Trigger Memoization in Self-Triggered Control

Indranil Saha[1] and Rupak Majumdar[1,2]

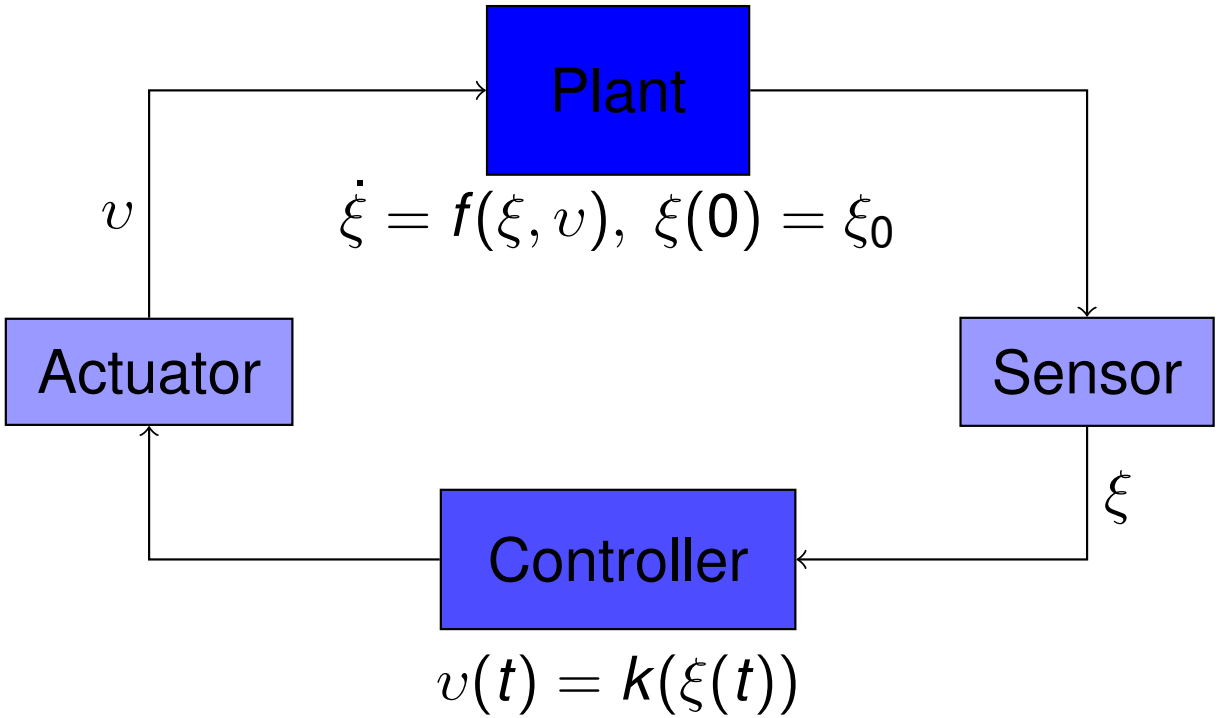
[1]University of California Los Angeles

[2]Max Planck Institute for Software Systems

EMSOFT 2012

October 8, 2012

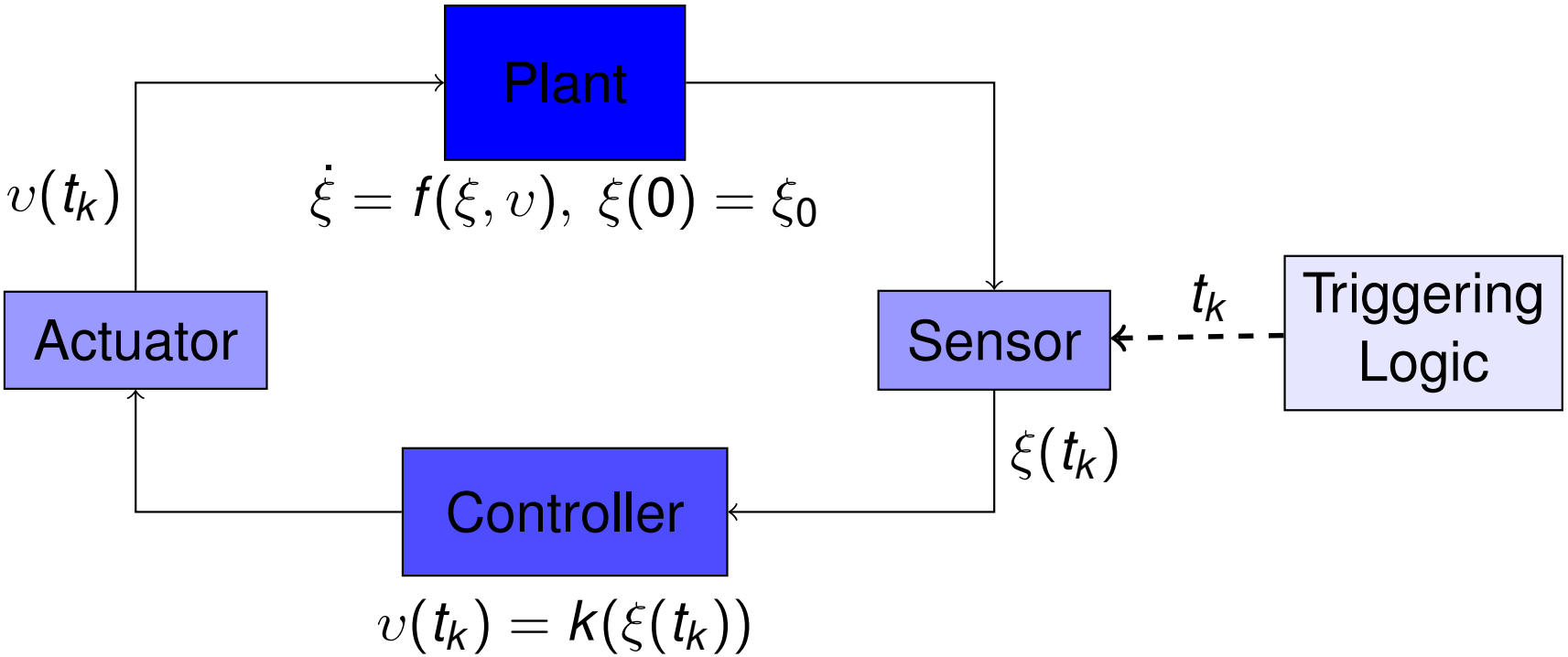
A Control System



ξ - State of the plant

υ - Control signal generated by the controller

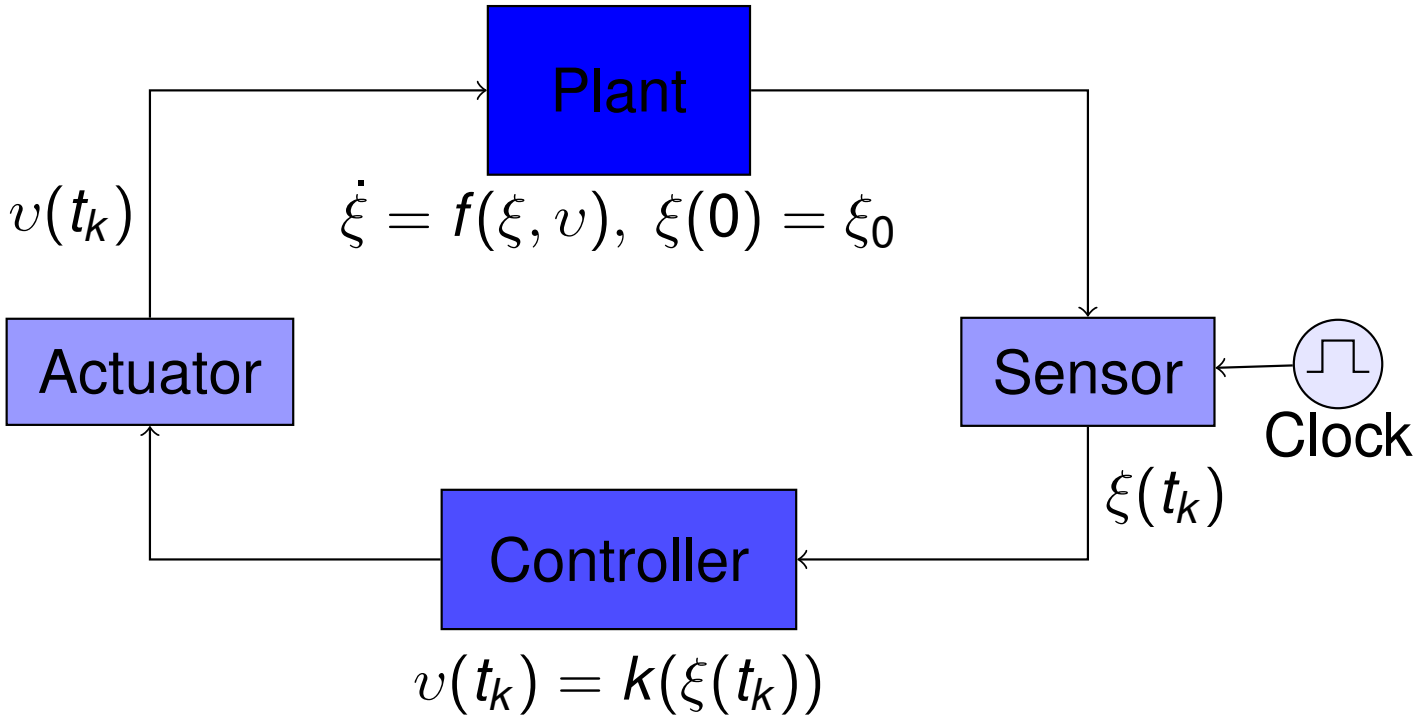
Implementation of a Control System



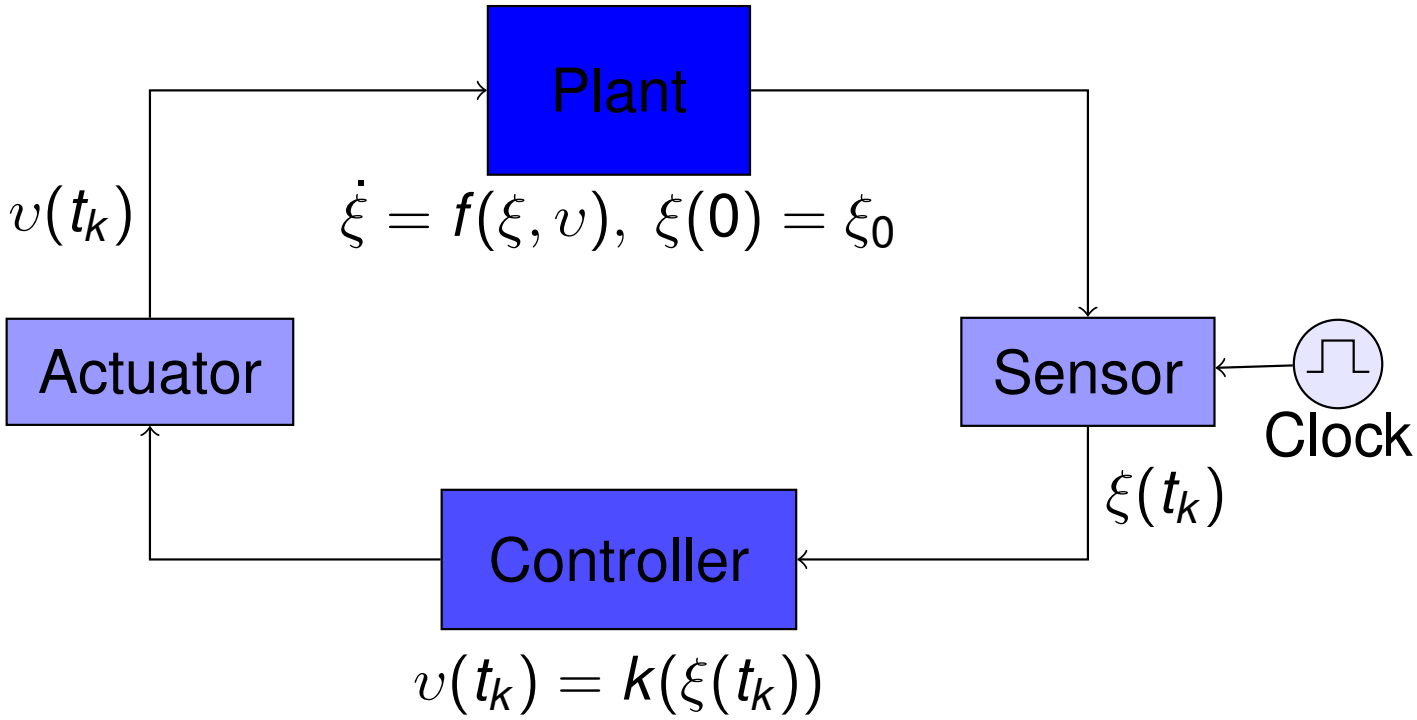
To implement the control law on a digital computer, the state of the plant is sampled at a sequence of time instants $t_0 = 0,\ t_1,\ t_2, \ldots$

The time instant t_k is called the trigger time

Time-Triggered Implementation



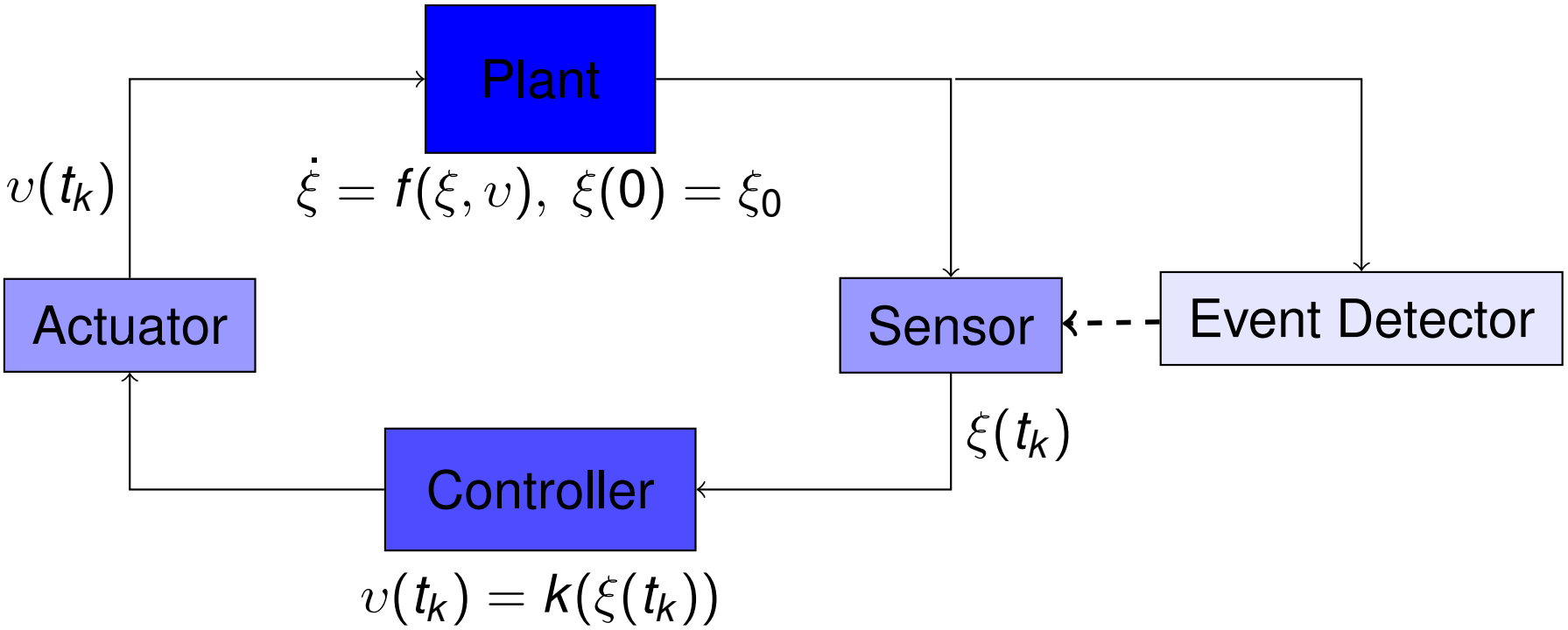
Time-Triggered Implementation



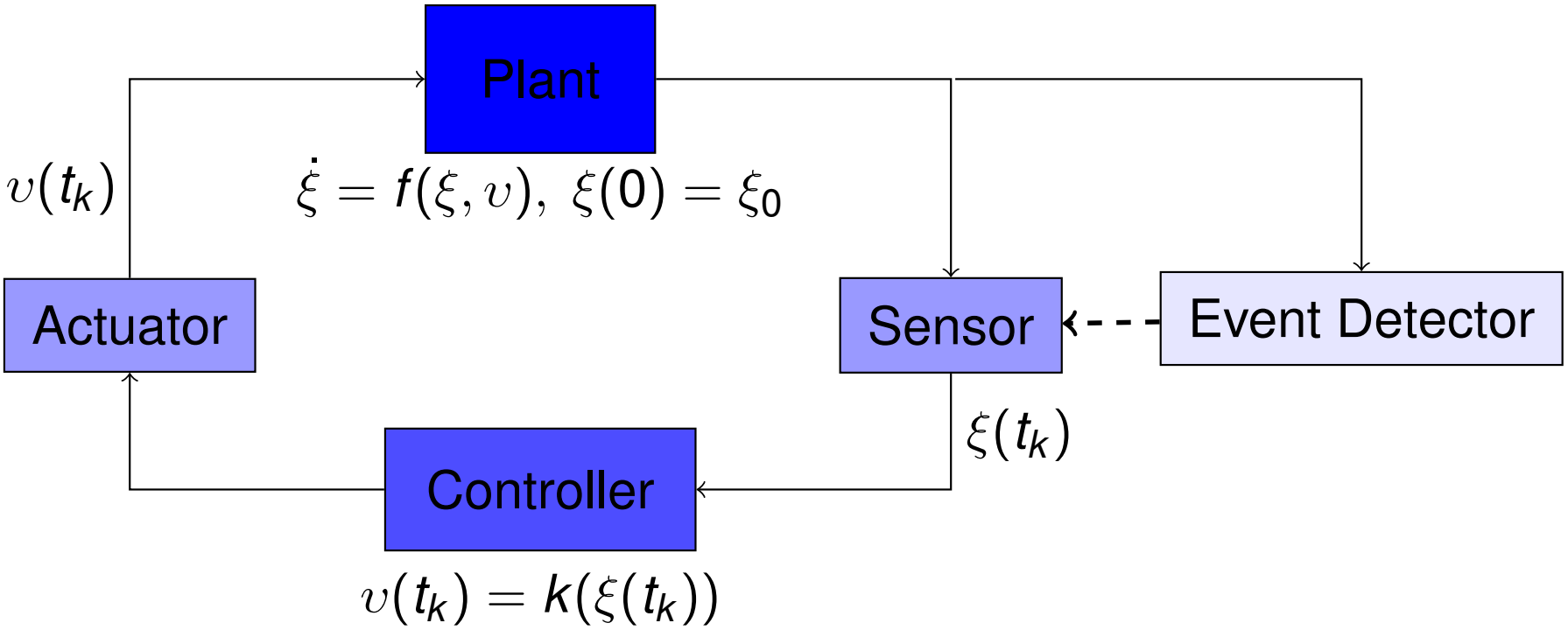
Sampling period is selected based on the worst case scenario

Inefficient usage of computational resource and communication bandwidth

Event-Triggered Implementation

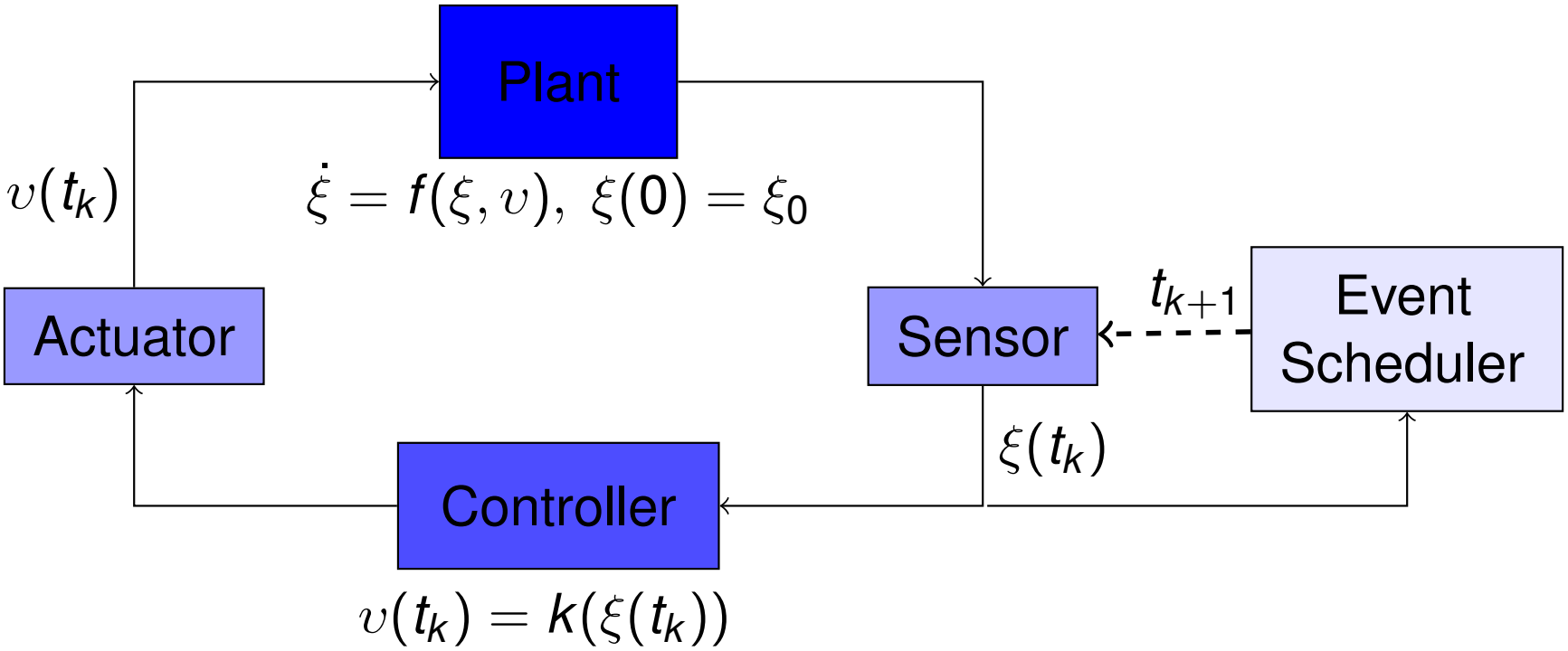


Event-Triggered Implementation

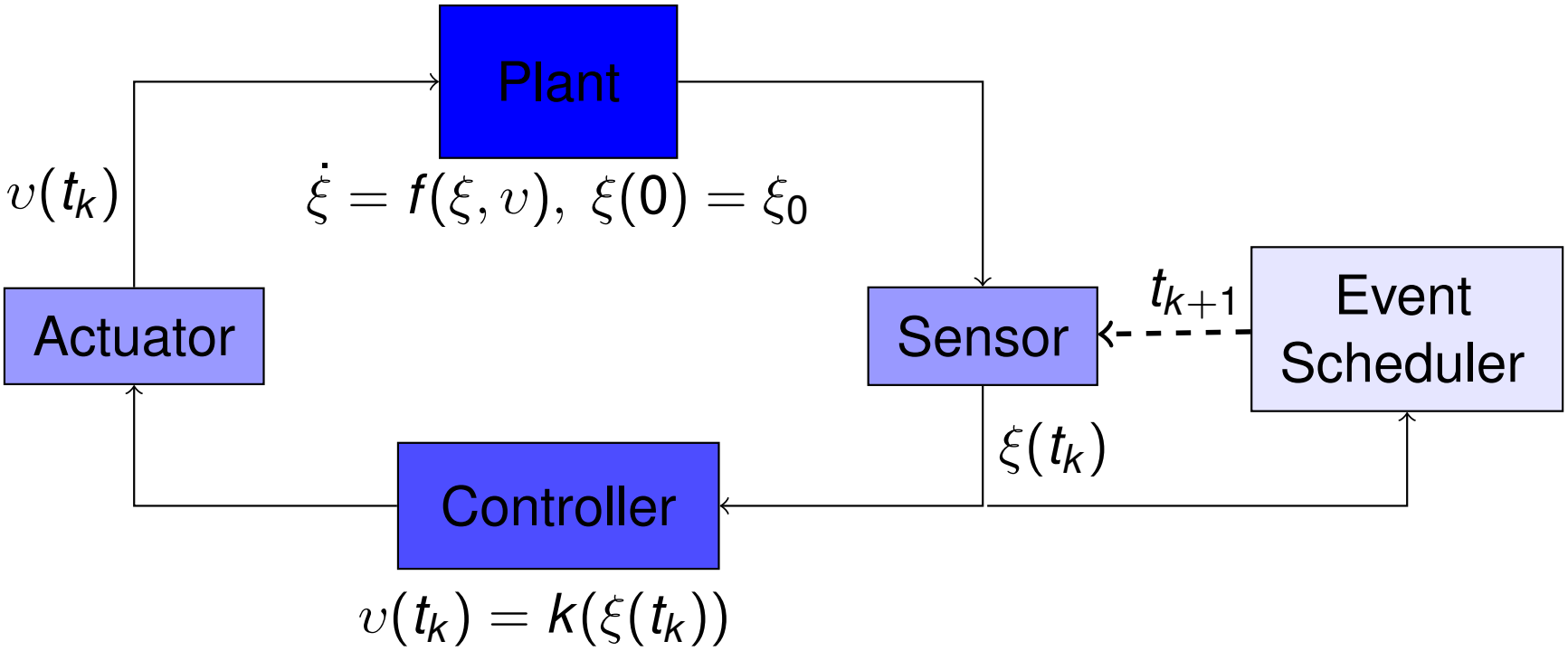


Requires dedicated hardware to monitor plant state

Self-Triggered Implementation



Self-Triggered Implementation



Has been shown to reduce the number of control computations significantly with respect to its time-triggered counterpart

Trigger Time Computation

Trigger time is computed based on two parameters:

- τ_{min} : minimum trigger time
  - The trigger-time which works in the worst case scenario
  - Can be computed from the parameters of the control system
- τ_{max}: maximum trigger time
  - The maximum duration the plant can be kept open loop
  - A design parameter

$$(t_k + \tau_{min}) \leq t_{k+1} \leq (t_k + \tau_{max})$$

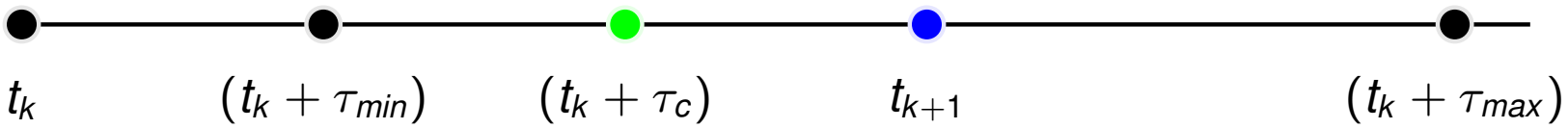
Computation Time of Trigger Time

τ_c - The time required to compute the trigger time

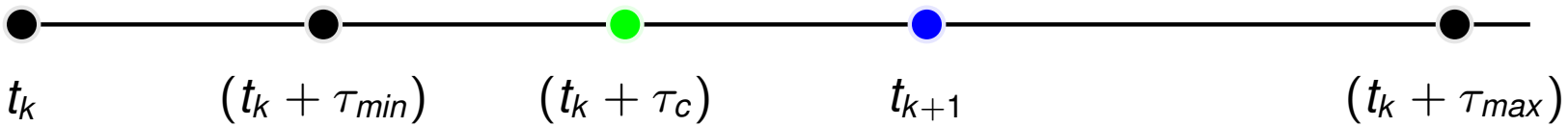
The self-triggered implementation scheme is feasible if and only if

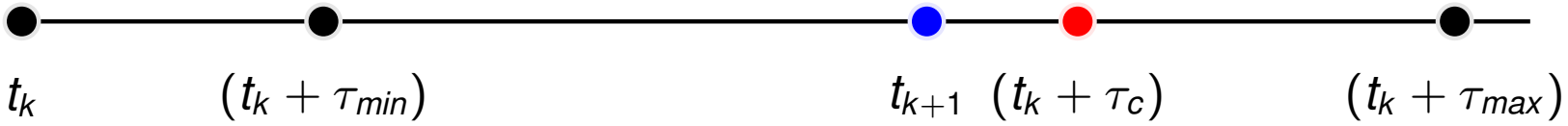
$$(t_k + \tau_c) \leq t_{k+1}$$

The Problem

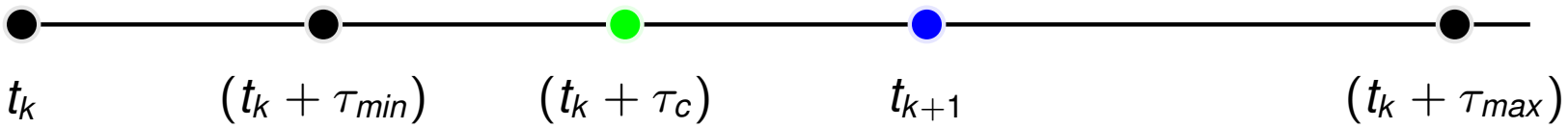


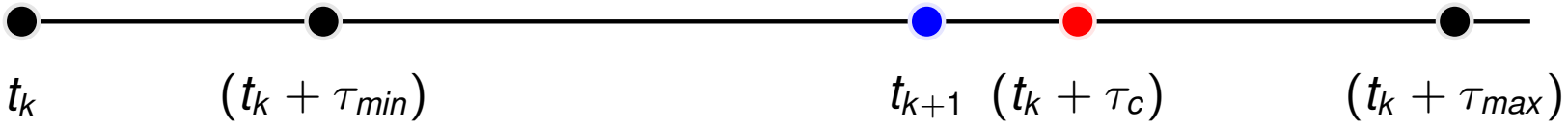
The Problem

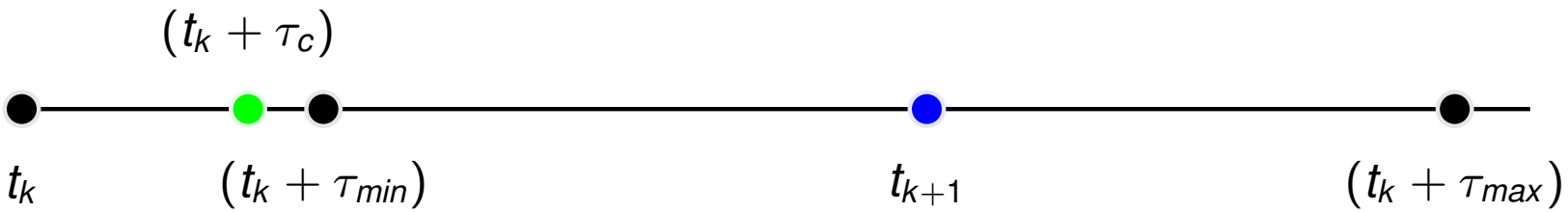

t_k
$(t_k + \tau_{min})$
$(t_k + \tau_c)$
t_{k+1}
$(t_k + \tau_{max})$


t_k
$(t_k + \tau_{min})$
t_{k+1}
$(t_k + \tau_c)$
$(t_k + \tau_{max})$

The Problem


t_k
$(t_k + \tau_{min})$
$(t_k + \tau_c)$
t_{k+1}
$(t_k + \tau_{max})$


t_k
$(t_k + \tau_{min})$
t_{k+1}
$(t_k + \tau_c)$
$(t_k + \tau_{max})$


$(t_k + \tau_c)$
t_k
$(t_k + \tau_{min})$
t_{k+1}
$(t_k + \tau_{max})$

Stability of a Control System

The closed loop system is exponentially stable if there exists a Lyapunov function V and a positive constant λ such that

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

The largest constant that can be used for λ in the above inequality is called the *rate of decay*.

Existence of such a Lyapunov function ensures that the state of the control system converges to the stable state or origin with an exponential decay.

Self-Triggered Control of a Linear Control System

Let V be a Lyapunov function 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:

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

The constant λ in the function S specifies the desired performance of the control system.

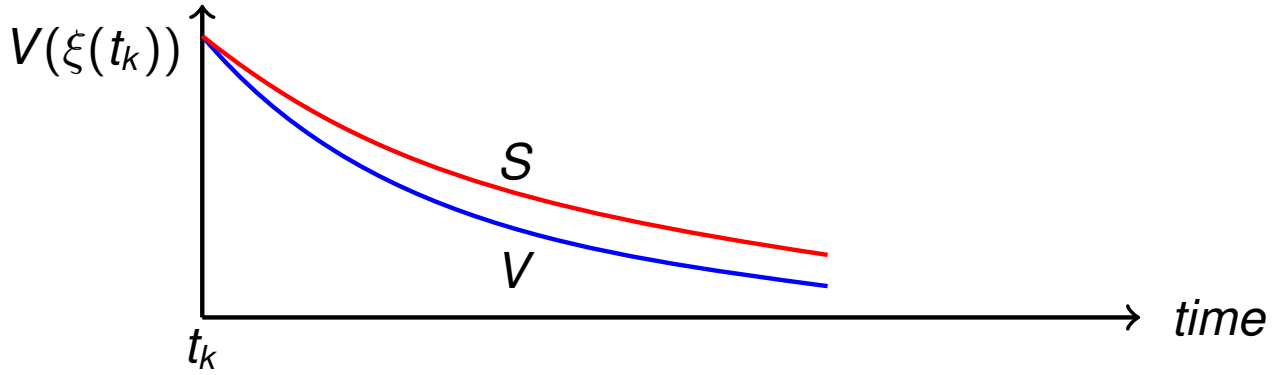
Self-Triggered Control of a Linear Control System

Let V be a Lyapunov function 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:

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

The constant λ in the function S specifies the desired performance of the control system.



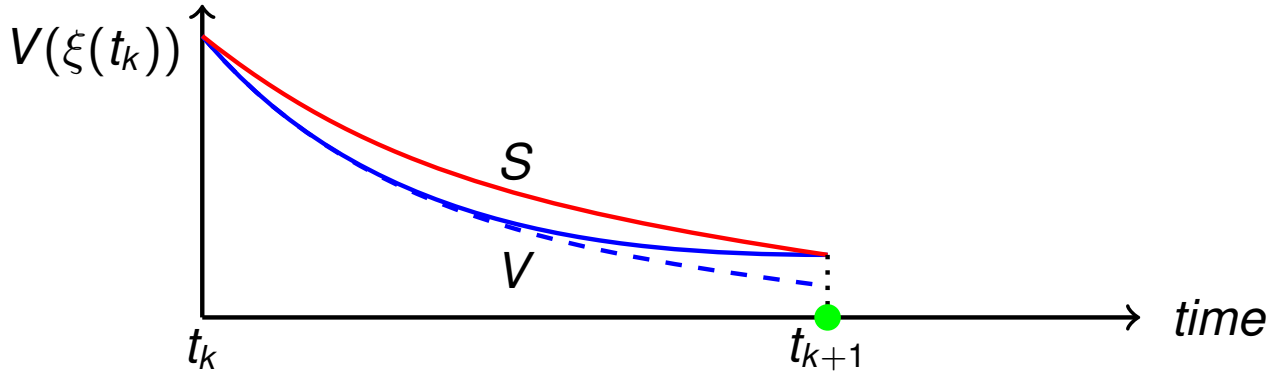
Self-Triggered Control of a Linear Control System

Let V be a Lyapunov function 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:

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

The constant λ in the function S specifies the desired performance of the control system.



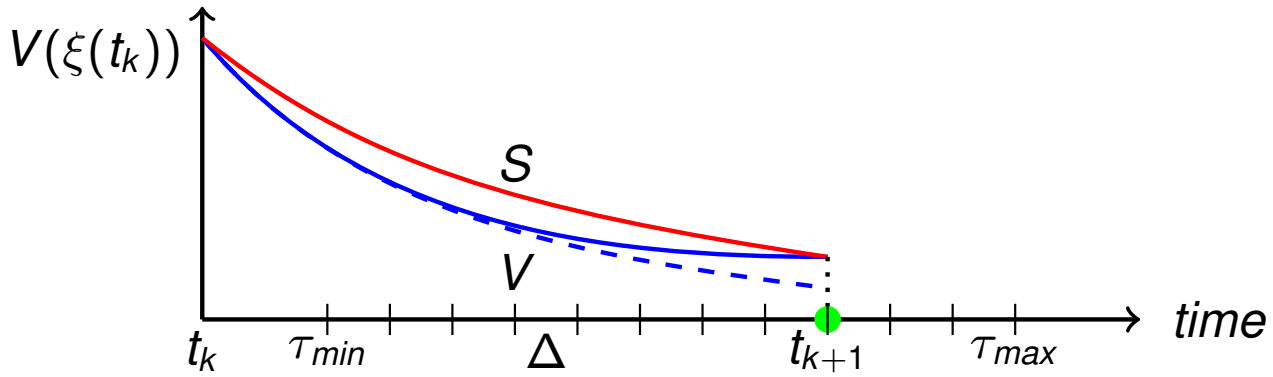
Self-Triggered Control of a Linear Control System

Let V be a Lyapunov function 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:

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

The constant λ in the function S specifies the desired performance of the control system.



An Example

The model of a batch reactor process 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.

For this system, $\tau_{\text{min}} = 18ms$
Following literature we chose $\tau_{\text{max}} = 358ms$
On a Leon 2 processor with frequency $100MHz$, the WCET of the trigger time computation is $29.793ms$

An Example

The model of a batch reactor process 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.

For this system, $\tau_{\text{min}} = 18ms$
Following literature we chose $\tau_{\text{max}} = 358ms$
On a Leon 2 processor with frequency $100MHz$, the WCET of the trigger time computation is $29.793ms$

It is possible that $(\mathbf{t_k} + \tau_\mathbf{c}) > \mathbf{t_{k+1}}$

Proposed Solution: Memoization of Trigger Time

- A time-vs-space tradeoff - memoizes trigger times for future reuse
- A *memoization region* $[-w, w] \subseteq \mathbb{R}^n$ is fixed around the origin of the control system
- The memoization region is discretized using a quantization factor $\epsilon \in \mathbb{R}^n$
- A trigger time is stored in a multi-dimensional array, called the Memoization Table
- A quantized state is mapped to an entry in the memoization table using a scaling parameter and an offset parameter

Memoization of Trigger Time

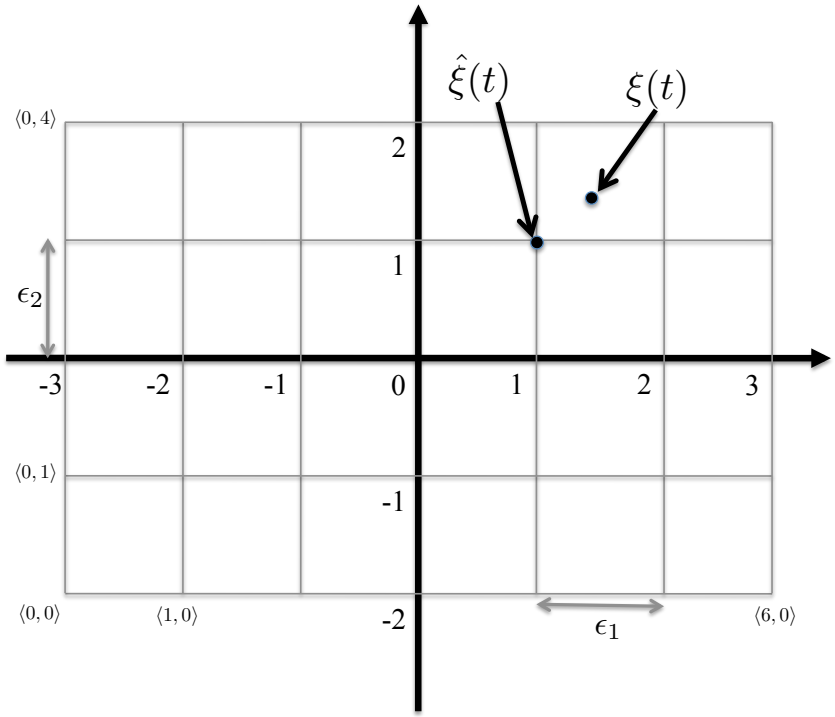


Figure: Memoization region and table

The state $(1.4, 1.3)$ is quantized to $(1, 1)$

The trigger time corresponding to the state $(1, 1)$ is stored in $Memo[4, 3]$

Hybrid Implementation

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

function `findTriggerTime(`ξ`)`
begin
    $\langle s_1 \ldots s_n \rangle$ = `findIndex(`ξ`)`
    if `outsideRange(`$\langle s_1 \ldots s_n \rangle$`)` **then**
        return min($\tau_{\min}$, computeTrigger(ξ))
    else
        if `Memo[`$s_1 \ldots s_n$`] is not found` **then**
            $\hat{\xi}$ = `quantizeState(`ξ`)`
            return min($\tau_{\min}$,
                scheduleTriggerAndMemoize($\hat{\xi}$))
        else
            return `Memo[`$s_1 \ldots s_n$`]`
        end
    end
end

Hybrid Implementation

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

function `findTriggerTime(`ξ`)`
begin
    $\langle s_1 \ldots s_n \rangle$ = `findIndex(`ξ`)`
    if `outsideRange(`$\langle s_1 \ldots s_n \rangle$`)` **then**
        return min($\tau_{\min}$, computeTrigger(ξ))
    else
        if `Memo[`$s_1 \ldots s_n$`] is not found` **then**
            $\hat{\xi}$ = `quantizeState(`ξ`)`
            return min($\tau_{\min}$,
                scheduleTriggerAndMemoize($\hat{\xi}$))
        else
            return `Memo[`$s_1 \ldots s_n$`]`
        end
    end
end

Hybrid Implementation

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

```
function findTriggerTime(ξ)
begin
    ⟨s_1 ... s_n⟩ = findIndex(ξ)
    if outsideRange(⟨s_1 ... s_n⟩) then
        return min(τ_min, computeTrigger(ξ))
    else
        if Memo[s_1 ... s_n] is not found then
            ξ̂ = quantizeState(ξ)
            return min(τ_min,
                    scheduleTriggerAndMemoize(ξ̂))
        else
            return Memo[s_1 ... s_n]
        end
    end
end
```

Hybrid Implementation

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

function `findTriggerTime(`ξ`)`
begin
    $\langle s_1 \ldots s_n \rangle$ = `findIndex(`ξ`)`
    if `outsideRange(`$\langle s_1 \ldots s_n \rangle$`)` **then**
        return min(τ_{min}, computeTrigger(ξ))
    else
        if `Memo[`$s_1 \ldots s_n$`] is not found` **then**
            $\hat{\xi}$ = `quantizeState(`ξ`)`
            return min(τ_{min},
                scheduleTriggerAndMemoize($\hat{\xi}$))
        else
            return `Memo[`$s_1 \ldots s_n$`]`
        end
    end
end

Hybrid Implementation

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

function `findTriggerTime(`ξ`)`
begin
    $\langle s_1 \ldots s_n \rangle$ = `findIndex(`ξ`)`
    if `outsideRange(`$\langle s_1 \ldots s_n \rangle$`)` **then**
        return min($\tau_{\min}$, computeTrigger(ξ))
    else
        if `Memo[`$s_1 \ldots s_n$`] is not found` **then**
            $\hat{\xi}$ = `quantizeState(`ξ`)`
            return min($\tau_{\min}$,
                scheduleTriggerAndMemoize($\hat{\xi}$))
        else
            return `Memo[`$s_1 \ldots s_n$`]`
        end
    end
end

Hybrid Implementation

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

function `findTriggerTime(`ξ`)`
begin
    $\langle s_1 \ldots s_n \rangle$ = `findIndex(`ξ`)`
    if `outsideRange(`$\langle s_1 \ldots s_n \rangle$`)` **then**
        return min($\tau_{\min}$, computeTrigger(ξ))
    else
        if `Memo[`$s_1 \ldots s_n$`] is not found` **then**
            $\hat{\xi}$ = `quantizeState(`ξ`)`
            return min($\tau_{\min}$,
                scheduleTriggerAndMemoize($\hat{\xi}$))
        else
            return `Memo[`$s_1 \ldots s_n$`]`
        end
    end
end

Hybrid Implementation

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

function `findTriggerTime(`ξ`)`
begin
    $\langle s_1 \ldots s_n \rangle$ = `findIndex(`ξ`)`
    if `outsideRange(`$\langle s_1 \ldots s_n \rangle$`)` **then**
        return min($\tau_{\min}$, computeTrigger(ξ))
    else
        if `Memo[`$s_1 \ldots s_n$`] is not found` **then**
            $\hat{\xi}$ = `quantizeState(`ξ`)`
            return min($\tau_{\min}$,
                scheduleTriggerAndMemoize($\hat{\xi}$))
        else
            return `Memo[`$s_1 \ldots s_n$`]`
        end
    end
end

Hybrid Implementation

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

function `findTriggerTime(`ξ`)`
begin
    $\langle s_1 \ldots s_n \rangle$ = `findIndex(`ξ`)`
    if `outsideRange(`$\langle s_1 \ldots s_n \rangle$`)` **then**
        return min($\tau_{\min}$, computeTrigger(ξ))
    else
        if `Memo[`$s_1 \ldots s_n$`] is not found` **then**
            $\hat{\xi}$ = `quantizeState(`ξ`)`
            return min($\tau_{\min}$,
                scheduleTriggerAndMemoize($\hat{\xi}$))
        else
            return `Memo[`$s_1 \ldots s_n$`]`
        end
    end
end

Hybrid Implementation

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

function `findTriggerTime(`ξ`)`
begin
    $\langle s_1 \ldots s_n \rangle$ = `findIndex(`ξ`)`
    if `outsideRange(`$\langle s_1 \ldots s_n \rangle$`)` **then**
        return min($\tau_{\min}$, computeTrigger(ξ))
    else
        if `Memo[`$s_1 \ldots s_n$`] is not found` **then**
            $\hat{\xi}$ = `quantizeState(`ξ`)`
            return min($\tau_{\min}$,
                scheduleTriggerAndMemoize($\hat{\xi}$))
        else
            return `Memo[`$s_1 \ldots s_n$`]`
        end
    end
end

Hybrid Implementation

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

function `findTriggerTime(`ξ`)`
begin
    $\langle s_1 \ldots s_n \rangle$ = `findIndex(`ξ`)`
    if `outsideRange(`$\langle s_1 \ldots s_n \rangle$`)` **then**
        return min($\tau_{\min}$, computeTrigger(ξ))
    else
        if `Memo[`$s_1 \ldots s_n$`] is not found` **then**
            $\hat{\xi}$ = `quantizeState(`ξ`)`
            return min($\tau_{\min}$,
                scheduleTriggerAndMemoize($\hat{\xi}$))
        else
            return `Memo[`$s_1 \ldots s_n$`]`
        end
    end
end

Dynamic Choice of Memoization Region

- A control system may have a number of operating points
- The system can move from one operating point to another during its life cycle
- If the operating point changes from ξ_1 to ξ_2, the memoization region changes from $[\xi_1 - w, \xi_1 + w]$ to $[\xi_2 - w, \xi_2 + w]$
- In case of a change in the operating point
  1. the memoization table is cleared
  2. the offset parameter is reconfigured to reflect the new memoization region

Effect of State Quantization

The state quantization can be modeled as a bounded disturbance added at the input of the plant.

In the presence of bounded disturbance, the controller can render the states of the plant exponentially in a region defined by

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

Given a quantization factor ϵ, we can give a guarantee on the maximum size of V_b.

Alternatively, if a maximum value V_b^{max} is given as specification, we can find out the value of the quantization factor ϵ to ensure that the specification is met.

Self-Triggered Control of Nonlinear Systems

The self-triggered implementation for a nonlinear control systems is given by:

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

Trigger time is computed based on the quantized value $\hat{\xi}$ of ξ.

Find the maximum possible error introduced in the trigger time due to state quantization:

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

The triggering time is computed as

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

Experimental Setup

- We choose PowerPC MPC5xx and Leon2 processors as the two target platforms
- The control computations and trigger time computations are implemented in C programming language, and then cross-compiled for respective processors using GNU-based cross compilation systems
- We use Truetime simulator to implement the control tasks and simulate the systems under different conditions
- The worst-case execution times for control computation and trigger time computation are obtained using the WCET analysis tool aiT

Experimental Setup

- We choose PowerPC MPC5xx and Leon2 processors as the two target platforms
- The control computations and trigger time computations are implemented in C programming language, and then cross-compiled for respective processors using GNU-based cross compilation systems
- We use Truetime simulator to implement the control tasks and simulate the systems under different conditions
- The worst-case execution times for control computation and trigger time computation are obtained using the WCET analysis tool aiT

Experimental Setup

- We choose PowerPC MPC5xx and Leon2 processors as the two target platforms
- The control computations and trigger time computations are implemented in C programming language, and then cross-compiled for respective processors using GNU-based cross compilation systems
- We use Truetime simulator to implement the control tasks and simulate the systems under different conditions
- The worst-case execution times for control computation and trigger time computation are obtained using the WCET analysis tool aiT

Experimental Setup

- We choose PowerPC MPC5xx and Leon2 processors as the two target platforms
- The control computations and trigger time computations are implemented in C programming language, and then cross-compiled for respective processors using GNU-based cross compilation systems
- We use Truetime simulator to implement the control tasks and simulate the systems under different conditions
- The worst-case execution times for control computation and trigger time computation are obtained using the WCET analysis tool aiT

Three implementations

1. Time-triggered implementation

   - Using the sampling period τ_{min}

2. Self-triggered implementation

   - The maximum trigger time τ_{max} is updated to ensure that the computation of the trigger time is guaranteed to be finished before the minimum trigger time τ_{min}

3. Hybrid implementation

   - Without decreasing the maximum trigger time

   - Falls back to the time-triggered implementation using trigger time τ_{min} when the trigger time computation takes more than τ_{min} time.

Simulation Scenarios

1. **Disturbance free**
   The system is free from any external disturbance

2. **Disturbance scenario 1**
   The system is subjected to a periodic external disturbance signal of pulse shape

3. **Disturbance scenario 2**
   The system is subjected to a normally distributed random disturbance signal

Example: Batch Reactor Process

Specification: The state of the system has to converge in a stability region $V_b^{max} = 0.5$

The upper bound on the quantization factor is found to be 0.04346

- We choose the quantization factor to be $\epsilon = 0.04$

We choose the memoization region to be $[-0.5, 0.5]$

Memory required to store the memoization table is $381.47KB$

Example: Batch Reactor Process

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

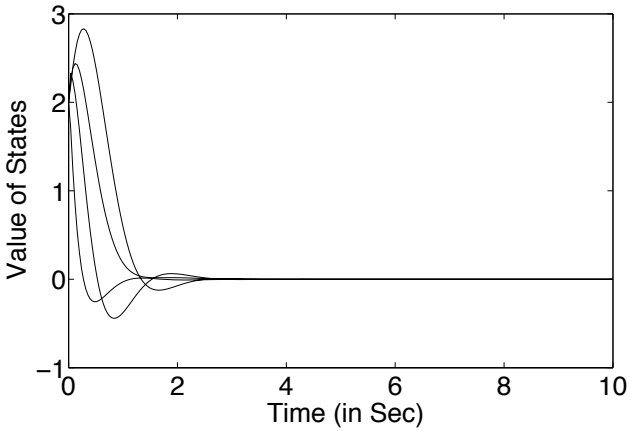
Table: **CPU time** required for different implementations of the controller of the batch reactor process running for 2000s

Example: Batch Reactor Process

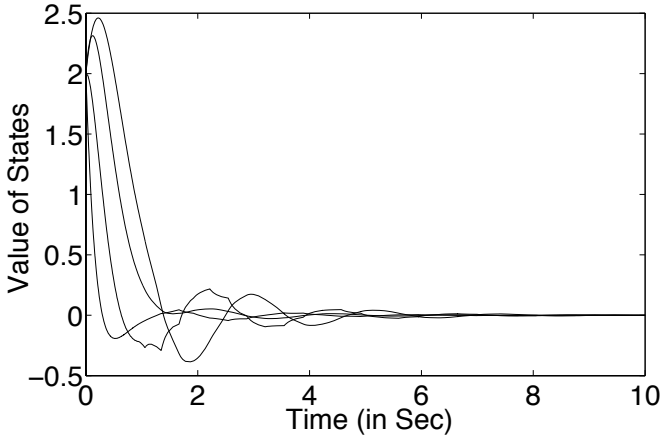
Implementation	Disturbance Free		Disturbance Scenario 1		Disturbance Scenario 2	
	PowerPC	Leon2	PowerPC	Leon2	PowerPC	Leon2
TT	50251	50251	50251	50251	50251	50251
ST	5731	7422	14735	15786	5974	7422
Hybrid	5868	5891	15847	15868	6932	7343

Table: **Number of control computations** for different implementations of the controller of the batch reactor process running for 2000s

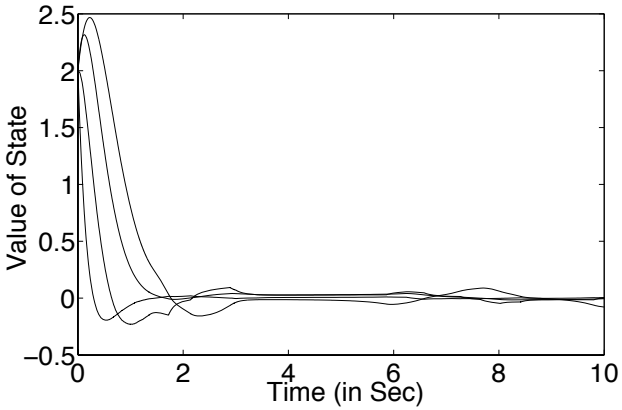
Example: Batch Reactor Process



(a) TT



(b) ST



(c) Hybrid

Figure: Evolution of states from initial state $\langle 2, 2, 2, 2 \rangle$ for no disturbance scenario

Example: Jet Engine Compressor

The formula to compute the trigger time [Anta and Tabuada 2010]:

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

The minimum trigger time $\tau_{min} = \tau^* = 7.63ms$.

The operating point of the jet engine is $\xi_1 = 0, \xi_2 = 0$.

We choose the memoization region to be $\langle \xi_1 \in [0.25, 0.75],\ \xi_2 \in [-1, 1] \rangle$.

Memory used - 256KB

Quantization factor - 0.002

In the memoization region, τ_e^{max} is 6.634ms.

Example: Jet Engine Compressor

Implementation	Disturbance Scenario 1			Disturbance Scenario 2		
	CPU Time		# Control Computations	CPU Time		# Control Computations
	PowerPC	Leon2		PowerPC	Leon2	
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: **CPU time** and **number of control computations** for different implementations of the controller of the jet engine compressor running for 2000s

Conclusions

- Self-triggered implementations have the potential to decrease communication costs and CPU requirements significantly

  - an attractive technique to be used for integrated architectures of cyber-physical systems

- We propose an implementation scheme for self-triggered control

  - uses state quantization and memoization of trigger times in a cache

  - provides guarantees on the region stability of the control systems

- Our implementation scheme is always feasible, and the performance meets the expectation arisen in the literature of self-triggered control systems