Trigger Memoization in Self-Triggered Control

Indranil Saha
UCLA
indranil@cs.ucla.edu

Rupak Majumdar
MPI-SWS
rupak@mpi-sws.org

ABSTRACT

Self-triggered implementations of controllers have been proposed as an alternative to traditional *time-triggered* implementations. In a self-triggered implementation, the control task computes the actuator signal as well as a triggering time that specifies the next time instant at which the control task should be run. Self-triggered implementations have the potential to decrease communication costs and CPU requirements over time-triggered ones, e.g., by running the steady-state plant in open loop for long intervals if there is no disturbance. We show that commonly claimed gains for self-triggered implementations are too optimistic. The analysis of most self-triggering algorithms ignore the execution times for computing the trigger times. We show, using implementations of several self-triggering algorithms proposed in the literature on common embedded platforms, that the execution time to compute the trigger time can be non-negligible compared to the trigger times, and may even be higher than the trigger time itself, rendering a naive implementation infeasible.

We propose a hybrid implementation scheme for self-triggered control using state quantization and memoization of trigger times in a cache. We perform trigger-time computation tasks with low priority, and fall back on a time-triggered implementation when the trigger time computations are not guaranteed to finish in time (but use the computed results to update the cache). Our implementation achieves communication costs similar to self-triggered implementations and computation costs close to time-triggered implementations, while providing a bound for the region of practical stability.

Categories and Subject Descriptors

D.2.10 [**Software**]: Software Engineering—*Design-Methodologies*

Keywords

Control System, Self-Triggered Implementation, Stability, Memoization, WCET

EMSOFT'12, October 7-12, 2012, Tampere, Finland.

1. INTRODUCTION

Self-triggered implementations of digital controllers have been proposed recently as a computation- and communication-efficient technique for the software realization of a control law [26, 17, 19, 27, 18, 20, 4, 6]. In a self-triggered implementation, the control task computes, in addition to the actuation signal, the next instant of time at which the control law should be recomputed (the *trigger time*). In between the control updates, the actuation signal is held constant and the plant evolves in open loop. The appropriate choice of trigger time ensures that the resulting system is still stable and has required performance.

Self-triggering is an attractive technique for integrated architectures of cyber-physical systems, in which multiple control loops and non-critical applications share the same resources (CPU or communication network) [21]. Unlike the traditional *time-triggered* approach, where the control computation is performed periodically with a fixed period, it has the potential of making many fewer computations and lower use of the communication bandwidth. This is because the period in a time-triggered approach is chosen under a worst-case assumption, and control computations are performed at this rate even when the plant is in steady state and there are no disturbances [8]. Unlike *event-triggered* approaches [7, 25, 12], in which the state of the plant is continuously monitored using dedicated hardware to detect an event that triggers control re-computation, self-triggered approaches do not require the computation costs or extra hardware for continuous sensing. Indeed, simulations of some benchmark control systems demonstrate that self-triggered implementations can provide significant savings in both computation and communication [18, 20, 4, 6].

We show in this paper that savings estimates claimed for self-triggered implementations are somewhat optimistic. In most simulations of self-triggered implementations, it is assumed that the execution time required to compute the trigger time is negligible. We show that this assumption is not realistic for a number of common embedded platforms, and that the time required to compute the trigger time can indeed be more than the trigger time itself, making a direct implementation infeasible.

We propose an implementation scheme for self-triggered controllers using memoization of trigger times. We demonstrate that our implementation can provide similar savings in communication bandwidth as the naive self-triggered implementation, while keeping computation costs low. We maintain a *memoization region*: a subset of the state space around a given operating point for the controller. We quan-

tize the memoization region into a grid of chosen precision, and compute the trigger times for states on the grid on demand. To compute the trigger time of a state ξ, we compute its quantization on the grid and see if the trigger time has been cached from a previous computation. If so, we return the pre-computed value. If not, instead of computing the trigger time with the same priority as the control task, we schedule it as a background task of lower priority, and fall back on a time-triggered implementation in case it is not computed in time. If the operating point is fixed, the entire memo table can be pre-computed and there is no runtime overhead. However, in many cases the operating points can change dynamically and may not be known statically, or the space required to keep the entire grid may be too high. In these cases, we compute the entries of the memo table incrementally and on demand, falling back on a time-triggered implementation if the computation does not finish in time.

Memo table based implementations for explicit model predictive control have been suggested before [9, 3], where the control signals for a compact state space, computed by solving an optimization problem, are stored in a lookup table for fast actuation. We show memo tables allow fast implementation of self-triggered controllers as well, even with dynamic computations in case it is not feasible to pre-compute the entire table.

The quantization of states in computing trigger times introduces an error in the implementation of the controller, since the trigger time may not be computed for the given state, but for its approximation on the grid. We show how we can bound the effects of this error, and provide a bound on the practical stability of the controlled system [16, 22, 5] based on results on self-triggered controllers for control systems with bounded disturbances [4]. While we focus on trigger times, quantization and memoization can also be applied to the computation of the control law, and a similar error analysis can be performed. (In our examples, the computation of the control law takes negligible time in comparison to the computation of the trigger time, so we focus only on memoizing trigger times.)

We have developed a framework to evaluate self-triggered implementations of control systems. Our framework uses Truetime [10] for the simulation of control applications and aiT [11] for the computation of worst case execution times. We evaluate our implementation on the example of a batch reactor process, a standard linear system benchmark used in previous papers on self-triggered control [18, 20, 4], and an example of a jet engine compressor, a standard nonlinear control benchmark [6]. Experimental results show that our hybrid implementation attains communication costs close to that of naive self-triggered implementations, and simultaneously reduces computation costs significantly.

2. SELF-TRIGGERED CONTROL

We denote by $\mathbb{N}$, $\mathbb{R}$ and $\mathbb{R}_0^+$ the set of natural numbers, the set of real numbers, and the set of nonnegative real numbers, respectively. We denote by $\mathbb{R}^n$ the set of vectors of real numbers of length n. For a vector $x \in \mathbb{R}^n$, we use x_i, $1 \leq i \leq n$, to denote the i-th element of x. We write $\|x\|$ for the *Euclidean norm* of x, given by $\|x\|^2 = x_1^2 + x_2^2 + \ldots + x_n^2$. We denote by $\mathbb{R}^{n\times m}$ the set of real matrices with n rows and m columns. We write I_n for the $n \times n$ identity matrix. For a matrix $X \in \mathbb{R}^{n\times m}$, we write X^T for the transpose of X, $\lambda_{max}(X)$ and $\lambda_{min}(X)$ for its maximum and minimum eigenvalues, respectively, and $\|X\|$ for its norm, given by $\sqrt{\lambda_{max}(X^T X)}$. For a signal $u : \mathbb{R}_0^+ \to \mathbb{R}^n$, the $\mathcal{L}_\infty$ norm is denoted by $\|u\|_\infty$ and is given by $\|u\|_\infty = \sup_{t>0}\|u(t)\|$.

2.1 Controlled Dynamical Systems

The evolution of a dynamical system with time is captured by a differential equation:

$$\dot{\xi} = f(\xi, \upsilon),\ \xi(0) = \xi_0 \tag{1}$$

where $\xi(t) \in \mathbb{R}^n$ denotes the state of the system at time t and $\upsilon(t) \in \mathbb{R}^m$ denotes the external input to the system at time t. The curve $\xi : \mathbb{R} \to \mathbb{R}^n$ is said to be a *solution* or *trajectory* of (1) when there exists a piecewise continuous input curve $\upsilon : \mathbb{R} \to \mathbb{R}^m$ such that the time derivative of ξ at time t equals $f(\xi(t), \upsilon(t))$. The control system is called *linear time invariant* if there are matrices $A \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^{n\times m}$ such that

$$\dot{\xi}(t) = A\xi(t) + B\upsilon(t), \qquad \xi(t) \in \mathbb{R}^n, \upsilon(t) \in \mathbb{R}^m. \tag{2}$$

A controller

$$\upsilon(t) = k(\xi(t)) \tag{3}$$

computes the input $\upsilon(t)$ as a function of the state $\xi(t)$. The goal of controller synthesis is to compute a controller so that the closed loop system $\dot{\xi} = f(\xi, k(\xi(t)))$ exhibits some desirable properties. Below we define different stability criteria that are widely used as desirable properties of a closed loop control system. For more detailed analysis, we refer the readers to the book by Khalil [13].

The closed loop system is *stable* with respect to the origin $\xi = 0$, if for every $b > 0$ there exists a $a > 0$ such that

$$\|\xi(0)\| \leq a \quad \Rightarrow \quad \|\xi(t)\| \leq b \qquad \text{for all } t \geq 0$$

The closed loop system is called *asymptotically stable* if it is stable and a can be chosen so that

$$\|\xi(0)\| \leq a \quad \Rightarrow \quad \xi(t) \to 0 \quad as \quad t \to \infty.$$

If the above condition holds for any $a > 0$, the closed loop system is called *globally asymptotically stable* with respect to the origin. The closed loop system is called *exponentially stable* if there exist positive constants a, c and λ such that for all $\|\xi(0)\| \leq a$, $\xi(t)$ satisfies the inequality

$$\|\xi(t)\| \leq c\|\xi(0)\|e^{-\lambda t}, \qquad \forall t \geq 0.$$

If the above inequality holds for any a, the closed loop system is called *globally exponentially stable*.

The stability of a closed loop control system in the presence of a piecewise continuous disturbance input $d(t)$ is given by the concept of *input-to-state stability*. A closed-loop dynamical system with a *disturbance input* is represented as

$$\dot{\xi} = f(\xi, k(\xi(t)), d(t)).$$

The system is called *input-to-state stable* (ISS) if there exists a $\mathcal{K}_\infty$ function α, and a $\mathcal{KL}$ function β,[1] such that for

[1]A continuous function $\gamma : \mathbb{R}_0^+ \to \mathbb{R}_0^+$, is said to belong to class $\mathcal{K}$ if it is strictly increasing and $\gamma(0) = 0$. A $\mathcal{K}$-function γ is said to belong to class $\mathcal{K}_\infty$ if $\gamma(s) \to \infty$ as $s \to \infty$. A continuous function $\beta : \mathbb{R}_0^+ \times \mathbb{R}_0^+ \to \mathbb{R}_0^+$ is said to belong to class $\mathcal{KL}$ if $\beta(r, t)$ belongs to class $\mathcal{K}_\infty$ for each fixed $t \geq 0$ and $\beta(r, t)$ is decreasing with respect to t and $\beta(r, t) \to 0$ as $t \to \infty$ for each fixed $r \geq 0$.

any initial state $\xi(0)$, the state $\xi(t)$ satisfies the following inequality:

$$\|\xi(t)\| \leq \beta(\|\xi(0)\|, t) + \alpha(\|d\|_\infty), \qquad \forall t \geq 0$$

Sufficient conditions on different stability criteria are given using Lyapunov functions. A *Lyapunov function* is a continuously differentiable function $V : D \to \mathbb{R}$, $D \subset \mathbb{R}^n$, for which the following conditions hold:

$$V(0) = 0, \qquad V(\xi) > 0 \quad \text{for all } \xi \neq 0.$$

The derivative of V along the solution of the closed loop system is given by

$$\dot{V}(\xi) = \frac{\partial V}{\partial \xi} f(\xi, k(\xi(t)))$$

The closed loop system is stable with respect to $\xi = 0$ if there exists a Lyapunov function V such that $\dot{V}(\xi) \leq 0$ for all $\xi \neq 0$. The closed loop system is asymptotically stable with respect to origin if there is a Lyapunov function V such that $\dot{V}(\xi) < 0$ for all $\xi \neq 0$. The closed loop system is globally asymptotically stable if there exists a radially unbounded[2] Lyapunov function with $\dot{V}(\xi) < 0$. The closed loop system is exponentially stable if there exists a Lyapunov function V and a positive constant λ such that

$$V(\xi(t)) \leq V(0)e^{-\lambda t}, \qquad \text{for all } t \geq 0.$$

The largest constant that can be used for λ in the above inequality is called the *rate of decay*. The closed loop system is ISS if there exists a radially unbounded Lyapunov function V and two $\mathcal{K}_\infty$ functions α_1 and α_2 such that

$$\dot{V}(\xi, d) \leq -\alpha_1(\|\xi\|) + \alpha_2(\|d\|) \qquad \text{for all } \xi,\ d.$$

2.2 Self-Triggered Implementation

To implement the control law (3) on a digital computer, the state of the plant is sampled at a sequence of time instants $t_0 = 0,\ t_1,\ t_2, \ldots$ The state of the plant at time instant t_i is $\xi(t_i)$, and the computed control signal is given by $u(t_i) = k(\xi(t_i))$. The time instant t_i is called the *trigger time*. The control signal is applied to the plant immediately after it is available, and is held constant until the next trigger time t_{i+1}. The control signal computation time is generally in the microsecond range and can be considered to be negligible. Thus the control signal in the digital implementation of the system in (1) is given by

$$\upsilon(t) = \upsilon(t_i) \quad \text{for } t \in [t_i, t_{i+1}[\tag{4}$$

In a *time-triggered* implementation, the control engineer fixes a sampling period $T > 0$, and selects a periodic sequence of trigger times $t_i = iT$, for $i \in \mathbb{N}$. For *self-triggered* implementations, the sequence $\{t_i\}_{i\in\mathbb{N}}$ is computed online. At time t_i, in addition to the control signal, the next time instant t_{i+1} when the plant's state would be sampled is computed based on the current state $\xi(t_i)$. This computation is done based on a rule that is designed to ensure stability and desired performance of the control system.

We focus on linear control systems and briefly recall the self-triggered implementation of a linear controller achieving exponential stability as proposed in [18, 20]. Consider the linear time-invariant control system (2). The system is rendered closed loop exponentially stable by using a controller

$$\upsilon(t) = -K\xi(t). \tag{5}$$

where $\xi(t)$ is the solution of (2) at time t, and matrices A, B, and K are of appropriate dimensions. The closed loop system is given by

$$\dot{\xi} = (A - BK)\xi \tag{6}$$

As the closed loop system is stable, there exists a Lyapunov function

$$V(\xi(t)) = \xi(t)^T P \xi(t) \tag{7}$$

where P is a symmetric positive definite matrix. The matrix P can be obtained by solving the following Lyapunov equation:

$$(A - BK)^T P + P(A - BK) + Q = 0 \tag{8}$$

where Q is a given symmetric positive definite matrix.

The Lyapunov function in (7) admits an exponential decay. Let V be a Lyapunov function satisfying (8) and let $\lambda_0 > 0$ denote its rate of decay. For self-triggered implementations, we fix $0 < \lambda < \lambda_0$ and define a function S upper-bounding the evolution of V:

$$S(t, \xi(t_k)) = V(\xi(t_k))e^{-\lambda(t-t_k)}. \tag{9}$$

Thus, at $t = t_k$, we have $S(t, \xi(t_k)) = V(\xi(t_k))$ and for $t_k < t < t_{k+1}$, we have $S(t, \xi(t_k)) > V(\xi(t))$. The constant λ in the function S specifies the desired performance of the control system.

The self-triggered implementation has to ensure that the value of $V(\xi(t))$ never goes beyond the value of $S(t, \xi(t_k))$. To ensure this, a *triggering function* is defined as

$$h(t, \xi(t_k)) = V(\xi(t)) - S(t, \xi(t_k)) \qquad \text{for all } t \geq t_k \tag{10}$$

Triggering happens when $h(t, \xi(t_k)) = 0$. At that moment, the plant's state is sampled again. The triggering scheme ensures that there exists a positive constant $\tau_{\min}$ such that for any i, we have $\tau_i = t_{i+1} - t_i \geq \tau_{\min}$. The value of $\tau_{\min}$ is effectively computable from the parameters of the control system and the Lyapunov function (see Theorem 5.1 in [18]).

To implement self-triggered control, the designer fixes two design parameters: a maximum duration $\tau_{\max}$ between two triggering instants that determines how long the system is allowed to operate in open loop, and a discretization parameter $\Delta > 0$ that puts a grid on the interval $[\tau_{\min}, \tau_{\max}]$. Then, a discrete version of the triggering function h from (10) is defined:

$$\widetilde{h}(n, \xi(t_k)) = V(\xi(t_k + n\Delta)) - S(n\Delta, \xi(t_k)) \tag{11}$$

The self-triggered implementation $\Gamma_l : \mathbb{R}^n \to \mathbb{R}^+$, $\Gamma_l(\xi(t_k)) = t_{k+1}$ for linear control systems is given by:

$$\begin{aligned} t_{k+1} &= \max\{t_k + \tau_{\min}, t_k + n_k\Delta\} \\ n_k &= \max\{s \leq \lfloor \frac{\tau_{\max}}{\Delta} \rfloor \mid \text{ for all } n \in [0, s] : \widetilde{h}(n, \xi(t_k)) \leq 0\} \end{aligned} \tag{12}$$

Note that the worst case execution time depends on n_k and n_k depends on $\tau_{\max}$ and Δ. The self-triggered implementation scheme is feasible only if the time required to compute the trigger time t_{k+1} is less than $t_{k+1} - t_k$.

[2] A function $F : \mathbb{R}^n \to \mathbb{R}$ is called *radially unbounded* if $F(x) \to \infty$ as $\|x\| \to \infty$.

2.3 Problem: Trigger Time Computation

We now illustrate the problem of implementing the self-triggering scheme defined above through a motivating example of a batch reactor process originally described in [24] and used in [18, 20, 4] as a benchmark. The model of the plant is given by

$$\dot{\xi} = \begin{bmatrix} 1.38 & -0.20 & 6.71 & -5.67 \\ -0.58 & -4.29 & 0 & 0.67 \\ 1.06 & 4.27 & -6.65 & 5.89 \\ 0.04 & 4.27 & 1.34 & -2.10 \end{bmatrix} \xi + \begin{bmatrix} 0 & 0 \\ 5.67 & 0 \\ 1.13 & -3.14 \\ 1.13 & 0 \end{bmatrix} \upsilon.$$

The feedback controller

$$\upsilon = - \begin{bmatrix} 0.1006 & -0.2469 & -0.0952 & -0.2447 \\ 1.4099 & -0.1966 & 0.0139 & 0.0823 \end{bmatrix} \xi$$

renders the system exponentially stable. Let us denote the worst case execution time (WCET) of the computation of the trigger time t_{k+1} by τ_c. The rate of decay is $\lambda_0 = 0.41$. Setting $\lambda = 0.9\lambda_0$, we get $\tau_{\min} = 18ms$. The authors of [20] chose $\tau_{\max} = 358ms$ and $\Delta = 10ms$. The maximum possible value for n_k is thus 35. With these choices, we implement the trigger time computation given by (12) using the discrete triggering function (11) as a C program and cross-compile it using a GNU-based cross-compiling system. We then compute the WCET of the trigger time computation using aiT [11], a state-of-the-art worst case execution time analysis tool. The WCET of the trigger time is $29.793ms$ on a Leon 2 processor, assuming the processor frequency to be $100MHz$ (Most of the commercial implementations of Leon 2 processor have clock frequency below $100MHz$ [2]). As we see, the WCET is greater than $\tau_{\min}$. This implies that there may be cases where the controller will produce the next trigger time at an instant when the trigger time has already passed. This clearly shows the infeasibility of the implementation of the triggering rule provided in [20]. (In simulating the performance of their controller, the authors of [20] ignore trigger computation times.)

Note that we cannot increase the value of $\tau_{\min}$ as it depends on the parameters of the control system, and increasing $\tau_{\min}$ may cause the self-triggered implementation of the control system to violate stability requirements. The self-triggered implementation may be made feasible by decreasing the value of τ_{max}, as τ_{max} is a design parameter and the worst case execution time is directly proportional to τ_{max}. Decreasing the value of τ_{max} does not have any effect on the performance of the control system. However, it causes the trigger of the controller to occur more frequently than what the designer in [20] expected, and thus increases communication between the controller and the actuators (negating much of the benefits of self-triggering). Thus, the performance advantages of self-triggered implementations must be evaluated taking the computation time for the trigger time into account.

3. HYBRID IMPLEMENTATION

We now present a hybrid implementation scheme for self-triggered control systems. Our implementation utilizes a time-vs-space tradeoff, and memoizes trigger times for future reuse. We assume that a fixed-size piece of memory is available to store the *memoization table*, which maps a state of the system (a vector in $\mathbb{R}^n$) to a trigger time. The memoization table can be accessed using indices computed from a state as described later.

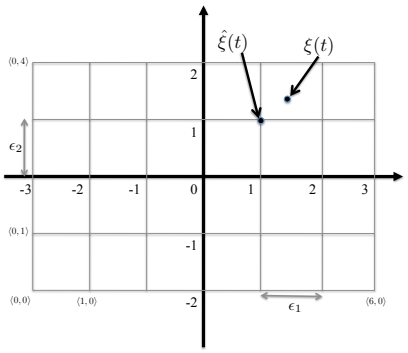


Figure 1: Memoization region and table

3.1 Memoized Trigger Time Computation

For the implementation, we fix a *memoization region* $[-w, w] \subseteq \mathbb{R}^n$ of the control system around the origin. We want to memoize the computed trigger time for any state in this region. Since the memoization table is finite, we discretize the memoization region using a *quantization factor* $\epsilon \in \mathbb{R}^n$. We describe the choice of ϵ in Section 3.3. The memoization region is represented as a multi-dimensional array, where each element is a trigger time. The number of entries in the table is given by

$$N = \prod_{i=1}^{n} \frac{2w_i}{\epsilon_i} \tag{13}$$

As an example, Figure 1 shows an example system with two state variables and the memoization region $[-3, 3] \times [-2, 2]$. The quantization factor $\epsilon = (1, 1)$, that is, each state is divided into intervals of size one unit. There are 24 different entries in the memoization table, and they are indexed as shown in the figure, from $(0, 0)$ at the bottom left corner to $(5, 3)$. The quantized value of a state ξ is the state corresponding to the closest grid point to its left and bottom, e.g., the state $\xi = (1.3, 1.3)$ is quantized to $\hat{\xi} = (1.0, 1.0)$, with index $(4, 3)$. Note that the difference between $\hat{\xi}(t)$ and $\xi(t)$ is bounded by the quantization factor ϵ, thus $\hat{\xi}(t) \geq \xi(t) - \epsilon$.

Algorithm 3.1 shows the pseudo-code for trigger-time computation with memoization. The function `findTriggerTime` takes as input the system state $\xi(t_k)$ at the kth sample time t_k and returns t_{k+1}, the next time the plant should be sampled. The memoization table `Memo` stores trigger times. The function `findIndex` takes the state $\xi(t_k)$ and returns the index of its discretization, if within the memoization region, or a special value denoting that the current state is outside the memoization region. In case the current state is outside the memoization region, the trigger time computation is scheduled (line 5). In case the current state is within the memoization region, the memoization table is first checked (line 7). If the trigger time has been pre-computed, it is returned (line 11). Otherwise, the trigger time computation is scheduled with the discretized state (line 9). When the computation is done, the memoization table is updated with the computed value.

The statement

$$\min(\tau_{\min},\ \texttt{schedule trigger}(\hat{\xi}))$$

Algorithm 3.1: Trigger Time Computation with Memoization.

Input: $\xi(t_k)$, the state of the system at time instant t_k

Output: t_{k+1}, the next trigger time of the controller

```
1  ;
2  function findDeliveryTime(ξ);
3  begin
4      ⟨s_1 ... s_n⟩ = findIndex(ξ);
5      if outsideRange(⟨s_1 ... s_n⟩) then
6          return min(τ_min, schedule trigger(ξ));
7      else
8          if Memo[s_1 ... s_n] is not found then
9              ξ̂ = quantizeState(⟨s_1 ... s_n⟩);
10             return min(τ_min, schedule trigger(ξ̂));
11         else
12             return Memo[s_1 ... s_n];
13         end
14     end
15 end
```

indicates that the task of computing the trigger time is scheduled as a background task of lower priority than the control loop. If the task finishes before $\tau_{\min}$, its value is returned, otherwise, the plant is sampled again at $t_k + \tau_{\min}$. That is, if the background task is not finished in time, the controller reverts to a time-triggered implementation with period $\tau_{\min}$. On completion of the task to compute the trigger time, the memoization table `Memo` is updated with the computed value (in case the state was within the memoization region). Moreover, in our implementation, when one trigger time computation is running, no other `trigger` task is spawned, and the controller works in a time-triggered mode using the minimum trigger time τ_{min} as the sampling period. We call this implementation scheme *hybrid*.

3.2 Dynamic Choice of Memoization Region

In Section 3.1 we assume that in the steady state the control system operates at the origin, and we fix the memoization region to be $[-w, w] \subseteq \mathbb{R}^n$. If a control system has only one operating point, the memoization table can be precomputed offline for all the states in the memoization region around the operating point instead of computing the table online. However, in reality, a control system may have a number of operating points, and the system can move from one operating point to another during its life cycle. All these operating points may not be known to the designer of the system a priori. For example, one of the states of the batch reactor process introduced in Section 2.3 is the temperature of the reactor. The desired temperature of the reactor is different during the different reaction phases. It is not possible to store the memoization table for all operating points in the memory.

We handle the change in operating point during runtime by online reconfiguration of a parameter used in Algorithm 3.1. Suppose the operating point changes from ξ_1 to ξ_2. We assume that the size of the memoization region remains the same for any operating point. Thus, the memoization region changes from $[\xi_1 - w, \xi_1 + w]$ to $[\xi_2 - w, \xi_2 + w]$. Even though there may be an overlap between the old and new memoization regions, the data in the overlapped region is not reused, as that data needs to be moved to a new location in the memoization table, and this operation may take many CPU cycles. The `findIndex` function in Algorithm 3.1 depends on an offset vector that maps a state in the memoization region to a particular position in the memoization table. When the operating point changes, we perform the following two actions. First, the memoization table is cleared. Second, the offset parameter is reconfigured to reflect the new memoization region.

3.3 Effect of State Quantization

As our memoization based implementation employs quantization of the state of the control system, we need to take into account the effect on the quantization on the triggering time. Since the trigger time computation does not have a closed form formula, correcting the triggering rule to take into account the quantization error is not straightforward. Rather, we show how the quantization error influences the behavior of the closed loop system in the steady state.

To show how the quantization error on the states effect the overall behavior of a linear control system, we resort to a result proved by Almeida, Silvestre, and Pascoal [4]. Consider the following linear time-invariant control system:

$$\dot{\xi}(t) = A\xi(t) + B\upsilon(t) + \delta(t), \qquad \xi(t) \in \mathbb{R}^n, \upsilon(t) \in \mathbb{R}^m, \quad (14)$$

where $\delta(t)$ is exogenous disturbance with bounded $\mathcal{L}_\infty$ norm δ_b. Note that the system in (14) is obtained by introducing an additive disturbance term $\delta(t)$ in the system in (2). In the presence of disturbance it is not possible to render the system (14) asymptotically stable to the origin. Rather, it was shown in [4] that by appropriately modifying the triggering rule in (10), it is possible to ensure that the system is exponentially stable with respect to a region around the origin.

Consider the Lyapunov function in (7). It can be shown that in the presence of bounded disturbance, the feedback law in (5) can render the states of the system (2) exponentially in a region defined by

$$V(\xi(t)) \leq \max\{V(\xi(t_0))e^{-\gamma_0(t-t_0)}, V_b\}, \quad (15)$$

where $\xi(t_0)$ denotes the initial state of the evolution at time t_0. Note that as $t \to \infty$, $V(\xi(t))$ is inside a region defined by V_b. For the system (14), V_b is given by

$$V_b = \frac{4\lambda^3_{\max}(P)\delta_b^2}{(\lambda_{\min}(Q) - \gamma_0\lambda_{\max}(P))^2}, \quad (16)$$

where P and Q are matrices in the Lyapunov Equation (8).

Now the performance function in (9) need be modified according to the behavior of $V(\xi(t))$ in (15). The modified specification function is given by

$$S'(t, \xi(t_k)) = \max\{V(\xi(t_k))e^{-\gamma(t-t_k)}, V_b\}, \quad (17)$$

where $0 < \gamma < \gamma_0$. The triggering function in (10) is now modified as follows:

$$h(t, \xi(t_k)) = U(t, \xi(t)) - S'(t, \xi(t_k)), \quad (18)$$

where $U(t, \xi(t))$ is given by

$$U(t, \xi(t_k)) = V(\xi(t_k)) + \mu(t, \xi(t_k)), \quad (19)$$

with $\mu(t, \xi(t_k)) = \beta(t)(2\|P\xi(t_k)\| + \lambda_{\max(P)}\beta(t))$ and $\beta(t) = \frac{\delta_b}{\sigma}(e^{\sigma(t-t_k)} - 1)$. The value of σ can be chosen as any value between $\|A\|$ and $\frac{1}{2}\lambda_{\max}(A + A^T)$.

As shown in [4], the self-triggered implementation in (12) with the triggering function given in (18) renders the states

of the system (14) in finite time to the set $\{\xi(t) \in \mathbb{R}^n : V(\xi(t)) \leq V_b\}$. We call this set the *stability region.*

We model the quantization error arising from the memoization based implementation of a self-triggered linear control system as the disturbance δ in (14). For a state $\xi(t)$ the control signal and the trigger time is computed by its quantized value $\hat{\xi}(t)$, where $\xi(t) - \hat{\xi}(t) = \bar{\epsilon}(t)$. The computed control signal is given by

$$\hat{\upsilon}(t) = -K\hat{\xi}(t) = \upsilon(t) + K\bar{\epsilon}(t).$$

Plugging the value of the control signal in (2) we get

$$\dot{\xi}(t) = A\xi(t) + B\upsilon(t) + BK\bar{\epsilon}(t), \qquad \xi(t) \in \mathbb{R}^n, \upsilon(t) \in \mathbb{R}^m. \tag{20}$$

Comparing (20) with (14) we get that the disturbance arising due to quantization in the states is given by

$$\delta(t) = BK\bar{\epsilon}(t). \tag{21}$$

Suppose we have been given an upper bound $V_b^{\max}$ on V_b as the specification. To find the upper bound of the quantization factor we proceed as follows: The upper bound on the $\mathcal{L}_\infty$ norm of $\delta(t)$ is given by

$$\delta_b^{\max} = \sqrt{\frac{(\lambda_{min}(Q) - \gamma_0\lambda_{\max}(P))^2}{4\lambda_{\max}^3(P)} V_b^{\max}}. \tag{22}$$

The upper bound on the quantization factor ϵ is found by solving the following optimization problem:

$$\begin{array}{ll} \text{maximize} & \epsilon_1 \\ \text{subject to} & \|BK\epsilon_1[1\ 1\ 1\ 1]^T\| \leq \delta_b^{\max}, \end{array} \tag{23}$$

where $\bar{\epsilon}(t) = \epsilon_1[1\ 1\ 1\ 1]^T$. Here we have chosen the quantization factors in all dimensions to be equal. The optimization problem in (23) is a convex one. The solution of the optimization problem provides the upper bound on the quantization factor of the state.

THEOREM 1. *Consider the linear control system* (2) *with stabilizing controller* (5) *and Lyapunov function V obtained from* (7) *and* (8)*. Let $V_b^{\max}$ be a given bound on the stability region. Suppose the controller is implemented using the hybrid implementation using Algorithm 3.1, using a discretization ϵ given by* (23)*. Then, the state ξ is guaranteed to converge to the set $\{\xi \in \mathbb{R}^n \mid V(\xi) \leq V_b^{\max}\}$.*

3.4 Fixed-point Representation

The trigger time is computed in (12) as a floating point entity. A single precision floating point value needs four bytes of space in memory. To save memory, we can store the trigger time as a fixed-point number. The trigger times are always positive, thus we do not need to deal with the sign of a number. The fixed-point representation of a trigger time is given as a pair $\langle n, m\rangle$ consisting of a *length* $n \in \mathbb{N}$, and a *length of the fractional part* $m \in \mathbb{N}$. The length of the integer part is $n - m$, if $n > m$, and 0 otherwise. To achieve the highest precision for a chosen length m of the representation, the length of the fractional part m is chosen such that the maximum trigger time $\tau_{\max}$ satisfies $2^{n-m-1} \leq \tau_{\max} < 2^{n-m}$. In this case, the maximum error in the fixed-point representation is given by 2^{-m}. For example, if the range of the trigger time to be represented is given by $[0.020, 0.400]$ and we choose $n = 8$, then $m = 9$ and the bound on the error in the representation is given by 2^{-9}.

Given a computed trigger time t, its fixed-point representation can be stored as an integer given by $\hat{t} = \lfloor (t \times 2^m) \rfloor$. From the fixed-point representation $\hat{t}$, the floating point value $\widetilde{t}$ of the trigger time is obtained by $\widetilde{t} = 2^{-m}\hat{t}$. To use fixed-point representations in Algorithm 3.1, line 9 and line 11 are adapted appropriately. Note that it is always safe to use $\widetilde{t}$ obtained from the memoization table instead of using t, as $\widetilde{t} \leq t$.

4. NONLINEAR SYSTEMS

We now extend the hybrid implementation to self-triggered implementations of nonlinear control systems.

Triggering Rule. Consider the nonlinear dynamical system (1). We recall the setting of [6]. The objective of the controller is to ensure that an invariant on the state of the plant, given as the specification, is never violated. Henceforth, we will refer to this specification as *invariant specification.* The idea behind a self-triggered implementation is that at any time instant t, the difference between the current state $x(t)$ and the last measured state $x(t_i)$ for $i \in \mathbb{N}$ is bounded in such a way that the invariant specification on the closed loop system is not violated. This error is referred as the *measurement error* and is given by

$$e(t) = \xi(t_i) - \xi(t) \quad \text{for } t \in [t_i, t_{i+1}]. \tag{24}$$

The dynamics of the closed loop control system is given by:

$$\dot{\xi} = f(\xi, k(\xi + e)). \tag{25}$$

Now if the control law is designed to render the system (1) ISS with respect to measurement error, there exists a Lyapunov function V for the system satisfying the following inequality:

$$\dot{V} \leq -\alpha_1(\|x\|) + \alpha_2(\|e\|). \tag{26}$$

where α_1 and α_2 are $\mathcal{K}_\infty$ functions. To maintain stability of the closed loop system (25), we have to ensure that $\dot{V} < 0$. This can be ensured if the following condition holds:

$$\alpha_2(\|e\|) \leq \kappa\alpha_1(\|x\|), \qquad \kappa > 0. \tag{27}$$

The triggering strategy in a nonlinear control system is designed based on the invariant specification on the states of the system around the equilibrium point. If a Lyapunov function satisfying the inequality in (26) can be discovered for the system, then an invariant on the admissible error can be derived from the invariant on the states using (27). As shown in [6], these two invariants can be used to derive a closed form formula z to compute the next sampling time t_{k+1} from the sampled state $\xi(t_k)$ at the current sampling time t_k.

The self-triggered implementation $\Gamma_{nl} : \mathbb{R}^n \to \mathbb{R}^+$, $\Gamma_{nl}(\xi(t_k)) = t_{k+1}$ for nonlinear control systems is given by:

$$t_{k+1} = z(\xi(t_k)). \tag{28}$$

Hybrid Implementation. Given the triggering rule (28), the hybrid implementation is similar to the implementation of linear controllers, where we discretize the state space and memoize computed trigger times.

Since there is a closed form formula to compute the triggering time, we can correct the triggering times by taking into account the maximum possible error introduced by quantizing the states. To find the maximum possible error

introduced in the trigger time, we solve the following optimization problem:

$$\begin{array}{rl} \text{maximize} & \tau_e \\ \text{Subject to} & \tau = z(\xi) \quad \hat{\tau} = z(\hat{\xi}) \\ & \tau_e = \tau - \hat{\tau} \\ & \xi - \hat{\xi} \leq \epsilon \quad \xi \geq \hat{\xi}. \end{array} \tag{29}$$

In this optimization problem, ξ denotes the sampled state and $\hat{\xi}$ denotes the quantized state obtained from ξ. We denote by τ and $\hat{\tau}$ the trigger time computed based on ξ and $\hat{\xi}$ respectively. We maximize τ_e, the difference between τ and $\hat{\tau}$, subject to additional constraints that the difference between ξ and $\hat{\xi}$ is bounded by the quantization factor ϵ and $\xi \geq \hat{\xi}$.

For a given state $\xi(t_k)$, if the maximum value of τ_e is obtained as τ_e^{max} then the triggering time is computed by the following modified form of (28):

$$t_{k+1} = z(\hat{\xi}(t_k)) - \tau_e^{max}. \tag{30}$$

Note that unlike linear control systems, the control signal in a nonlinear control system is computed based on $\xi(t_k)$ instead of $\hat{\xi}(t_k)$.

5. EXPERIMENTS

Experimental Setup. In our experiments we have used the Truetime simulator [10] to implement the control tasks for our example control systems and simulate the systems under different conditions. We choose PowerPC MPC5xx [1] and Leon2 [2] processors as the two target platforms on top of which the embedded control applications are built. The PowerPC MPC5xx is widely used in Delphi Corporation and Robert Bosch GmbH to develop engine controllers. The Leon 2 processor was developed by European Space Research and Technology Centre to be used for European space projects. The worst-case execution times [28] for control computation and trigger time computation on the target processors have been obtained using the worst-case execution time analysis tool aiT [11]. The control computations and trigger time computations are implemented in C, and then cross-compiled for respective processors using GNU-based cross compilation systems. The worst-case execution times are computed by aiT based on the compiled binary code. The computation of trigger time involves computation of square root function and exponential function. We implement the square root computation using the *Babylonian method* [14]. The exponential function is computed by expanding it as a Taylor series.

We report CPU times and communications costs. Given a time interval, CPU times are reported as the total CPU time required by all control tasks over the time interval. The communication cost is the number of control tasks run over a time interval. This captures the cost of communication, since each control task requires transmitting the plant state from the sensors to the controller and the actuation signal from the controller to the actuators.

5.1 Examples

We demonstrate our results on a benchmark linear control system: a *batch reactor processor*, and a benchmark nonlinear control system: a *jet engine compressor*.

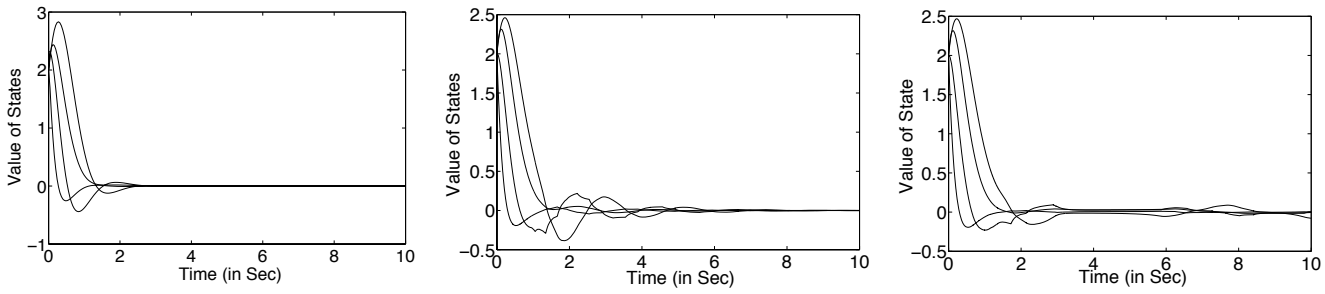
Linear System: Batch Reactor Process. First, we present our results on the batch reactor process from Section 2.3. The Lyapunov function for the closed loop system has been computed using (8) with $Q = I_4$. With this choice of Q, we have $\lambda_{\max}(P) = 1.2185$ and $\lambda_{\min}(Q) = 1$. The other parameters are chosen in accordance with [4]: $\tau_{\min} = 39.8ms$, $\tau_{\max} = 720ms$, $\Delta = 10ms$, $\gamma_0 = 0.08207$, $\gamma = 0.008207$. We compute the values of $\tau_{\max}$ on both PowerPC and Leon2 processors to ensure that the trigger computation time τ_c does not exceed the minimum trigger time $\tau_{\min}$. The values of modified $\tau_{\max}$ are $510ms$ and $270ms$ on PowerPC and Leon2, respectively.

Suppose we want the state of the system to converge in a stability region $V_b^{\max}$ of size 0.5 when the system is free from any external disturbance. The upper bound on the quantization factor can be found to be 0.04346 by first finding the value of $\delta_b^{\max}$ for $V_b^{\max} = 0.5$ using (22), and then solving the optimization problem in (23). We choose the quantization factor to be 0.04 in each dimension. We store the trigger times in the memoization table as a 8-bit fixed-point number. The fixed-point representation is given by $\langle 8, 8\rangle$, as $0.5 < \tau_{\max} < 1$. The trigger time retrieved from the memoization table may by at most 2^{-8} less than its computed value. We choose the memoization region to be $[-0.5, 0.5]$ in each dimension. With the quantization factor to be 0.04, the number of entries in the memoization table is computed by (13) to be 390625. As each entry of the memoization table takes 1 byte, the memoization table requires at most 381.47KB. Note that if this amount of memory is not available, we have two options. First, we may increase the value of the quantization factor which in turn increases the bound on the size of the stability region. Second, we may shrink the memoization region without changing the quantization factor. The second option ensures that the specification on the size of the stability region is met, but we memoize trigger time for the states in a narrower region, thus the computation time grows.

We compare three different implementations for the batch reactor processor: (i) a time-triggered implementation using the sampling period $\tau_{\min}$, (ii) a self-triggered implementation using the triggering rule (12), where the triggering function is given by (18). The maximum trigger time $\tau_{\max}$ is updated to ensure that the computation of the trigger time is guaranteed to be finished before the minimum trigger time $\tau_{\min}$ ($\tau_{max} = 510ms$ for PowerPC and $\tau_{max} = 270ms$ for Leon2 processor), and (iii) our hybrid implementation, without decreasing the maximum trigger time ($\tau_{max} = 720ms$). We fall back to the time-triggered implementation using trigger time $\tau_{\min}$ when the trigger time computation takes more than $\tau_{\min}$ time.

We simulate the self-triggered batch reactor process on three scenarios: (a) the system is free from any external disturbance (Disturbance free), (b) the system is subjected to a periodic external disturbance signal of pulse shape with amplitude 8, period 800 samples, pulse width 40 samples, zero phase delay, and sample time $10ms$. (Disturbance scenario 1), and (c) the system is subjected to a normally distributed random disturbance signal with mean 0, variance 1, and sample time $1ms$ (Disturbance scenario 2).

The evolution of the state of the batch reactor process with initial state $\langle 2, 2, 2, 2\rangle$ is shown in Figure 2 for the case when no external disturbance is present. While the state of the time-triggered implementation and the self-triggered implementation eventually go to the origin, the state of the



(a) TT (b) ST (c) Hybrid

Figure 2: Evolution of states of a batch reactor process from initial state $\langle 2, 2, 2, 2\rangle$ for (a) time-triggered (TT), (b) self-triggered (ST), and (c) Hybrid implementation

Implementation	Disturbance Free		Disturbance Scenario 1		Disturbance Scenario 2	
	PowerPC	Leon2	PowerPC	Leon2	PowerPC	Leon2
TT	0.164s	0.226s	0.164s	0.226s	0.164s	0.226s
ST	167.514s	342.554s	167.188s	341.117s	167.691s	342.541s
Hybrid	0.959s	1.824s	4.039s	7.336s	31.322s	57.360s

Table 1: CPU time required for different implementations of the controller of the batch reactor process running for 2000s

hybrid implementation eventually enters the region defined by $V_b^{\max}$ and remains there forever.

Table 1 and Table 2 show the CPU time and communication cost for different implementations of the controller for the batch reactor process for 2000s runtime. From Table 1, we see that the naive self-triggered implementation takes significantly more CPU time in comparison to the time-triggered implementation. The hybrid implementation brings the required CPU time close to that of the time-triggered implementation. Table 2 shows that the number of communications from the plant to the controller improves significantly in self-triggered implementations, as claimed in the existing literature on self-triggered control systems. The Hybrid implementation maintains similar communication cost as the naive self-triggered implementations. For self-triggered implementation, the number of communications is always more for Leon2 processor, as τ_{max} for Leon2 is less than that for PowerPC. For Hybrid implementation, the number of communications is more for Leon2 processor than that for PowerPC as the trigger time computation takes more time with the Leon2 processor than with the PowerPC processor and the system remains in time-triggered mode when the trigger time computation is running.

Figure 3 shows how the CPU time required for the controller of the batch reactor process for different implementations using Leon2 processor vary for different duration of runtime between $500s$ and $5000s$ when there is no external disturbance acting on the system. The figure shows that the naive self-triggered implementation always takes significantly more computation time than the time-triggered implementation. However, our hybrid implementation is always close to the time-triggered implementation in terms of computation time, and as time progresses and the memoization table gets filled up with trigger times, the difference between the computation time for our hybrid implementation and that of the time-triggered implementation slowly decreases.

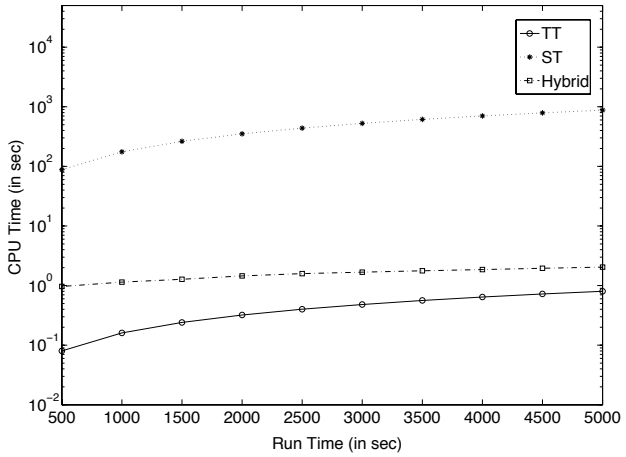


Figure 3: CPU time required for batch reactor process for different implementations using the Leon2 processor for different running times in the disturbance-free scenario

Nonlinear System: Jet Engine Compressor. We illustrate our experimental results on nonlinear self triggered control systems using the standard benchmark of a jet engine compressor. The model of the system originally appeared in [15]. In [6], it was adapted to translate the desired equilibrium point to the origin as follows:

$$\begin{aligned} \dot{\xi_1} &= -\xi_2 - \frac{3}{2}\xi_1^2 - \frac{1}{2}\xi_1^3 \\ \dot{\xi_2} &= \frac{1}{\beta^2}(\xi_1 - u) \end{aligned} \tag{31}$$

where ξ_1 and ξ_2 denote the mass flow and pressure rise in the compressor respectively. The symbol β is a constant

Implementation	Disturbance Free		Disturbance Scenario 1		Disturbance Scenario 2	
	PowerPC	Leon2	PowerPC	Leon2	PowerPC	Leon2
TT	50251	50251	50251	50251	50251	50251
ST	5731	7422	14735	15786	5974	7422
Hybrid	5868	5891	15847	15868	6932	7343

Table 2: Communication cost for different implementations of the controller of the batch reactor process running for 2000s

positive parameter. The output of the controller u denotes the throttle mass flow.

A control law $u = k(\xi_1, \xi_2)$ was designed in [6] to render the system in (31) globally asymptotically stable. The control law is given by:

$$u = \xi_1 + \frac{\beta^2}{4}(z(\xi_1^2 + 1) + 3\xi_1^2 y + \xi_1^3 + \xi_1 + 3y) \quad (32)$$

where $y = (3\xi_1^2 + 2\xi_2 - \xi_1)/(\xi_1^2 + 1)$ and $z = 2\xi_1 y(y-3) + \xi_1^2(4y-6)$. By substituting u with its corresponding expression, the closed loop system becomes

$$\begin{aligned} \dot{\xi_1} &= -\frac{1}{2}(\xi_1^2 + 1)(\xi_1 + y) \\ \dot{\xi_2} &= -(\xi_1^2 + 1)y \end{aligned} \quad (33)$$

The following ISS Lyapunov function is provided in [6]

$$V = 1.46\xi_1^2 - 0.35\xi_1 y + 1.16y^2,$$

which is computed using SOStools [23]. Using this Lyapunov function, this can be shown that the state of the closed loop system $\xi = (\xi_1, y)^T$ and the measurement error on the state $e = (e_1, e_2)^T$ are related by the following inequality:

$$0.90\|e\|^2 \leq 0.74\nu^2\|\xi\|^2$$

Now based on the specification that the state ξ is contained in the invariant set $\{\xi \in \mathbb{R}^n | V(\xi) = 27.04\}$, the following formula is provided in [6] to compute the trigger time:

$$t_{k+1}(\xi(t_k)) = \frac{29\xi_1(t_k) + \|\xi(t_k)\|^2}{5.36\|\xi(t_k)\|\xi_1(t_k)^2 + \|\xi(t_k)\|^2}\tau^*,$$

where $\tau^* = 7.63ms$ for $\nu = 0.33$.

The operating point of the jet engine is $\xi_1 = 0, \xi_2 = 0$. The state ξ_1 is mass flow, which is a positive quantity. The state ξ_2 is pressure rise, which may be both positive and negative. We choose the memoization region to be $\langle \xi_1 \in [0, 0.5],\ \xi_2 \in [-1, 1] \rangle$. The minimum trigger time $\tau_{min} = \tau^* = 7.63ms$. We choose the maximum trigger time $\tau_{max} = 250ms$, and store the trigger times in the memoization table as a fixed-point number with representation $\langle 8, 10 \rangle$. This enables us to store the trigger time in 1 byte. We assume that we have 256KB memory available to store the memoization table. With this amount of memory, the quantization factor can be chosen to be 0.002 in each dimension, using (13). For this quantization factor, we solve the optimization problemn (29) to find out the value of τ_e^{max}. The optimization problem is not convex and we solve the problem numerically by dividing the region $\langle \xi_1 \in [0, 0.5],\ \xi_2 \in [-1, 1] \rangle$ into a grid with precision 0.0001 and exhaustively searching all the points on the grid. We find that the value of τ_e^{max} for the values of ξ_1 in the range $[0, 0.25]$ is fairly high (greater than the minimal trigger time 7.63ms). We shift the memoization region in the ξ_1 direction to exclude the region from the memoization region. Thus our memoization region is $\langle \xi_1 \in [0.25, 0.75],\ \xi_2 \in [-1, 1] \rangle$. In the memoization region, τ_e^{max} is 6.634ms.

We consider two scenarios to compare the performance of the hybrid implementation with the time-triggered and self-triggered implementations without memoization. In both scenarios we consider the evolution of the system for $2000s$ in the presence of external disturbances. In the first scenario, the disturbance signal is a periodic pulse signal with amplitude 8, period 400 samples, pulse width 40 samples, zero phase delay and sample time $10ms$. In the second scenario, the disturbance is a normally distributed random signal with mean 1, variance 1, and sample time $1ms$. We do not show results for the scenario when no external disturbance is present, as the memoization-based implementation does not provide any benefit due to not including a region around the origin (ξ_1 is in the range $[0, 0.25]$). Table 3 shows the CPU time and the communication cost for the two processors for different implementations of the controller for runs of length $2000s$. The results show that a self-triggered implementation without memoization itself gives significant savings on both the CPU time and number of communications. However, with memoization, we can gain even more in CPU time with a slight increase in the number of communications. The number of communications increases in the hybrid implementation as we need to decrease the trigger time to correct any error arising due to quantization of states in the trigger time computation. On both processors, the time to compute the trigger time is always less than the minimum trigger time τ_{min}. Thus the hybrid implementation never falls back to the time-triggered implementation, and the number of communications for both the processors are the same.

Figure 4 shows how the CPU time required for jet engine compressor for different implementations using PowerPC processor vary in the disturbance scenario 2 for different duration of runtime between $500s$ and $5000s$. The figure shows that the savings in CPU time grows with time from the time-triggered implementation to self-triggered implementation without memoization, and also from the self-triggered implementation without memoization to the hybrid implementation. As time progresses, more and more trigger times are memoized, and the ratio of cache hit to cache miss also increases. Thus the CPU time requirement keeps on decreasing with time for the hybrid implementation.

Acknowledgements. We thank AbsInt for making the aiT tool available to us for free. We thank Adolfo Anta, Manuel Mazo Jr., Paulo Tabuada, and Majid Zamani for helpful discussions. We thank Toyota Motors for partially sponsoring the research.

Implementation	Disturbance Scenario 1			Disturbance Scenario 2		
	Computation		Commu-	Computation		Commu-
	PowerPC	Leon2	nication	PowerPC	Leon2	nication
TT	1.094s	1.732s	262122	1.094s	1.732s	262122
ST	0.338s	0.473s	18729	0.303s	0.422s	16447
Hybrid	0.190s	0.315s	19009	0.117s	0.211s	16464

Table 3: CPU time and communication cost for different implementations of the controller of the jet engine compressor running for 2000s

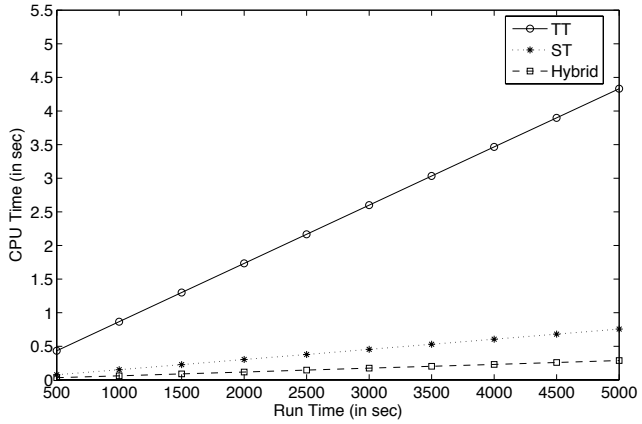


Figure 4: CPU time required for the jet engine compressor for different implementations on the PowerPC processor for different running times for disturbance scenario 2

6. REFERENCES

[1] Powerpc 5xx controllers. http://www.freescale.com/webapp/sps/site/taxonomy.jsp?code=DRMCRMPC500MC.

[2] Leon2 processor. http://vlsicad.eecs.umich.edu/BK/Slots/cache/ www.gaisler.com/products/leon2/leon.html.

[3] A. Alessio and A. Bemporad. A survey on explicit model predictive control. In *Nonlinear Model Predictive Control*, volume 384 of *LNCIS*, pages 345–369. Springer, 2009.

[4] J. Almeida, C. Silvestre, and A. M. Pascoal. Self-triggered state feedback control of linear plants under bounded disturbances. In *Proc. CDC*, pages 7588–7593, 2010.

[5] A. Anta, R. Majumdar, I. Saha, and P. Tabuada. Automatic verification of control system implementations. In *Proc. EMSOFT*, pages 9–18, 2010.

[6] A. Anta and P. Tabuada. To sample or not to sample: Self-triggered control for nonlinear systems. *IEEE Transaction on Automatic Control*, 55(9):2030–2042, 2010.

[7] K.-E. Årzén. A simple event-based PID controller. In *Proc. IFAC*, volume 18, pages 423–428, 1999.

[8] K. J. Åström and B. Wittenmark. *Computer-controlled systems: theory and design*. Prentice-Hall, Inc., 2nd edition, 1990.

[9] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos. The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1):3–20, 2002.

[10] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K.-E. Årzén. How does control timing affect performance? Analysis and simulation of timing using Jitterbug and TrueTime. *IEEE Control Systems Magazine*, 23(3):16–30, 2003.

[11] R. Heckmann and C. Ferdinand. Worst-case execution time prediction by static program analysis. White paper, AbsInt Angewandte Informatik GmbH, 2009.

[12] W. P. M. H. Heemels, J. H. Sandee, and P. P. J. Van Den Bosch. Analysis of event-driven controllers for linear systems. *Intl. J. of Control*, 81(4):571–590, 2008.

[13] H. K. Khalil. *Nonlinear Systems*. Prentice Hall, 2002.

[14] O. Kosheleva. Babylonian method of computing the square root: Justifications based on fuzzy techniques and on computational complexity. In *Annual Meeting of the North American Fuzzy Information Processing Society (NAFIPS)*, pages 1–6, 2009.

[15] M. Krstic and P. Kokotovic. Lean backstepping design for a jet engine compressor model. In *Proceedings of IEEE Conf. Control App.*, pages 1047–1052, 1995.

[16] J. LaSalle and S. Lefschetz. *Stability by Lyapunov's Direct Method*. Academic Press, Inc., 1961.

[17] M. Lemmon, T. Chantem, X. S. Hu, and M. Zyskowski. On self-triggered full information H-infinity controllers. In *Proc. HSCC*, pages 371–384, 2007.

[18] M. Mazo Jr., A. Anta, and P. Tabuada. On self-triggered control for linear systems: Guarantees and complexity. In *Proc. ECC*, 2009.

[19] M. Mazo Jr. and P. Tabuada. On event-triggered and self-triggered control over sensor/actuator networks. In *Proc. CDC*, pages 435–440, 2008.

[20] M. Mazo Jr. and P. Tabuada. Input-to-state stability of self-triggered control systems. In *Proc. CDC*, pages 928–933, 2009.

[21] R. Obermaisser, C. E. Salloum, B. Huber, and H. Kopetz. From a federated to an integrated architecture. *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, 28(7):956–965, 2009.

[22] A. Podelski and S. Wagner. Region stability proofs for hybrid systems. In *Proc. FORMATS*, pages 320–335, 2007.

[23] S. Prajna, A. Papachristodoulou, P. Seiler, and P. A. Parrilo. SOSTOOLS: Control applications and new developments. In *Proc. IEEE CASC*, pages 315–320, 2004.

[24] H. H. Rosenbrock. *Computer-Aided Control System Design*. Academic Press, 1974.

[25] J. Sandee. *Event-driven control in theory and practice: Tradeoffs in software and control performance*. PhD thesis, Technische Universeteit Endhoven, 2006.

[26] M. Velasco, P. Martí, and J. M. Fuertes. The self-triggered task model for real-time control systems. In *Work In Progress Proceedings of RTSS*, pages 67–70, 2003.

[27] X. Wang and M. Lemmon. Self-triggered feedback control systems with finite gain $\mathcal{L}_2$ stability. *IEEE Transaction on Automatic Control*, 54(3):452–467, 2009.

[28] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström. The worst-case execution-time problem – overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, 7(3):36:1–36:53, 2008.