$\sum_{i=1}^{N} \left( \left\lfloor \frac{T_j + D_j - D_i}{T_i} \right\rfloor + 1 \right) C_i \leq D_j + T_j$$

which is the inequality required to ensure that the processor demand is satisfied prior to the given release time. We choose $j = i^*$ to complete the proof. $\diamondsuit$

## 6 Self-triggered Real-time $\mathcal{H}_\infty$ Controllers

This section discusses how theorems 1 and 2 are used to elastically schedule self-triggered control tasks. Upon release, the $i$th task numerically integrates equation 10 forward to determine $T_i^r$. $T_i^r$ serves as the task's desired next release time. The scheduler handles $T_i^r$ as a request for the specified task period. If the scheduler simply grants the task's requested period, we say it is a **rigid** task scheduler. If the scheduler adjusts the requested period as is done in Buttazzo's algorithm, then we say the task scheduler is **elastic**.

Let $U_i^r = C_i/T_i^r$ denote the utilization requested by the $i$th task. For the simulations in section 7, our elastic task scheduler assigns task utilization, $U_i = C_i/T_i$, in a manner that solves the following optimization problem.

$$\begin{array}{ll} \text{minimize:} & \sum_{i=1}^{N} (U_i - U_i^r)^2 \\ \text{subject to:} & \underline{U}_i \leq U_i \leq \min(U_i^r, C_i/\underline{T}_i) \\ & \sum_{i=1}^{N} U_i \leq 1 - \overline{U} \end{array} \tag{20}$$

where $\underline{U}_i$ is the minimum individual task utilization for closed loop stability, $\underline{T}_i$ is the minimum task period required in theorem 2, $U_i^r = C_i/T_i^r$ is the task's requested utilization and

$$\overline{U} = \frac{1}{\min\{\underline{T}_i\}} \sum_{i=1}^{N} C_i$$

is the upper bound on the total utilization given in equation 15 of theorem 2. This optimization problem seeks to minimize the squared difference between a task's desired utilization, $U_i^r$, and its actual utilization, $U_i$. The allocation is done subject to constraints in theorem 2. These constraints require that the allocated utilizations assure closed loop stability while remaining schedulable under EDF.

It is important to note that the optimization problem in equation 20 is precisely the problem considered in Hu et al. [16] and Chantem et al. [8]. In those papers it was shown that Buttazzo's elastic scheduling algorithm [15] actually assigns task periods in a way that always satisfies the optimization problem in equation 20. So in our proposed self-triggered system, we can simply use Buttazzo's algorithm directly to allocate task utilization.

## 7 Simulation Results

This section presents preliminary simulation results for the real-time control of three identical inverted pendulums that are controlled by three control tasks running on the same processor. The controllers are full-information $\mathcal{H}_\infty$ controllers. The plants are all identical and the linearized state equation for the $i$th plant is

$$\dot{x}_i(t) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -(mg/M) & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & g/\ell & 0 \end{bmatrix} x_i(t) + \begin{bmatrix} 0 \\ 1/M \\ 0 \\ -1/M\ell \end{bmatrix} u_i(t) + \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} w_i(t)$$

where $M$ is the cart mass, $m$ is the mass of the pendulum bob, $\ell$ is the length of the pendulum and $g$ is gravitational accleration. The initial state is zero. For these simulations we let $M = 10$, $\ell = 3$, and $g = 10$. The system state $x = \begin{bmatrix} y & \dot{y} & \theta & \dot{\theta} \end{bmatrix}$ where $y$ is the cart's position and $\theta$ is the pendulum bob's angle with respect to the vertical.

The external disturbance, $w_i$, is time-varying and takes the form,

$$w_i(t) = \nu(t) + W\text{Rect}(t - \tau_i)$$

where $\nu$ is band-limited white noise with a noise power of 100 and sampling period of .001 sec. $\text{Rect}(t-\tau_i)$ is a unit rectangle function of duration 0.25 seconds starting at time $\tau_i$. $W$ represents the strength of the rectangular disturbance. In these simulations, all three plants are hit with the same rectangular disturbance at the same time ($\tau_i = 2$) and the disturbance level, $W$, was set to 1000.

In these simulations we required the state magnitude to lie below a specified level, $M$. In other words, we require $\|x_i(t)\|_2 \leq M$ for some specified $M \in \Re$. Immediately after the rectangular pulse, the system state changes and we assume that the next released task can measure this change. Once the sampled system state exceeds $M$, the task reduces its gain to $\gamma = 100$ to more aggressively reject the rectangular pulse. $M$ was set to 5 for these simulations. Once the system state is sufficiently small, the gain is reset to a large value ($\gamma = 500$) consistent with a less aggressive disturbance rejection objective. By adjusting the gain in this manner we kept the system output relatively small without having to use the more aggressive gain throughout the entire system's history.

We simulated the real-time system using a Matlab stateflow/simulink model. The model was built to accurately capture task timing that might be seen in actual real-time systems. The real-time computer was modeled as a stateflow chart that consisted of the parallel composition of three control tasks, six interrupt handlers, and two processes for the scheduler. The control tasks sample the plant state upon their release, compute the requested next release time $T^r$. Upon finishing their execution, these tasks output the control signal. This simulation, therefore, forces the control, $u_i(t)$, to be constant between consecutive finishing times rather than consecutive release times as assumed in theorem 1. The scheduler was implemented as two processes. One process assigned priorities

to the released control tasks and the other process used the requested sampling period to compute the actual release times.

We simulated three different cases. The first "baseline" case simulated the response of a single inverted pendulum by an "idealized" real-time system in which scheduler performance was not an issue. The second case simulated the response of a self-triggered real-time system under "rigid" and "elastic" scheduling. The third case simulated the response of the periodically-triggered real-time system under two different task periods. These three cases are discussed below.

**Baseline Case:** The baseline case simulated the response of a single inverted pendulum in which the control, $u_i(t)$, was constant between consecutive release times. Tasks were periodically released every 0.25 seconds. The controller's state feedback gain was chosen to ensure the closed loop system's $\mathcal{H}_\infty$ gain was less than 100. This case is therefore an "idealized" real-time system in which missed deadlines and processor contention are abstracted away. The lefthand side of figure 1 shows the time histories for the four system states: cart position, cart velocity, pendulum bob angle and angular velocity. This plot serves as a baseline against which the other cases will be compared. The plot shows that when the rectangular disturbances hits the system, the cart moves quickly to ensure the pendulum bob angle, $\theta$, remains small. This corrective action results in a large displacement, $x$, of the cart that takes about 10 seconds to return to the home position.

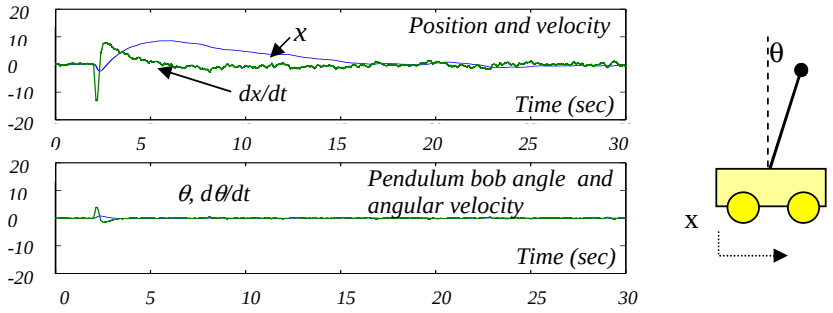


**Fig. 1.** Transient Response of Baseline System

**Self-Triggered Case:** The self-triggered case consisted of two simulations. One simulation used rigidly scheduled and the other used elastically scheduled task sets. The system clock ran at 0.001 seconds. All control tasks had identical computation times, $C_i = 50$ clock ticks, and deadlines, $D_i = 100$ clock ticks. Task periods were selected based on the results in theorem 1. In general, this resulted in a probabilistic distribution of task periods that were dependent on the system state at the release time. For a system gain $\gamma = 100$ and 500 the requested period averaged 200 and 500 clock ticks, respectively. In the "rigidly"

self-triggered system, these requested task periods were granted by the scheduler. The "elastically" self-triggered system had the requested task periods adjusted by the Buttazzo algorithm using the schedulability condition of theorem 2.

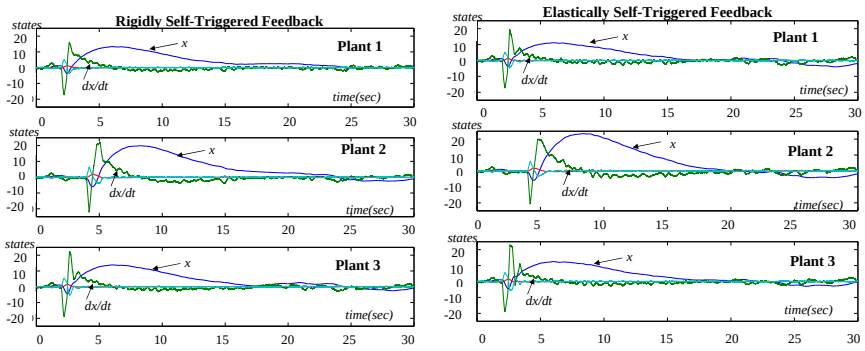


**Fig. 2.** State histories for rigidly (left) and elastically (right) self-triggered controllers

The state histories for the rigid and elastic self-triggered systems are shown in figure 2. The lefthand graphs are state histories for the three plants in the rigidly self-triggered system and the righthand graphs are for the elastically self-triggered system. What is perhaps most interesting here is that all systems have the same transient behavior regardless of whether task periods were assigned in an elastic or rigid manner. Comparing the state trajectories in figure 2 against the baseline trajectories in figure 1, we see little difference; thereby suggesting that the self-triggered system was maintaining the baseline transient behavior regardless of which scheduling scheme was used.

**Periodically-Triggered Case:** As a point of comparison we simulated a periodically triggered system. The task computation times and deadlines were 50 and 100 respectively for all tasks with a clock tick of 0.001 seconds. For one set of simulations, we set the task period to 250. The state trajectories for the three plants are shown in the lefthand plots of figure 3. A task period of 250 was the average task period for the rigidly scheduled self-triggered simulations. The lefthand plots in figure 3 are therefore comparable to the "rigidly scheduled" simulations in figure 2. In the other set of simulations, we set the task period to 500. This simulation's state histories are shown in the righthand plots of figure 3. A task period of 500 was the average task period for the "elastically scheduled" self-triggered simulations. The righthand plots are therefore comparable to the "elastically scheduled" simulations in figure 2. These simulations show that at the shorter task period (lefthand side of figure 3), the systems appears to have a transient response similar to that for the baseline system. At the longer task period (righthand side of figure 3), some of the plants become extremely oscillatory with one of the systems actually becoming unstable.

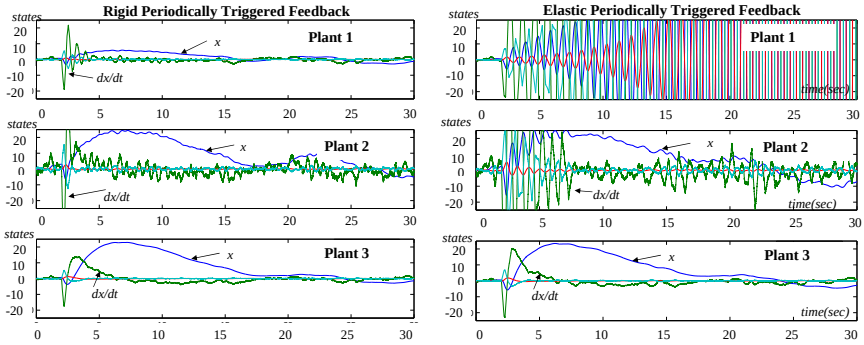


**Fig. 3.** State trajectories for periodically triggered system. (Left) Task period = 250, (Right) Task period = 500

## 8 Final Remarks

In comparing the simulation results between the baseline, periodically triggered, and self-triggered systems it should be apparent that the self-triggered system was able to maintain an acceptable level of transient performance regardless of whether a rigid or elastic scheduler was used. This was somewhat surprising at first glance. But in hindsight it was conjectured that this might be due to the inherent feedback nature of self-triggering. In selecting the next release based on the current state, a self-triggered system is using state feedback to adjust task periods in a way that assures overall system "performance".

The feedback nature of this interaction suggests that the performance of self-triggered systems should be very robust to processor overloads and late (delayed) jobs. Our simulations showed that the rigidly scheduled system was overloaded whereas the elastically scheduled system was not overloaded. Inspite of the fact that the rigidly scheduled system was overloaded, figure 2 clearly shows that the transient response is very similar to the baseline response. The task periods in the elastically scheduled self-triggered system were long enough to destabilize the periodically triggered system (figure 3). The results in figure 2 clearly show that even with this longer task period, the self-triggered system was able to preserve control performance. In other words, self-triggered systems appear to maintain acceptable levels of application performance in the face of significant processor overloading.

This paper's results therefore demonstrate that self-triggering can maintain acceptable levels of system performance regardless of whether or not we schedule in an elastic manner. In particular, these results seem to suggest a practical way for relaxing the need for "hard" real-time support in computer controlled systems. These results, however, are only preliminary and future work will need to more rigorously verify them.

## References


1. Arzen, K.: A simple event-based pid controller. In: Proceedings of the 14th IFAC World Congress. (1999)
2. Velasco, M., Marti, P., Fuertes, J.: The self triggered task model for real-time control systems. In: Work-in-Progress Session of the 24th IEEE Real-Time Systems Symposium (RTSS03). (2003)
3. Hristu-Varsakelis, D., Kumar, P.: Interrupt-based feedback control over a shared communication medium. In: Proceedings of the IEEE Conference on Decision and Control. (2002)
4. Astrom, K., Bernhardsson, B.: Comparison of riemann and lebesgue sampling for first order stochastic systems. In: Proceedings of the IEEE Conference on Decision and Control. (1999)
5. Voulgaris, P.: Control of asynchronous sampled data systems. IEEE Transactions on Automatic Control **39**(7) (1994) 1451–1455
6. Tabuada, P., Wang, X.: Preliminary results on state-triggered scheduling of stabilizing control tasks. In: IEEE Conference on Decision and Control. (2006)
7. Buttazzo, G., Lipari, G., Abeni, L.: Elastic task model for adaptive rate control. In: IEEE Real-Time Systems Symposium (RTSS). (1998)
8. Chantem, T., Hu, X., Lemmon, M.: Generalized elastic scheduling. In: Real-Time Systems Symposium (RTSS). (2006)
9. Zheng, Y., Owens, D., Billings, S.: Fast sampling and stability of nonlinear sampled-data systems: Part 2. sampling rate estimations. IMA Journal of Mathematical Control and Information **7** (1990) 13–33
10. Nesic, D., Teel, A., Sontag, E.: Formulas relating kl stability estimates of discrete-time and sampled-data nonlinear systems. Systems and Control Letters **38** (1999) 49–60
11. Seto, D., Lehoczky, J., Sha, L., Shin, K.: On task schedulability in real-time control systems. In: IEEE Real-time Technology and Applications Symposium. (1996) 13–21
12. Cervin, A., Eker, J., Bernhardsson, B., Arzen, K.E.: Feedback-feedforward scheduling of control tasks. Real-time Systems **23(1-2)** (2002) 25–53
13. Marti, P., Lin, C., Brandt, S., Velasco, M., Fuertes, J.: Optimal state feedback resource allocation for resource-constrained control tasks. In: IEEE Real-Time Systems Symposium (RTSS 2004). (2004) 161172
14. Caccamo, M., Buttazzo, G., Sha, L.: Elastic feedback control. In: IEEE Euromicro Conference on Real-Time Systems (ECRTS). (2000)
15. Buttazzo, G., Lipari, G., Caccamo, M., Abeni, L.: Elastic scheduling for flexible workload management. IEEE Transactions on Computers **51(3)** (2002) 289–302
16. Hu, X., Chantem, T., Lemmon, M.: Optimal elastic scheduling. In: IEEE Real-time and embedded technology and applications symposium - works in progress track. (2006)
17. Liu, C., Layland, J.: Scheduling for multiprogramming in a hard-real-time environment. Journal of the Association for Computing Machinery **20**(1) (1973) 46–61
18. Baruah, S., Rosier, L., Howell, R.: Algorithms and complexity concerning the preemptive scheduling of periodic , real-time tasks on one processor. Journal of Real-Time Systems **2** (1990) 301–324