

Further Results on Event-Based PID Controller

Sylvain Durand, Nicolas Marchand

▶ **To cite this version:**

Sylvain Durand, Nicolas Marchand. Further Results on Event-Based PID Controller. ECC 2009 - European Control Conference, Aug 2009, Budapest, Hungary. pp.1979-1984. hal-00368535

HAL Id: hal-00368535
https://hal.science/hal-00368535

Submitted on 19 Feb 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Further Results on Event-Based PID Controller

Sylvain Durand and Nicolas Marchand

Abstract— **In this paper, some improvements of the simple event-based PID controller presented by K-E Årzén in [2] are proposed. This controller, contrary to a time-triggered controller which calculates the control signal at each sampling time, calculates the new control signal only when the measurement signal *sufficiently* changes. In the original work of Årzén, a safety maximum period is added forcing the control to be recomputed even if the measurement signal remains unchanged. The contribution of this paper is to propose a scheme to avoid this re-computation. Besides a noticeable reduction of the mean control computation cost, the performance of the closed loop system is also improved.**

I. INTRODUCTION

The classical so-called discrete time framework of controlled systems consists in sampling the system uniformly in the time with some constant sampling period h_{nom} and in computing and updating the control law every time instants $t = kh_{nom}$. This field, denoted the time-triggered case (or the synchronous case in sense that all the signal measurements are synchronous), has been widely investigated [5] even in the case of sampling jitter or measure loss that can be seen as some asynchronicity. However, some works addressed more recently event-based sampling where the sampling intervals are event-triggered (also called asynchronous), for example when the output crosses a certain level. Extending the analogy between Riemann and Lebesgues integral calculation (the first one summing the height at each instant whereas the second sums the instants at all height), the notion of Lebesgues sampling was introduced to denote this sampling scheme: the measures are taken only when variables cross some specific levels by opposition to the Riemann sampling where the measures are taken at specific time instants. Thus in the event-triggered sampling scheme, the term *sampling period* denotes a time interval between two consecutive level crossings of the measure, that is two successive sampling instants, and the sampling periods are hence not equidistant in time anymore.

Event-based notion is taking more and more importance in the signal processing community with now various publications on this subject (see for instance [1] and the references therein). In the control community, very few works have been done. In [3], it is proved that such an approach reduces the number of sampling instants for the same final performance. In [6], it is shown that controlling a Lebesgues sampled system or a continuous time system with quantized measurements and a constant control law over sampling periods are equivalent problems.

Many reasons are motivating the event-triggered systems and in particular because more and more asynchronous systems or systems with asynchronous needs are encountered. Actually, the demand of low power electronic components in all embedded and miniaturized applications encourages companies to develop asynchronous versions of the existing time-triggered components, where a significant power consumption reduction can be achieved by decreasing the samplings and consequently the CPU utilization: about four times less power than its synchronous counterpart for the 80C51 microcontroller of Philips Semiconductors in [12] for example. Moreover, the absence of synchronization in the asynchronous circuits considerably reduces the noises and the electromagnetic emissions by improving the time repartition of the events [11], [10]. Note that the sensors and the actuators based on level crossing events also exist, rendering a complete asynchronous control loop now possible. But the most important contributions come from the real-time control community. Indeed, the real-time synchronous control tasks are often considered as hard tasks in term of time synchronization, requiring some strong real time constraints. Efforts are so carried on the co-design between the controller and the task scheduler in order to soften these constraints. The adopted approach in this field is often either to change dynamically the sampling period related to the load [8], [9] or to use an event-driven control where the events are generated with a mix of level crossings and a maximal sampling period [7], [2].

In this paper we are interested on this maximal sampling period, firstly introduced by Karl-Erik Årzén with his *simple event-based PID controller* [2], which seems to be added for stability reasons in order to fulfill the condition of Nyquist-Shannon sampling theorem: a new control signal is performed when the time elapsed since the last sample exceeds a certain limit. Nevertheless, we propose to remove this safety condition because, thanks to the level detection, the Nyquist-Shannon sampling condition is no more consistent.

The next section recalls both the conventional time-triggered PID structure and the event-triggered PID controller proposed by Årzén. Then, a small discretizetion's improvement is presented and the simulation test benches are described. The main contribution of this paper is developed in section III where several event-based PID algorithms *without safety limit condition* are proposed. These new event-based controllers are finally successfully compared (in terms of performance and CPU need) to the conventional PID and the Årzén's PID controllers.

S. Durand is with NeCS Project-Team, INRIA - GIPSA-lab - CNRS, Grenoble, France, sylvain.durand@inrialpes.fr

N. Marchand is with NeCS Project-Team, INRIA - GIPSA-lab - CNRS, Grenoble, France, nicolas.marchand@gipsa-lab.inpg.fr

II. PID CONTROL

In order to compare our work with the existing controllers we propose to recall the conventional time-based PID controller structure and the event-based PID controller introduced by Årzén [2].

A. Time-Based PID Controller

The textbook PID controller in frequency domain is given as following:

$$U(s) = K\left(E(s) + \frac{1}{T_i s}E(s) + T_d s E(s)\right)$$

This equation can be divided into a proportional, an integral and a derivative parts, i.e. U_p, U_i and U_d respectively, which are then modified to improve performances [4]. First, set point weighting is applied on U_p and U_d for a more flexible structure, giving the PID two dimensions of freedom. Moreover, a low-pass filter is added in the derivative term to avoid problems with high frequency measurement noise.

$$\begin{aligned} U_p(s) &= K\left(\beta Y_{sp}(s) - Y(s)\right) \\ U_i(s) &= \frac{K}{T_i s}E(s) \\ U_d(s) &= \frac{K T_d s}{1 + T_d s/N}\left(\gamma Y_{sp}(s) - Y(s)\right) \end{aligned}$$

A discrete time PID controller is finally obtained by discretizing: the proportional part is straightforward, forward and backward difference approximation is used for the integral part and the derivative part respectively.

The resulting code is:

```
% inputs
ysp = u(1);
y = u(2);
e = ysp - y;

% calculate control signal
up = K*( beta*ysp - y );
ud = Td/(N*hact + Td)*ud
      - K*Td*N/(N*hact + Td)*( y - y_old );
u = up + ui + ud;

% update
ui = ui + K/Ti*hact*e;
y_old = y;
```

B. Årzén's Event-Based PID Controller

The basic setup depicted in [2] consists of two parts: *a time-triggered event detector* used for level crossings and *an event-triggered PID controller* which calculates the control signal. The first part runs with the sampling period h_{nom} (that is the same as for the corresponding conventional time-triggered PID) whereas the second part runs with the sampling interval h_{act} which depends on the requests sent by the event detector when a new control signal has to be calculated. This is required either when the relative measurement crosses a certain level, i.e. when the absolute value of the difference between the measured error of the last sampling and that of the current sampling crosses the limit e_{lim}, or if the maximal sampling period is achieved, i.e. $h_{act} \geq h_{max}$.

The resulting code is:

```
% inputs
ysp = u(1);
y = u(2);
e = ysp - y;

% calculate control signal
hact = hact + hnom;
if abs( e - e_old ) > elim || hact >= hmax
    up = K*( beta*ysp - y );
    ud = Td/(N*hact + Td)*ud
         - K*Td*N/(N*hact + Td)*( y - y_old );
    u = up + ui + ud;

    % update
    ui = ui + K/Ti*hact*e;
    e_old = e;
    y_old = y;
    hact = 0;
end
```

C. Discretization Improvement

Let t_k denotes the current time, t_{k-1} the last time where a control signal was calculated and t_{k+1} the next time where a control signal will be calculated. Furthermore, let $h(t_k)$ denotes the current sampling period, i.e. the interval time between the current sampling and the last one, and $h(t_{k+1})$ the next sampling period, i.e. the interval time between the current sampling and the next one. These notations are shown on Figure 1. Note that the sampling interval previously depicted and used by the event-triggered PID controller is $h_{act} = h(t_k)$.

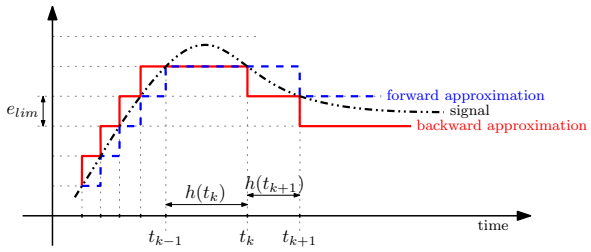


Fig. 1. Forward and backward difference approximation

Currently, the forward difference approximation is used to calculate the integral part of the Årzén's PID controller, which means that the integral part is precalculated during the current interval for the next sampling, i.e. $u_i(t_{k+1}) = u_i(t_k) + K/T_i h(t_{k+1})e(t_k)$. This method can be a good choice for time-triggered controllers (where the sampling interval h is constant) but for event-based controllers the next sampling period $h(t_{k+1})$ varies and so has to be *a priori* known. But this is not possible in practice. One could note that in [2] the proposed algorithm is wrong because h_{act} and $h(t_{k+1})$ are mixed up (whereas $h_{act} = h(t_k)$). However, a solution could be to calculate the current integral part from the previous error by shifting the instant times in the equation, i.e. $u_i(t_k) = u_i(t_{k-1}) + K/T_i h(t_k)e(t_{k-1})$.

Nevertheless, we propose to calculate the integral part with a more recent value of the error by using the backward difference approximation. This leads to calculate the current integral part during the current time with the current sampling period and the current error, as follows:

$$u_i(t_k) = u_i(t_{k-1}) + K/T_i h(t_k)e(t_k) \qquad (1)$$

D. Simulation Results

The simulations are performed on a simple first order system which is described as:

$$H(s) = \frac{G}{1+\tau s} \qquad \text{where} \quad G = 1 \text{ and } \tau = 1$$

This system will be controlled with different controllers: firstly with the conventional time-triggered PI controller, then with the Årzén's event-based PI controller and finally with our proposals, i.e. the event-based PI controllers without safety limit condition (detailed in section III).

The parameter's values of the controllers are obtained by pole placement of the closed-loop system with the time-triggered PI controller. The event-based controllers are then designed with these same values and they will finally try to be as closed as possible of the time-triggered closed-loop shaping. $K = 1.83$ and $T_i = 0.457$, the nominal sampling interval is chosen as $h_{nom} = 0.05s$ and the maximal one as $h_{max} = 0.5s$. The system is simulated for $20s$ and two different test benches are proposed:

- *Test bench 1:* The set point is changed from 0 to 1 at time $1s$ and changed again at time $10s$ to achieve 2.
- *Test bench 2:* The set point is changed from 0 to 1 at time $2s$. A load disturbance is introduced at time $12s$ with an amplitude of 0.1.

The simulation results are then plotted in order to compare two controllers using both test benches. The top plot shows the set point and the measured signals, the bottom plot shows the sampling intervals (i.e. this signal changes each time the controller calculates a new control signal). Note that the sampling intervals for the time-triggered controller are shown only once on Figure 2.

The first simulation results are shown on Figures 2 and 3 where the conventional time-triggered PI controller is compared to the Årzén's event-based one for different value of e_{lim}, i.e. the event detection level. Thus, Figure 2 shows that the Årzén's controller permits to obtain a system response as quick as the time-triggered one by calculating a control signal twice less. However, whereas the event detection level e_{lim} is increased, the results become deteriorated and the measured signal oscillates as shown on Figure 3. These oscillations come from the bad discretization of the integral part (see the subsection II-C for further details). Actually, a mistake is done with the Årzén's algorithm and we propose two solutions to avoid that: ***i)*** calculating the current integral part from the previous error by shifting the instant times in the integral part (still using the forward approximation) or ***ii)*** using the backward approximation. The second solution is applied and Figure 4 shows the difference between the original Årzén's event-based PI controller and the improved Årzén's controller using (1). The improvement is immediate: the results obtained with $e_{lim} = 0.01$ are better than with the original Årzén's controller when $e_{lim} = 0.001$.

In the following section, we will base the new PI architectures on the backward difference approximation and keep the level detection used for the simulations equal to $e_{lim} = 0.01$.

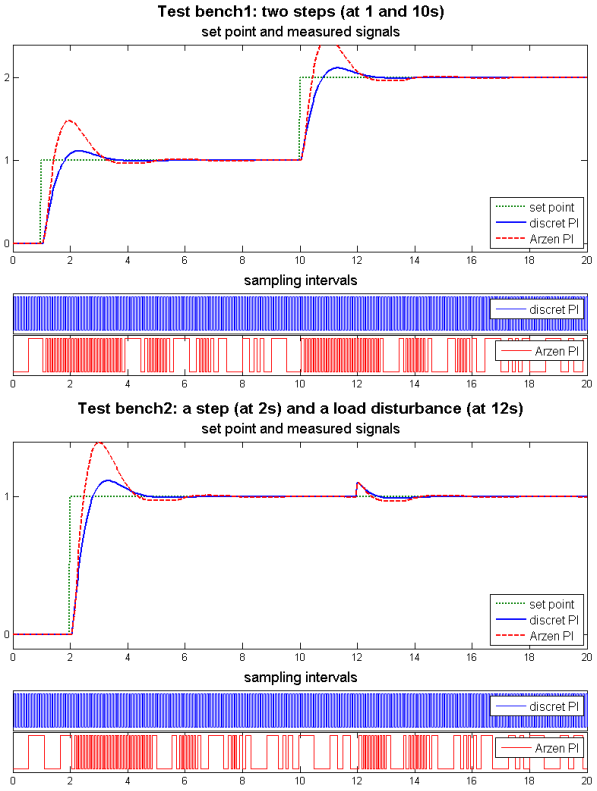


Fig. 2. *The conventional time-triggered PI controller* (400 sampling intervals) vs. *the Årzén's one* with $e_{lim} = 0.001$ (195 intervals for bench1 and 158 for bench2, that is 49 and 39.5% respectively)

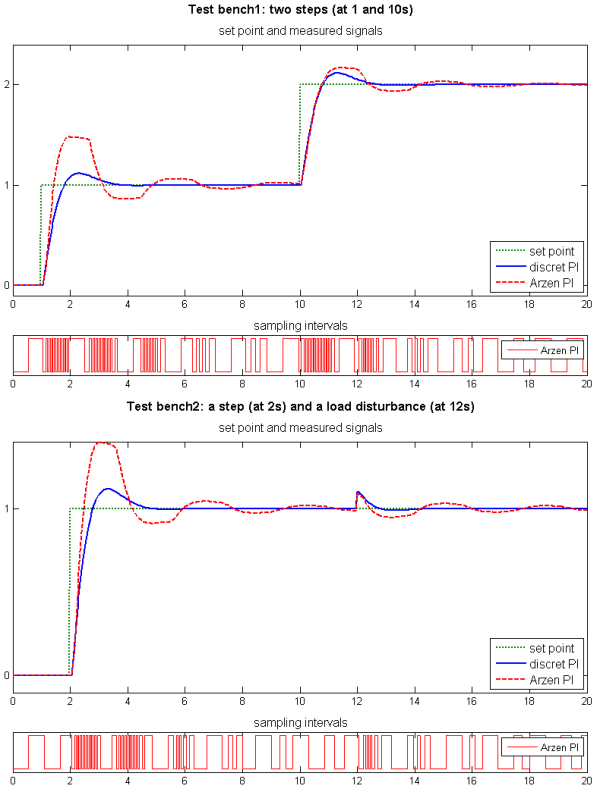


Fig. 3. *The time-triggered PI controller* (400 sampling intervals) vs. *the Årzén's one* with $e_{lim} = 0.01$ (126 intervals for bench1 and 97 for bench2, that is 31.5 and 24.5% respectively)

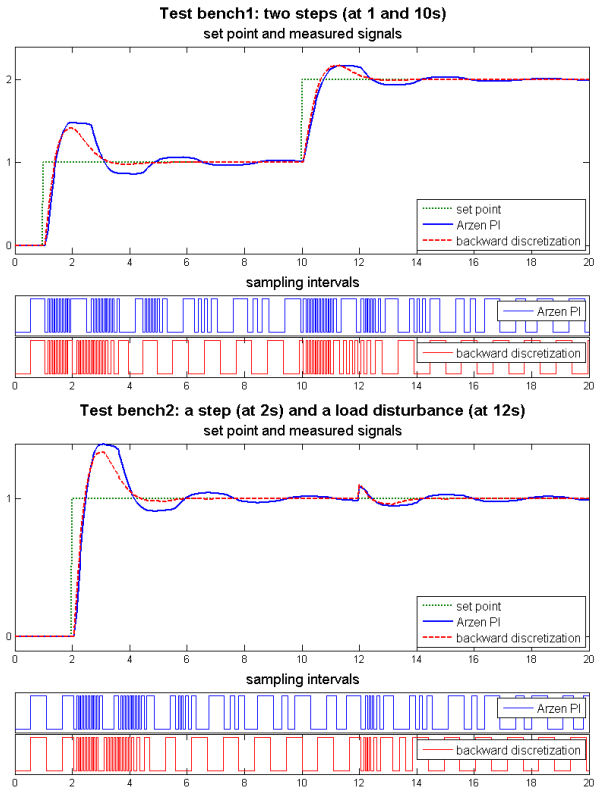


Fig. 4. *The original Årzén's PI controller* (126 sampling intervals for bench1 and 97 for bench2) vs. *the Årzén's PI controller with improved discretization* (101 intervals for bench1 and 82 for bench2, that is 80 and 84.5% respectively) with $e_{lim} = 0.01$

III. EVENT-BASED PID CONTROL WITHOUT SAFETY LIMIT CONDITION

As explained before, our idea is to remove the safety limit condition $h_{hact} \geq h_{max}$ introduced by Årzén, in order to improve and simplify the event-based controller. However, by only doing that the controller will correct the system output too much each time the set point changes after a long steady state interval, which leads to important overshoots as one can see on Figure 5.

Actually, the integral part of the event-based PI controller, i.e. $u_i(t_k) = u_i(t_{k-1}) + K/T_i h_{act} e(t_k)$ from (1), is responsible of this problem because the value of h_{act} becomes huge due to the absence of event. The integral part hence explodes when the set point changes, i.e. the error $e(t_k)$ becomes high. In fact, the time interval between *the last sample before the steady state* and *the first sample of the transient* can be divided into a "real" steady state interval, which is equal to $h_{act} - h_{nom}$, plus the detection time period h_{nom} (because the event detector is time-triggered with the constant sampling period h_{nom}). During the first part the error is very small, i.e. lower than e_{lim} else the steady state is not achieved, and so is the product he, i.e. lower than $(h_{act} - h_{nom})\, e_{lim}$. As regards the second part, when the set point changes the error becomes large but only during the event detection and therefore the product he is lower than $h_{nom} e$. In fact, the product he was over-estimated until now and so we propose to include a more precise value in the code of our proposals. Thus we can write:

$$\begin{aligned} &u_i(t_k) = u_i(t_{k-1}) + K/T_i he \\ &\text{where } he \leq (h_{act} - h_{nom})\, e_{lim} + h_{nom} e(t_k) \end{aligned} \tag{2}$$

Moreover, this inequality which was built for the steady state intervals remains true for the transients, i.e. when $h_{act} = h_{nom}$. Several algorithms *without safety limit condition* and based on this assumption are proposed: the first one where nothing else is done and the others which modify the integral part in order to reduce its impact after a long steady state interval (the depicted approaches are somehow similar to the antiwindup mechanism where the error induced by the saturation has to be compensated):

1) algorithm only without safety limit condition

This algorithm corresponds to the Årzén's one where the safety limit condition $h_{hact} \geq h_{max}$ is removed without doing anything else. Results are shown on Figure 5 where important overshoots appear after the steady state intervals because of the principle described before.

2) algorithm with saturation of the product he

This algorithm consists in bounding the product he after a long steady state interval in order to reduce the overshoots. Thus, when the sampling period becomes too large, i.e. $h_{act} \geq h_{max}$, the product is saturated according (2), i.e. $he = (h_{act} - h_{nom})\, e_{lim} + h_{nom} e$. Results are shown on Figure 6.

3) algorithm with exponential forgetting factor of h_{act}

Another method consists in adding a forgetting factor of the sampling period so that, after a long steady state interval, the h_{act} value is reduced enough to not impact the control signal too much. Thus, the exponential function $h^i_{act} = h_{act} \cdot \exp(h_{nom} - h_{act})$ is chosen to decrease the sampling period impact as the elapsed steady state time increases (with h^i_{act} corresponding to the new sampling interval used in the integral part). This function leads to have a nominal sampling period during the transients, i.e. $h_{act} = h_{nom} \iff h^i_{act} = h_{nom}$, and an exponential decreasing sampling period during the steady state intervals. Results are shown on Figure 7.

4) hybrid algorithm

This algorithm is a mix between the previous two ones. Indeed, the exponential algorithm does not correctly reduce the overshoot in the transient if the steady state interval was not long enough (as one can see at time $1s$ on test bench 1 or at time $2s$ on test bench 2 on Figure 7). Actually, the exponential function used in the algorithm 3 increases first and then decreases, and so if the set point changes before the function decreasing then the sampling period h^i_{act} is still too high. In another way, once the sampling interval exponentially decreases the results are quite good. Furthermore, results obtained with the algorithm 2 are interesting. For this reason, we propose to use the exponential forgetting factor into the algorithm with saturation. Thus, if $h_{act} \geq h_{max}$ the product he is bounded in $\left(h^i_{act} - h_{nom}\right) e_{lim} + h_{nom} e$. Results are shown on Figure 8.

The resulting code for theses different algorithms (where the value of *choice* depends on the chosen algorithm) is:

```
% inputs
ysp = u(1);
y = u(2);
e = ysp - y;

% calculate control signal
hact = hact + hnom;
if ( abs(e-e_old)>elim )

    up = K*( beta*ysp - y );

    switch choice
        % only without safety limit condition
        case 1
            ui = ui + K/Ti*hact*e;

        % saturation of h*e
        case 2
            if hact >= hmax
                he = (hact - hnom)*elim + hnom*e;
            else
                he = hact*e;
            end
            ui = ui + K/Ti*he;

        % exponential forgetting factor of hact
        case 3
            hact_i = hact*exp(hnom-hact);
            ui = ui + K/Ti*hact_i*e;

        % hybrid
        case 4
            if hact >= hmax
                hact_i = hact*exp(hnom-hact);
                he = (hact_i - hnom)*elim + hnom*e;
            else
                he = hact*e;
            end
            ui = ui + K/Ti*he;
    end

    ud = Td/(N*hact + Td)*ud
          - K*Td*N/(N*hact + Td)*( y - y_old );
    u = up + ui + ud;

    % update
    e_old = e;
    y_old = y;
    hact = 0;
end
```

The results obtained with the event-based PI controllers *without safety limit condition* are quite interesting. Indeed, we obtain some measured signals really similar to the conventional time-triggered one, both in term of transients and overshoots. Two algorithms can be highlighted: ***a)*** The hybrid algorithm is the best one: it leads to a control without performance degradation, by calculating a control signal 80% of time less than with the time-triggered controller. If we compare it with the Årzén's controller, the gain on sampling interval number is still about 50% and the performance improvements are very important. A problem could be the computational complexity of the hybrid algorithm in practice because of the exponential function, but a look-up table with precalculated values of the function can replace the online calculation and thereof highly reduce the computational cost. ***b)*** On the other hand, the algorithm with saturation of the product he is quite simple and gives similar results.

IV. CONCLUSIONS AND FUTURE WORKS

This paper proposes some algorithms to improve the simple event-based PID controller presented in [2]. The

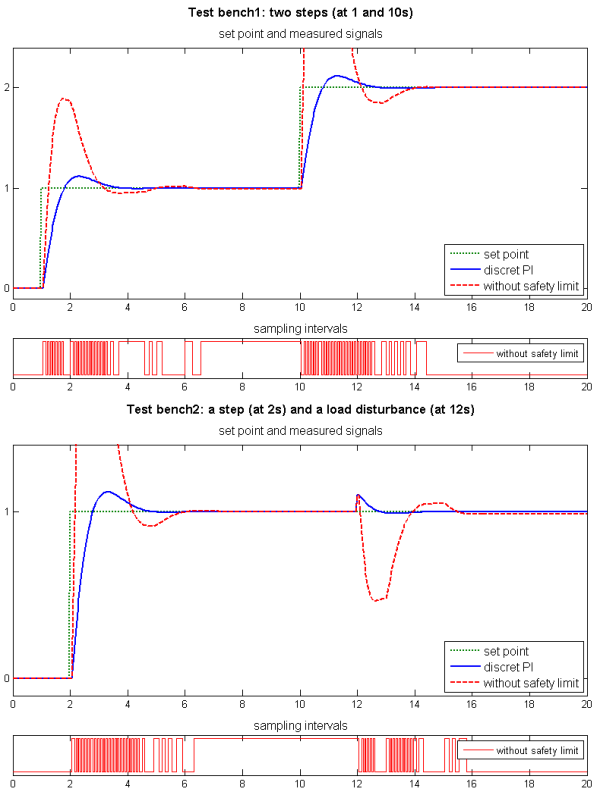


Fig. 5. *The time-triggered PI controller* (400 sampling intervals) vs. *the event-based PI controller without safety limit condition* (112 intervals for bench1 and 96 for bench2, that is 28 and 24% respectively)

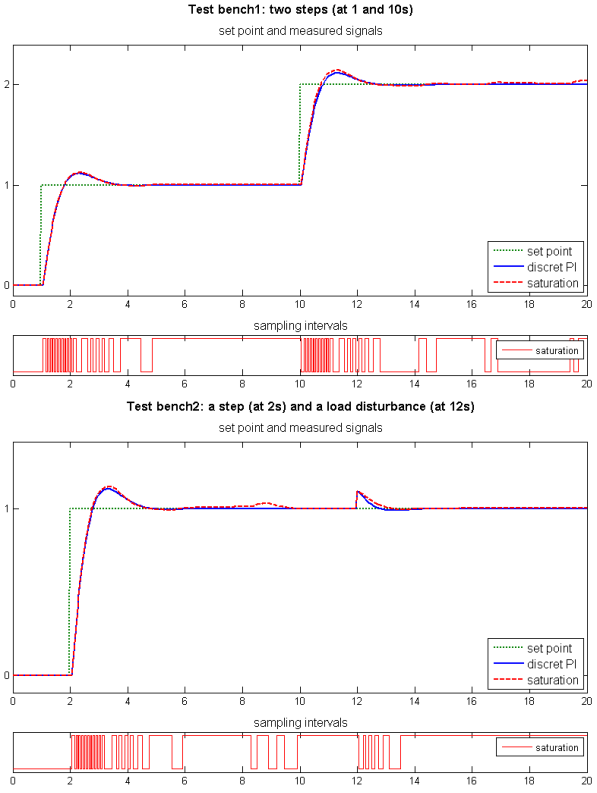


Fig. 6. *The time-triggered PI controller* (400 sampling intervals) vs. *the controller with saturation of the product* he (77 intervals for bench1 and 51 for bench2, that is 19.5 and 13% respectively)

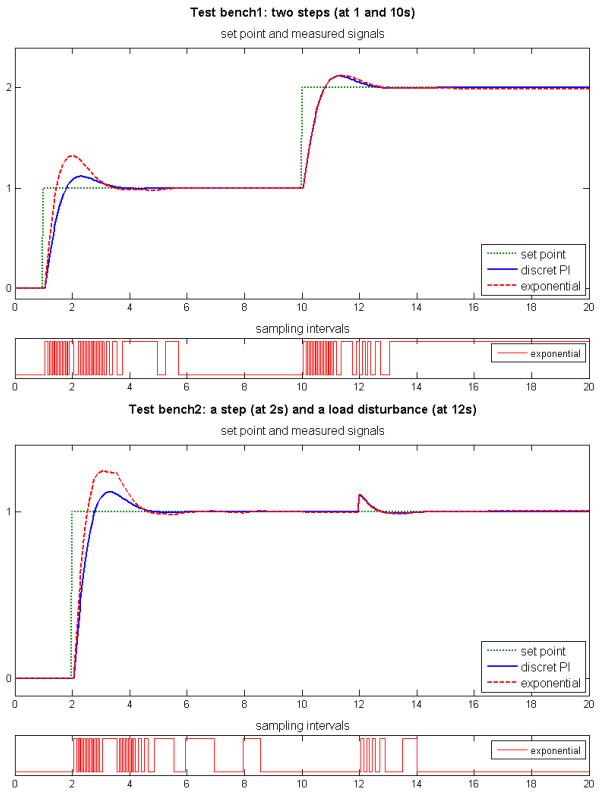


Fig. 7. *The time-triggered PI controller* (400 sampling intervals) vs. *the controller with exponential forgetting factor of* h_{act} (77 intervals for bench1 and 52 for bench2, that is 19.5 and 13% respectively)

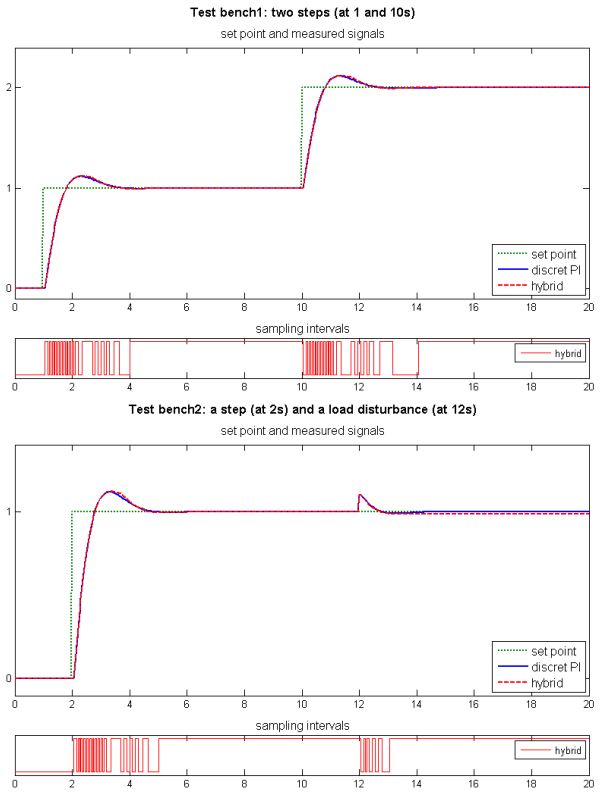


Fig. 8. *The time-triggered PI controller* (400 sampling intervals) vs. *the hybrid controller* (67 intervals for bench1 and 43 for bench2, that is 16.5 and 11% respectively)

improvement comes from the removing of a safety limit condition used for stability reason by the author (maximum sampling period without control update). To compensate this condition, a forgetting factor is imagined in order to reduce the sampling period impact in the integral part of the PID controller. This approach is somehow similar to the antiwindup one where the error induced by the saturation has to be compensated. Based on this idea, event-based PID controllers without safety limit condition are proposed and compared both with the conventional time-triggered controller and the Årzén's event-based controller. Two of the proposed approaches clearly give good performances with a minimum of sampling intervals.

Next steps in this research is naturally to test these event-based controllers in practice and develop other event-based methods for more general types of control.

V. ACKNOWLEDGMENTS

This research has been supported by the NeCS Project-Team (INRIA, GIPSA-lab, CNRS) in the FeedNetBack project context. The project aims to close the control loop over wireless networks by applying a co-design framework that allows the integration of communication, control, computation and energy management aspects in a holistic way.

REFERENCES

[1] F. Aeschlimann, E. Allier, L. Fesquet, and M. Renaudin. Asynchronous FIR filters: towards a new digital processing chain. In *Proceedings of the 10th International Symposium on Asynchronous Circuits and Systems*, pages 198–206, 2004.

[2] K.-E. Årzén. A simple event-based PID controller. In *Preprints of the 14th World Congress of IFAC*, Beijing, P.R. China, 1999.

[3] K. Åström and B. Bernhardsson. Comparison of Riemann and Lebesque sampling for first order stochastic systems. In *Proceedings of the 41st IEEE Conference on Decision and Control*, 2002.

[4] K. Åström and T. Hägglund. *PID controllers: theory, design, and tuning, 2nd Edition*. The Instrumentation, Systems, and Automation Society, 1995.

[5] K. Åström and B. Wittenmark. *Computer Controlled Systems, 3rd Edition*. Prentice Hall, 1997.

[6] N. Marchand. Stabilization of Lebesgue sampled systems with bounded controls: the chain of integrators case. In *Proceedings of the 17th IFAC World Congress*, 2008.

[7] J. Sandee, W. Heemels, and P. van den Bosch. Event-driven control as an opportunity in the multidisciplinary development of embedded controllers. In *Proceedings of American Control Conference*, pages 1776–1781, 2005.

[8] O. Sename, D. Simon, and D. Robert. Feedback scheduling for real-time control of systems with communication delays. In *Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation*, volume 2, 2003.

[9] D. Simon, D. Robert, and O. Sename. Robust control/scheduling co-design: application to robot control. In *Proceedings of the IEEE Symposium on Real-Time and Embedded Technology and Applications*, pages 118–127, 2005.

[10] C. Van Berkel, M. Josephs, and S. Nowick. Scanning the technology: Applications of asynchronous circuits. *Proceedings of the IEEE*, 87(2):223–233, 1999.

[11] K. Van Berkel, R. Burgess, J. Kessels, M. Roncken, F. Schalij, and A. Peeters. Asynchronous circuits for low power: a DCC error corrector. *IEEE Design and Test of Computers*, 11(2):22–32, 1994.

[12] H. van Gageldonk, K. van Berkel, A. Peeters, D. Baumann, D. Gloor, and G. Stegmann. An asynchronous low-power 80C51 microcontroller. In *Proceedings of the 4th International Sympsonium on Advanced Research in Asynchronous Circuits and Systems*, pages 96–107, 1998.