Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

RESEARCH ARTICLE

Self-triggered stabilizing controllers for linear continuous-time systems

Fairouz Zobiri*[1,2] | Nacim Meslem[1] | Brigitte Bidegaray-Fesquet[2]

[1]Univ. Grenoble Alpes, CNRS, Grenoble INP, GIPSA-lab, 38000, Grenoble, France
[2]Univ. Grenoble Alpes, CNRS, Grenoble INP, LJK, 38000, Grenoble, France

Correspondence
*Fairouz Zobiri, 700 avenue centrale, 38400, Saint Martin d'Heres. Email: fairouz.zobiri@grenoble-inp.fr

Abstract

Self-triggered control is an improvement on event-triggered control methods. Unlike the latter, self-triggered control does not require monitoring the behavior of the system constantly. Instead, self-triggered algorithms predict the events at which the control law has to be updated before they happen, relying on system model and past information.

In this work, we present a self-triggered version of an event-triggered control method in which events are generated when a pseudo-Lyapunov function (PLF) associated with the system increases up to a certain limit. This approach has been shown to considerably decrease the communications between the controller and the plant, while maintaining system stability. To predict the intersections between the PLF and the upper limit, we use a simple and fast root-finding algorithm. The algorithm mixes the global convergence properties of the bisection and the fast convergence properties of the Newton-Raphson method.

Moreover, to ensure the convergence of the method, the initial iterate of the algorithm is found through a minimization algorithm.

KEYWORDS:
Self-triggered control; Lyapunov function; root-finding algorithm

1 | INTRODUCTION

For a long time, the implementation of continuous-time control tasks on digital hardware has been tied to the so-called Shannon-Nyquist theorem. This condition requires the sampling frequency of the continuous control signal to be relatively high in order to avoid aliasing phenomena. This in turn requires the sensors, controller and actuators to communicate at high speed, tasks that can be straining on communication channels, energy sources and processing units. With the establishment of event-triggered control, researchers and engineers alike realized the possibility of taking samples at a lower pace, provided the samples are non-uniformly distributed over time. Less samples means less interactions between the different blocks of the system, less demand on the communication channels and computation resources.

Event-triggered control, however, only half-solves the problem. Event-triggered control works by updating the control law only when the controlled system violates predefined conditions on its states or output. This implies monitoring the state of the system continuously, thus inducing the high frequency exchanges that we were trying to avoid. Monitoring the event-triggering conditions might also require extra circuitry that is often difficult, if not impossible to build into existing plants.

One way to cancel the need for constant monitoring of the state is to predict in advance the time instants at which the conditions on system behavior are infringed. For this, we use the system's model to predict the evolution of its states. Control strategies in which the times of the control update are known beforehand are the topic of self-triggered control, a variant of event-triggered control. Self-triggered control is most often encountered in the framework of discrete-time systems [2], [3] [4]. In [5], the event-triggering conditions are developed in continuous-time, whereas the next execution time is found by setting a time horizon that is divided in sub-intervals. An event is then determined by checking the event-triggering conditions in each sub-interval. Continuous-time systems have also been studied in [6], where the problem is treated as an optimal control problem, with the next sampling instant as a decision variable. The result is a non-convex quadratic programming problem which is then approximated by a convex problem. In [7] and [8] the authors suggest a self-triggered control method that preserves the $\mathcal{L}_2$ stability of the system in the presence of disturbances. Furthermore, self-triggered control schemes have often been coupled with model predictive control, as both use the model to project the behavior of the system up to some future time [9], [10].

In this work, we design a self-triggered control algorithm for continuous-time linear time-invariant (LTI) systems. The algorithm predicts the times at which the system's behavior will infringe some predefined performance measures. We consider that the system is functioning properly when a pseudo-Lyapunov function (PLF) of its states is below a predefined upper bound. The control law is updated when the PLF reaches this upper bound. Predicting the events analytically is a difficult task, and thus, the self-triggered control algorithm computes an approximation of the event times via a minimization algorithm followed by a root-finding algorithm. The root-finding algorithm detects the intersections between the PLF and the upper limit, but needs to be properly initialized to converge to the right value. To do this, we take advantage of the shape of the PLF between two events; after the control is updated, the PLF decreases for some time, reaches a minimum and then increases again. This local minimum is easily computed via a minimization algorithm, and provides a good initial iterate for the root-finding algorithm.

This paper is divided as follows. In Section 2, we present the problem that we are solving and establish the mathematical formalism necessary to expose our method. Section 3 is divided into two parts. In the first part, we present the minimization algorithm and explain the motivation behind why we need this stage. In the second part, we give the details of the root-finding algorithm. Finally, in Section 4, we validate the method through a numerical example.

2 | PROBLEM FORMULATION

In this section, we first summarize the event-triggered control algorithm introduced in [1] Then we introduce a self-triggered algorithm that predicts the events generated by this event-triggered algorithm.

Consider the following LTI system

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t), \\ x(t_0) &= x_0. \end{aligned} \tag{1}$$

We want to stabilize System (1) with the following control sequence

$$\begin{aligned} u(t_k) &= -Kx(t_k), \\ u(t) &= u(t_k), \qquad\qquad \forall t \in [t_k, t_{k+1}), \end{aligned} \tag{2}$$

where K is the feedback gain, selected such that the matrix $A - BK$ is Hurwitz. The time instants t_k represent the instants at which the control law has to be updated to satisfy predefined stability or performance criteria. The objective of a self-triggered control implementation is to predict the time sequence t_k, $k = 0,\ 1,\ 2, \ldots$ at which the value of the control is updated.

The closed-loop form of System (1) can be written in an augmented form, with augmented state $\xi_k(t) = [x(t),\ e_k(t)]^T \in \mathbb{R}^{2n}$ in $[t_k, t_{k+1})$, with $e_k(t) = x(t) - x(t_k)$

$$\dot{\xi}_k(t) = \begin{bmatrix} A - BK & BK \\ A - BK & BK \end{bmatrix} \xi_k(t) =: \Psi\, \xi_k(t), \tag{3}$$

where 0_n is the vector of zeros in $\mathbb{R}^n$. The system of equations (3) admits a unique solution on the interval $[t_k, t_{k+1})$

$$\xi_k(t) = e^{\Psi(t - t_k)} \xi_k(t_k), \tag{4}$$

where $\xi_k(t_k) = \left[\, x(t_k)\ 0_n^T \,\right]^T$.

We define $I_k(t)$ as the indicator function

$$I_k(t) = \begin{cases} 1, & t \in [t_k, t_{k+1}), \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

Then, for all t, the state of the augmented system is given by

$$\xi(t) = \sum_k \xi_k(t)\ I_k(t), \tag{6}$$

with initial state

$$\xi(t_0) = \left[\, x_0\ 0_n^T \,\right]^T =: \xi_0, \tag{7}$$

In what follows, we designate $\xi_k(t)$ as $\xi(t)$ when the two can be distinguished from the context.

Remark 1. When $t \in [t_k, t_{k+1})$, System (1) is written in closed-loop form as $\dot{x}(t) = Ax(t) - BKx(t_k)$, with a solution $x(t) = (e^{A(t-t_k)} - A^{-1}(e^{A(t-t_k)} - I)BK)x(t_k)$, which requires A to be non-singular. For this reason, we have chosen to work with the augmented system (3), which admits a solution for all A and does not exclude any class of systems. The proposed approached is then applicable to all stabilizable systems.

To determine the control sequence, we first need to define the performance criteria that we impose on the system. For this, we associate to the system a positive definite, energy-like function of the state, that we refer to as a pseudo-Lyapunov function or PLF and which takes the following form

$$V(\xi(t)) = \xi(t)^T \begin{bmatrix} P & 0_{n\times n} \\ 0_{n\times n} & 0_{n\times n} \end{bmatrix} \xi(t) \equiv \xi(t)^T\ \mathcal{P}\ \xi(t), \tag{8}$$

where $0_{n\times n}$ is the $n \times n$ matrix of zeros, and P is a positive definite matrix that satisfies the following inequality

$$(A - BK)^T P + P(A - BK) \leq -\lambda P, \tag{9}$$

where $\lambda > 0$.

For the control sequence given by Equation (2) to stabilize the system, the PLF associated with the system has to decrease along the trajectories of the system. In this work, however, we relax this condition and only require from the PLF to remain upper bounded by a user-defined strictly decreasing threshold. Let the function $W(t)$ be such an upper bound, then, the PLF has to satisfy

$$V(\xi(t)) \leq W(t). \tag{10}$$

The upper bound $W(t)$ has to satisfy a few conditions. It has to be positive, strictly decreasing in time, and to ultimately tend toward zero. One suitable candidate is the exponentially decaying function of the form

$$W(t) = W_0 e^{-\alpha(t-t_0)}, \tag{11}$$

where $W_0 \geq V(\xi(t_0))$ and $\alpha > 0$. The behaviors of $V(\xi(t))$ and $W(t)$ are depicted in Figure 1.

Furthermore, since we want to drive the system trajectory to equilibrium as fast as possible, and since the evolution of $V(\xi(t))$ is determined by the evolution of $W(t)$, we want $W(t)$ to decay to zero as fast as possible as well. The fastest possible rate of change of $W(t)$ is the largest scalar λ, that can be achieved from Inequality (9), as shown in [1] The largest possible value of λ is the solution of the following generalized eigenvalue problem,

$$\begin{aligned} &\text{maximize} \quad \lambda \\ &\text{subject to} \quad (A - BK)^T P + P(A - BK) \leq -\lambda P, \quad P > 0. \end{aligned} \tag{12}$$

Let $\lambda_{\max}$ denote the solution of Problem (12). The rate of decay of $W(t)$ can be chosen as $0 < \alpha < \lambda_{\max}$.

Then, we can define the time instants t_k as

$$t_{k+1} = \inf\{t > t_k \mid V(\xi(t)) = W(t)\}. \tag{13}$$

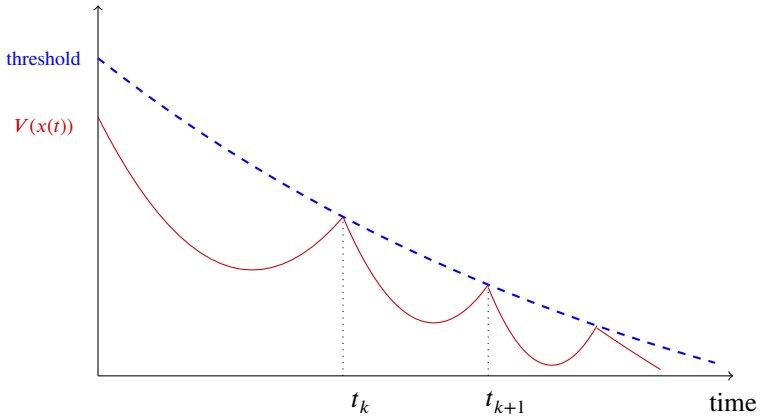


FIGURE 1 The pseudo-Lyapunov function and the upper limit.

with $t_0 = 0$.

In the next section, we detail the procedure used to predict the lower bounds of the entries of the time sequence t_1, t_2, ..., knowing t_0.

3 | SELF-TRIGGERED ALGORITHM

Let $Z(t)$ denote the difference $W(t) - V(\xi(t))$. From Equation (13), to determine t_{k+1}, it suffices to determine the successive time instants at which the following equation is verified

$$Z(t) = 0. \tag{14}$$

Equation (14) depends on time and implicitly on the state $\xi(t)$ which depends on time through a transition matrix as seen from Equation (4). This configuration renders Equation (14) extremely difficult, if not impossible, to solve analytically. For this reason, we propose a numerical solution to Equation (14), where the instant t_{k+1} is computed through a root-finding algorithm.

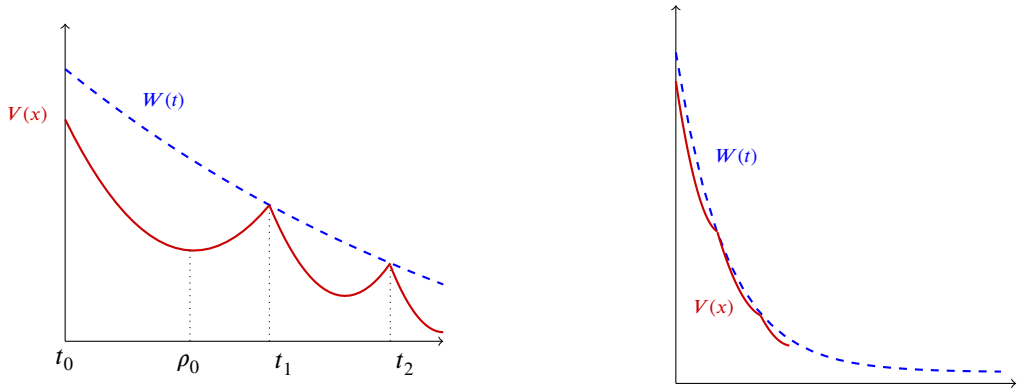
A numerical scheme needs an initial value, and our first guess would be to initialize the root-finding algorithm at instant t_k in order to predict the instant t_{k+1}. However, the instant t_k is itself a root, and as a result, the algorithm fails to converge to t_{k+1} and finds t_k as a solution again. Therefore, we have to initialize our algorithm at a later time instant. Let ρ_k denote the first time instant at which the PLF reaches a local minimum after the time t_k. The instant ρ_k is a good candidate for an initial value, and in what follows, we further justify its use in the root-finding algorithm.

To do this, we classify the evolution of the PLF between two triggering instants, t_k and t_{k+1}, into two categories. The first case, shown in Figure 2a, the minimum of the PLF occurs in between two consecutive triggering instants so that $t_k < \rho_k < t_{k+1}$. In this case, we can see that it is better to initialize our algorithm at time ρ_k, which when combined with the global properties of the bisection method, avoids a convergence toward the time t_k. In the second case, the PLF intersects with the threshold before reaching a local minimum (see Figure 2b). In this case the instant ρ_k offers an upper bound on t_{k+1} from which we can work our way backwards to recover the instant t_{k+1}.

Therefore, we need to precede the root-finding algorithm by a minimization stage, aimed at identifying the time instants at which $V(\xi(t))$ reaches a local minimum.

3.1 | Minimization Stage

Once again, the complexity of the problem makes it impossible to synthesize a closed form analytical solution, and we suggest a numerical solution instead. The minimization algorithm is a modified Newton algorithm that uses t_k as an initial guess to



(a) Case $\rho_k \leq t_{k+1}$. **(b)** Case $\rho_k > t_{k+1}$.

FIGURE 2 Shape of the PLF for different choices of α.

locate the minimum of $V(\xi(t))$ for $t > t_k$.

At each iteration, we compute the Newton step denoted by $\Delta\rho$. Let $\nabla_t V$ and $\nabla_t^2 V$ denote the first and second time derivatives of $V(\xi(t))$. Then, the Newton step is computed as

$$\Delta\rho = \frac{-\nabla_t V}{|\nabla_t^2 V|}. \tag{15}$$

The expressions of $\nabla_t V$ and $\nabla_t^2 V$ are given by

$$\nabla_t V = \xi(t)^T \begin{bmatrix} M & L \\ L^T & 0_{n\times n} \end{bmatrix} \xi(t), \tag{16}$$

$$\nabla_t^2 V = \xi(t)^T \begin{bmatrix} \Lambda & \Gamma \\ \Gamma^T & \gamma \end{bmatrix} \xi(t), \tag{17}$$

where $\xi(t)$ is given by Equation (4) and

$$\begin{aligned} M &= (A - BK)^T P + P(A - BK), \\ L &= PBK, \\ \Lambda &= (A - BK)^T M + M(A - BK) + (A - BK)^T L^T + L(A - BK), \\ \Gamma &= (A - BK)^T L + MBK + LBK, \\ \gamma &= L^T BK + K^T B^T L. \end{aligned}$$

The minimization procedure is given in Algorithm 1. The current iterate is denoted as ρ while the Newton step is represented by $\Delta\rho$. The number of iterations is bounded by the parameter $MaxIter$ for safety, in case the algorithm fails to converge. The procedure starts by computing a Newton step as given by Equation (15). Then, a line search is performed to scale the Newton step. The Newton step is scaled such that the function V decreases enough in the search direction. This step is needed because Newton's method for minimization is an algorithm that computes the roots of the first derivative of the function to be minimized. In our case, we have observed that the first derivative may contain an extremum near the root. Taking the tangent of $\nabla_t V$ at these points yields unreasonable Newton steps [11] that need to be damped. For this reason, this method is sometimes referred to as the damped Newton's method [12]. Once the scaling factor is found, the damped Newton step is taken and a new iterate is found.

Lines 5 through 8 of Algorithm 1 correspond to a backtracking line search. The line search works as follows: a Taylor series approximation of $V(\xi(t))$ is computed, then the line search variable is decreased until a suitable reduction in $V(\xi(t))$ is achieved. The parameter κ_1 indicates the percentage by which $V(\xi(t))$ has to decrease along the search direction. The final value of s is the quantity by which the Newton step is scaled, and β is the fraction by which s is decreased in each line search iteration.

Algorithm 1 terminates when the change in ρ from one iteration to the next becomes negligible. The algorithm's convergence can be very fast, first, because many time consuming operations can be carried out offline. This is the case for matrices M, L, Γ,

Algorithm 1 Minimization Algorithm

function MINIMIZATION(t_k)
  $\rho \leftarrow t_k$
  while $iter \leq MaxIter$ **do**
    $\Delta\rho \leftarrow -\nabla_t V / |\nabla_t^2 V|$
    $s \leftarrow 1$
    while $V(\xi(\rho + s\Delta\rho)) - V(\xi(\rho)) \geq \kappa_1 \, \nabla_t V \, s \, \Delta\rho$ **do**
      $s \leftarrow \beta s, \quad \beta \in (0,1), \kappa_1 \in (0, 0.5)$
    end while
    $tmp \leftarrow \rho$
    $\rho \leftarrow \rho + s\Delta\rho$
    if $|tmp - \rho| < tol$ **then**
      return $\rho_k = \rho$
    end if
    $iter++$
  end while
  return ρ_k
end function

Λ and γ. Even the introduction of a backtracking line search, which is usually a time consuming procedure, does not slow down the algorithm. This is due to the fact that the line search is only performed when we are far from the minimizer, but becomes unnecessary as we approach the minimal value. Therefore, we noticed through our experiments that the algorithm's execution time is negligible compared to the length of the interval $t_{k+1} - t_k$.

Remark 2. In the case of one-dimensional systems, the times at which the local minima of $V(\xi)$ occur can be found analytically. The analytical expression for finding ρ_k and its derivation are given in the Appendix.
When tested on numerical examples, the analytical expression and the numerical approach return the same time sequence.

3.2 | Root-finding Algorithm

Since we want our root-finding algorithm to be both fast and precise, we select an algorithm that combines Newton's method and the bisection method. The bisection method is a globally convergent method that acts as a safeguard against failures of the algorithm when we are far from the root. Newton's algorithm, on the other hand, has a quadratic convergence rate near the root and is used to speed up the algorithm.

To be able to use the bisection, we need to locate the root within an interval, that we denote $[t_{\min}, t_{\max}]$. This is a simple enough task once we know the time instant ρ_k. As explained earlier, t_{k+1} can occur either before or after the time instant ρ_k. Either case is identified by computing $Z(\rho_k)$; if $Z(\rho_k) > 0$, then $t_{k+1} > \rho_k$, whereas if $Z(\rho_k) < 0$, $t_{k+1} < \rho_k$. We then define two time instants t_1 and t_2, we set $t_1 = \rho_k$ and we follow the appropriate procedure

- Case $t_{k+1} > \rho_k$:
  We pick a parameter $\theta > 0$. We suggest to scale the value of θ on the time lapse $\rho_k - t_k$. The scaling factor κ_2 is chosen between 0 and 0.5, depending on how crude we want the search to be, resulting in $\theta = \kappa_2(\rho_k - t_k)$. Then, starting from $t_2 = t_1 + \theta$, we keep increasing t_2 by a value θ until $Z(t_2) < 0$. This procedure is depicted in Figure 3a. Finally, we find $t_{\min} = t_1$ and $t_{\max} = t_2$.

- Case $t_{k+1} < \rho_k$:
  In this case, we pick $\theta = -\kappa_2(\rho_k - t_k)$. Starting from $t_2 = t_1 + \theta$, and while $Z(t_2) < 0$, θ is decreased by a factor of 2 and t_2 is decreased by a value θ. This procedure is depicted in Figure 3b. We keep dividing θ by 2 to avoid the situation $t_2 < t_k$ when the search is too crude. Then, we set $t_{\min} = t_2$ and $t_{\max} = t_1$.

The pre-processing stage is synthesized in Algorithm 2.

(a) Case $\rho_k < t_{k+1}$ **(b)** Case $\rho_k > t_{k+1}$

FIGURE 3 Locating the root inside an interval.

Algorithm 2 Interval Finding

function PRE-PROCESSING(t_k)
  $t_1 \leftarrow \rho_k$
  if $Z(t_1) < 0$ **then**
    $\theta \leftarrow -\kappa_2(\rho_k - t_k),\ 0 < \kappa_2 \leq 0.5$
  else
    $\theta \leftarrow \kappa_2(\rho_k - t_k)$
  end if
  $t_2 \leftarrow \rho_k + \theta$
  while $Z(t_1)Z(t_2) \geq 0$ **do**
    $t_2 \leftarrow t_2 + \theta$
    if $t_2 \leq t_k$ **then**
      $t_2 \leftarrow t_2 - \theta,\ \theta \leftarrow \theta/2$
      $t_2 \leftarrow t_2 + \theta$
    end if
  end while
  $t_{\max} \leftarrow \max(t_1, t_2)$
  $t_{\min} \leftarrow t_{\max} - |\theta|$
  return $t_{\min}, t_{\max}$
end function

The root-finding algorithm can only find approximate event times t_k, and so at $t = t_k$, we only have $W(t_k) \approx V(\xi(t_k))$. For this reason, to ensure the convergence of the algorithm, at $t = t_k$, we make the correction $W(t_k) = V(\xi(t_k))$. If we let $W(t_k) = W_k$, the expression of $W(t)$ on the interval $[t_k, t_{k+1})$ becomes

$$W(t) = W_k e^{-\alpha(t-t_k)}, \tag{18}$$

The function $Z(t)$, on $[t_k, t_{k+1})$, is then given by the

$$Z(t) = W_k e^{-\alpha(t-t_k)} - \xi(t)^T\ \mathcal{P}\ \xi(t), \tag{19}$$

where $\xi(t)$ is given by equation (4).
The first derivative with respect to time, along the trajectories of $\xi(t)$ is

$$\frac{dZ(t)}{dt} = -W_k \alpha e^{-\alpha(t-t_k)} - \nabla_t V. \tag{20}$$

To decide whether to take a Newton step or a bisection step, we first compute an iterate with Newton's method. If the new iterate is located within the previously identified interval $[t_{\min}, t_{\max}]$, it is accepted. Otherwise, the Newton iterate is rejected and instead a bisection iterate (the mid-point of the search interval) is computed. The interval $[t_{\min}, t_{\max}]$ is then updated.
Algorithm 3 describes the root-finding procedure in details. It is a slightly modified version of the hybrid Newton-bisection algorithm found in [13]. To make the notations shorter, from now on we refer to $dZ(t)/dt$ as $\nabla_t Z(t)$.

The algorithm starts by making sure that neither $t_{\min}$ nor $t_{\max}$ are the root, the procedure is exited if it is the case. Checking whether $t_{\min}$ is a root or not should be performed before the pre-processing, but for the sake of separation, we include it in the

Algorithm 3 Root-Finding Algorithm

1: **function** NEWTON-BISECTION($t_{\min}, t_{\max}$)
2: **if** $Z(t_{\min}) == 0$ **then**
3: return $t_{\min}$
4: **end if**
5: **if** $Z(t_{\max}) == 0$ **then**
6: return $t_{\max}$
7: **end if**
8: $t \leftarrow (t_{\min} + t_{\max})/2$
9: $\Delta t \leftarrow t_{\max} - t_{\min}, \quad \Delta t_{\text{old}} \leftarrow \Delta t$
10: compute $Z(t)$, $\nabla_t Z(t)$
11: **while** $iter \leq MaxIter$ **do**
12: $step \leftarrow \frac{Z(t)}{\nabla_t Z(t)}$
13: **if** $t_{\min} \geq t - step$ *or* $t_{\max} \leq t - step$ *or* $\frac{|\Delta t_{\text{old}}|}{2} < |step|$ **then**
14: $\Delta t_{\text{old}} \leftarrow \Delta t$
15: $\Delta t \leftarrow (t_{\max} - t_{\min})/2$
16: $t \leftarrow t_{\min} + \Delta t$
17: **else**
18: $\Delta t_{\text{old}} \leftarrow \Delta t$
19: $\Delta t \leftarrow step$
20: $t \leftarrow t - \Delta t$
21: **end if**
22: **if** $|\Delta t| < tol_2$ **then** return t
23: **end if**
24: **if** $Z(t) > 0$ **then** $t_{\min} \leftarrow t$
25: **else** $t_{\max} \leftarrow t$
26: **end if**
27: **end while**
28: return t_{k+1}
29: **end function**

root-finding algorithm at this stage. The iterate t is initialized as the midpoint of the interval $[t_{\min}, t_{\max}]$.

The variables Δt and Δt_{old} store the current and the former step lengths, respectively. We compute $Z(t)$ and $\nabla_t Z(t)$ in order to compute the Newton step. The condition on line 13 of Algorithm 3 decides whether a Newton step is taken or rejected. If by taking the Newton step we exceed $t_{\max}$ or regress below $t_{\min}$ or if Newton's algorithm is too slow, the Newton step is rejected, and a bisection step is taken instead. Lines 14 to 16 represent a bisection step, whereas lines 18 to 20 represent the case where the Newton step is taken.

After the new iterate is computed, we evaluate $Z(t)$ at that point. If $Z(t)$ is positive, the new iterate is located before the root, and it becomes $t_{\min}$. Otherwise, the current iterate become $t_{\max}$. The algorithm terminates when the change in t between two consecutive iterates is too small, i.e. when the step length becomes smaller than a tolerance tol_2.

3.3 | Summary of the Self-Triggered Algorithm

The three steps of the self-triggered algorithm, described separately so far, are grouped in the order in which they are called, in Algorithm 4.

The main contribution of this paper about the design of self-triggered stabilizing controllers is introduced in the following proposition

Algorithm 4 Self-Triggered Algorithm

```
procedure SELF-TRIGGERED
    ρ_k = MINIMIZATION (t_k)
    if Z(ρ_k) == 0 then
        t_{k+1} = ρ_k
    end if
    [t_min, t_max] = PRE-PROCESSING (ρ_k)
    t_{k+1} = NEWTON-BISECTION(t_min, t_max)
end procedure
```

Proposition 1. Let $\lambda_{\max}$ be the solution to problem (12). If we choose α between 0 and $|\lambda_{\max}|$, Algorithm 4 provides update instants t_k for the control law $u(t)$, given by Equation (2), such that System (1) is asymptotically stable.

The proof for Proposition 1 is given in details in [1]. In what follows, a brief summary of the proof is given. Since $W(t)$ decreases exponentially toward zero, we need to show that $V(\xi(t)) < W(t)$ for all t (or equivalently that $Z(t) > 0$ for all t) to prove that System (1) is asymptotically stable. We know that in the interval (t_{k-1}, t_k), $k \geq 1$, $Z(t) > 0$. And since Algorithm 4 predicts the time t_k when $Z(t)$ approaches zero from above, at $t = t_k$, the control law $u(t)$ is updated so that $Z(t)$ becomes strictly positive again. Therefore, $Z(t) > 0$ for all t.

4 | NUMERICAL SIMULATION

Consider the following third order LTI system [14],

$$\dot{x}(t) = \begin{bmatrix} 1 & 1 & 0 \\ -2 & 0 & 4 \\ 5 & 4 & -7 \end{bmatrix} x(t) + \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} u(t),$$

with initial state $x_0 = [-2\ 3\ 5]^T$.
The system is unstable with poles at -8.58, 0.58, 2.00. We stabilize the system with a state-feedback control law with feedback gain

$$K = \begin{bmatrix} 8.38 & 26.36 & 10.38 \end{bmatrix},$$

that places the poles at $-1.14 \pm 1.35i$, -5.71. Solving the generalized eigenvalue problem (12) yields $\lambda_{\max} = 2.28$ and

$$P = \begin{bmatrix} 275.7 & 1025.5 & 577.9 \\ 1025.5 & 3840.1 & 2173.5 \\ 577.9 & 2173.5 & 1234.1 \end{bmatrix}. \tag{21}$$

We select $\alpha = 2.18\ \text{s}^{-1}$ and $W_0 = 1.3V(x_0)$.
We simulate the system's operation for 7 s, with a sampling period $T_s = 10^{-3}$.

The values of the parameters required by the minimization algorithm and the root-finding algorithm are given in Table 1.

TABLE 1 Values of the parameters needed in the self-triggered control algorithm

Parameter	Value
$MaxIter$	50
β	0.35
κ_1	0.01
tol_1	10^{-5}
κ_2	0.25
tol_2	10^{-5} at $t = 0$

The tolerance tol_2, at which the root-finding algorithm terminates, is set dynamically. Such a choice is motivated by the exponential decrease of $W(t)$, which tends to zero as time tends to infinity. If tol_2 is constant, at some point, $W(t)$ can decrease below this tolerance, and so does $V(\xi(t))$, leading to a small $Z(t)$ that could be mistaken for the root, when there is actually no intersection. Therefore, we index the value of tol_2 on W_k. As long as $W_k > 1$, $tol_2 = 10^{-5}$ as given, but W=when $W_k < 1$, then tol_2 is decreased according to the following equation

$$tol_2 = 10^{-5-\phi}, \quad \text{with } \phi = \lceil |\log_{10}(W_k)| \rceil.$$

At $t = 0$, we apply the control law $u(t_0) = -Kx_0$ and we compute the instant t_1 using the self-triggered algorithm. The system is then on an open-loop configuration, only maintaining a control value of $u(t_0)$, until the clock signal displays the time t_1. At this point, the operation is repeated.

Figure 4a shows the time evolution of the functions $V(\xi(t))$ and $W(t)$. It shows that $V(\xi(t))$ remains below $W(t)$ at all times, which proves that the algorithm manages to identify correctly the times at which events occur, inducing an update of the control law. Even when the two functions approach zero, the intersections are still detected as shown on Figure 4b, which singles out an event at $t = 6.476$ s and $W(t) = 0.0948$.

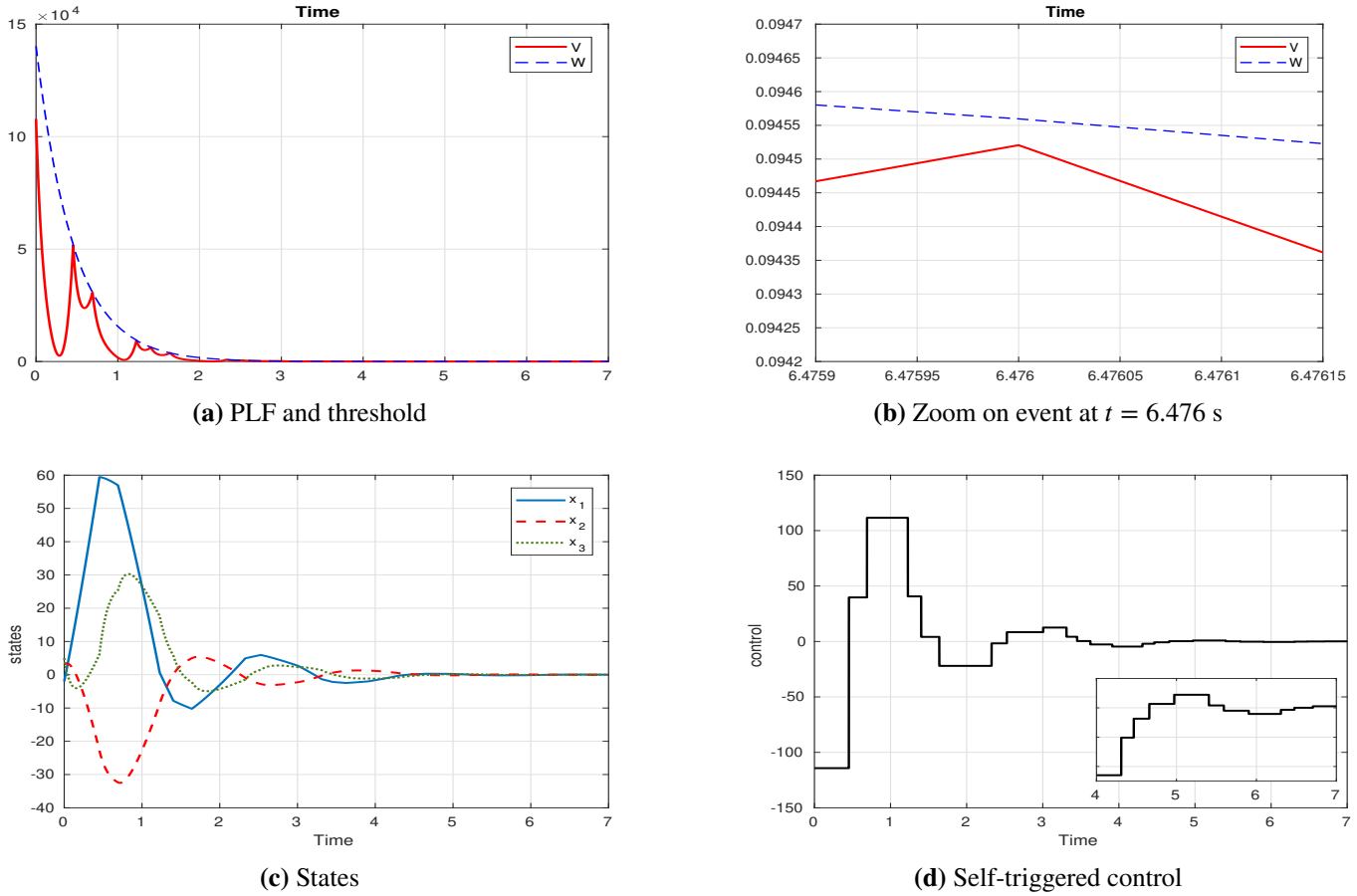
The zoom on the event at $t = 6.476$ s shows that the update of the control law is carried out one time step before the intersection occurs. This is due to the fact that the control can only be updated at multiples of the simulation sampling period T_s. For this reason, when an intersection is predicted somewhere between sampling instants $t = 6.476$ s and $t = 6.477$ s, we update the control law at the earlier instant, $t = 6.476$ s, to prevent the PLF from crossing the threshold. By contrast, in the event-triggered control algorithm on which this approach is based, the event is detected one time step after it occurs. From this point of view, the self-triggered control algorithm represents another improvement on event-triggered control.

The three state variables, shown on Figure 4c, tend to equilibrium and the $\|x(t)\|$ stabilizes below 0.05 within 6.94 s. The stabilizing control law is shown on Figure 4d. This figure shows the uneven distribution of updates in time. Figure 4d also includes a zoom on the control in the time interval [4 s, 7 s], which emphasizes the asynchronous nature of the updates, and which is not visible on the larger figure.

Table 2 lists the first six event times with the corresponding inter-event times $t_k - t_{k+1}$ and running times of the self-triggered control algorithm. We notice that for our experimental conditions, the algorithm's running time is much smaller than the corresponding inter-event time, allowing the online use of the algorithm. Moreover, the running time decreases as we go further in time, the highest running time being the first call of the algorithm, but this call can be made offline. Eventually, the running time settles around 0.002 s. Additionally, matrices M, L, Λ, Γ and γ are computed offline, and thus do not affect running time. Figure 5 further illustrates the disparity between the running times of the algorithm and the inter-event times.

TABLE 2 The first 6 events

Update time	Inter-event time	Running time
0.453	0.453	0.0481
0.691	0.238	0.0081
1.228	0.537	0.0043
1.403	0.175	0.0029
1.641	0.238	0.0089
2.328	0.687	0.0030



(a) PLF and threshold

(b) Zoom on event at $t = 6.476$ s

(c) States

(d) Self-triggered control

FIGURE 4 Simulation results of self-triggered control.

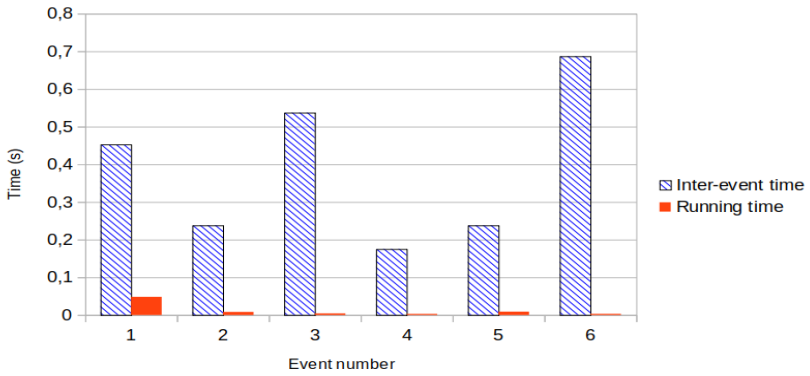


FIGURE 5 The running times of the self-triggered algorithm versus the inter-event times.

5 | CONCLUSION

We presented a self-triggered control algorithm for linear time-invariant systems. The approach approximately predicts the times at which a pseudo-Lyapunov function associated to the system reaches an upper limit, which are the times at which the system ceases to be stable and the control needs to be updated. These time instants are approximated using numerical methods in two stages. In the first stage, a minimization algorithm locates the time ρ_k at which the pseudo-Lyapunov function reaches a minimum value in the interval between two events. In the second stage, a root-finding algorithm initialized at ρ_k approximates

the time of the next event.

The strength of this root-finding method is that it combines a globally convergent method with a locally convergent method. The globally convergent method ensures convergence to the right solution while the locally convergent method speeds up the convergence. Additionally, the minimization stage guarantees that the algorithm is initialized with a value close to the region of attraction of the actual root. The convergence and speed properties of this method make it suitable for both offline and online implementations.

To further validate this approach, the next step would be to apply the self-triggered control algorithm on a real system. This would allow us to test its efficiency against the uncertainties encountered in practical application.
Another perspective would be to extend this method to solve the problem of reference tracking, as this involves, in addition to stabilizing the system, the difficulty of detecting the changes in the reference trajectory.

6 | ACKNOWLEDGMENT

This work has been partially supported by the LabEx PERSYVAL-Lab (ANR-11-61 LABX-0025-01).

□

APPENDIX

A ONE-DIMENSIONAL SYSTEMS

In the case of one-dimensional systems, the local minimum of $V(\xi(t))$ can be found analytically. In what follows, we give a detailed procedure to determine ρ_k analytically. We consider the first order LTI system described as

$$\begin{aligned} \dot{x}(t) &= ax(t) + bu(t), \\ y(t) &= cx(t), \end{aligned} \tag{A1}$$

where $x(t), u(t) \in \mathbb{R}$, and $a, b, c \in \mathbb{R}^*$, $\forall t > 0$.

Let x_k denote $x(t_k)$. The event-triggered control law is given by $u(t) = -Kx_k$ and System (A1) in its closed-loop form is given by

$$\dot{x}(t) = ax(t) - bKx_k, \quad \forall t \in [t_k, t_{k+1}), \tag{A2}$$

Since we assumed that $a \neq 0$, the augmented system described by Equation (3) is not needed for the scalar case. The differential equation (A2) admits a unique solution for $t > t_k$, given by

$$x(t) = \left(\frac{bK}{a} + \left(1 - \frac{bK}{a} \right) e^{a(t-t_k)} \right) x_k. \tag{A3}$$

To System (A1), we associate a Lyapunov-like function of the form

$$V(x(t)) = px(t)^2, \tag{A4}$$

where $p > 0$ is a solution to the Lyapunov inequality

$$2p(a - bK) \leq -q, \tag{A5}$$

where $q > 0$ is a user-defined design parameter.

The minimum of $V(x(t))$ corresponds to

$$0 = \frac{dV(x(t))}{dt} = 2p(ax(t) - bKx_k)x(t). \tag{A6}$$

Equation (A6) admits two solutions, $x(t) = 0$, and $x(t) = bKx_k/a$. However, the solution $x(t) = bKx_k/a$ is impossible as it is equivalent to

$$\left(\frac{bK}{a} + \left(1 - \frac{bK}{a}\right) e^{a(t-t_k)}\right) x_k = \frac{bKx_k}{a},$$
$$e^{a(t-t_k)} \left(1 - \frac{bK}{a}\right) = 0.$$

We know that $e^{a(t-t_k)} \neq 0$ and we cannot choose K such that $bK/a = 1$ or else we would destabilize the system. Therefore, in the scalar case, $dV/dt = 0$, if and only if $x(t) = 0$.

Consequently, the local minima of $V(x(t))$ occur only when $x(t) = 0$ and ρ_k can be directly computed from Equation (A3)

$$\left(\left(1 - \frac{bK}{a}\right) e^{a(\rho_k - t_k)} + \frac{bK}{a}\right) x_k = 0.$$

We know that $x_k \neq 0$, because at $t = t_k$, $V(x_k) = px_k^2 = W(t_k) \neq 0$, hence $x_k \neq 0$. Therefore, the times ρ_k are given by the expression

$$\rho_k = t_k + \frac{1}{a} \log\left(\frac{bK}{bK - a}\right). \tag{A7}$$

We can always take the logarithm of $bK/(bK - a)$ because this is always a positive quantity, as can be seen from the following proof.

- Case $a > 0$:
  The feedback gain is chosen such that $a - bK < 0$. Then, $bK - a > 0$, and $bK > a > 0$. Since the numerator and denominator are both positive, then $bK/(bK - a) > 0$. Moreover, $bK/(bK - a) > 1$, proving that the ρ_k computed by Equation (A7) occurs indeed after t_k.

- Case $a < 0$:
  If the open-loop system is already stable, the objective of the control is certainly to place the pole further to the left. Then, the feedback gain is chosen such that $a - bK < a < 0$. Then, we must have $bK > 0$ and $bK - a > 0$. Consequently, as in the previous case, $bK/(bK - a) > 0$. Even if in this case $bK/(bK - a) < 1$, the ρ_k given by Equation (A7) still occurs after t_k.

Equation (A7) is independent of x_k, indicating that the interval $[t_k, \rho_k]$ has the same length for all k.

References

[1] Zobiri F, Meslem N, Bidégaray-Fesquet B. Event-triggered stabilizing controllers based on an exponentially decreasing threshold. In: Third International Conference on Event-Based Control, Communications, and Signal Processing, Funchal, Portugal. ; 2017.

[2] Durand S, Guerrero-Castellanos JF, Lozano-Leal R. Self-triggered control for the stabilization of linear systems. In: 9th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE). ; 2012.

[3] Velasco M, Martí P, Bini E. Optimal-sampling-inspired Self-Triggered control. In: International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP). ; 2015.

[4] Kishida M. Event-triggered Control with Self-triggered Sampling for Discrete-time Uncertain Systems. *IEEE Transactions on Automatic Control* 2018; 64(3): 1273-1279.

[5] Mazo M, Anta A, Tabuada P. On Self-Triggered Control for Linear Systems: Guarantees and Complexity. In: Proceedings of the European Control Conference. ; 2009; Budapest, Hungary.

[6] Kobayashi K, Hiraishi K. Self-Triggered Optimal Control of Linear Systems Using Convex Quadratic Programming. In: AMC2014-Yokohama. ; 2014; Yokohama, Japan.

[7] Wang X, Lemmon MD. Self-Triggered Feedback Control Systems With Finite-Gain $\mathcal{L}_2$ Stability. *IEEE Transactions on Automatic Control* 2009; 54(3): 452-467.

[8] Wang X, Lemmon MD. Self-triggered feedback systems with state-independent disturbances. In: 2009 American Control Conference. ; 2009; St. Louis, MO, USA.

[9] Kobayashi K, Hiraishi K. Self-triggered model predictive control with delay compensation for networked control systems. In: 38th Annual Conference on IEEE Industrial Electronics Society. ; 2012.

[10] Henriksson E, Quevedo DE, Peters EGW, Sandberg H, Johansson KH. Multiple-loop self-triggered model predictive control for network scheduling and control. *IEEE Transactions on Control Systems Technology* 2015; 23(6): 2167-2181.

[11] Gill PE, Murray W, Wright MH. *Practical optimization*. Elsevier Academic Press . 1986.

[12] Boyd S, Vandenberghe L. *Convex Optimization*. Cambridge university press, New York . 2014.

[13] Press WH, Teukolsky S, Flannery BP, Vetterling WT. *Numerical recipes : the art of scientific computing*. Cambridge University Press. third edition ed. 2007.

[14] Dorf RC, Bishop R. *Modern control systems*. Pearson Education, Inc. . 2014.