Event-Driven Control as an Opportunity in the Multidisciplinary Development of Embedded Controllers[1]

J.H. Sandee[§], W.P.M.H. Heemels[†] and P.P.J. van den Bosch[§]

[§]Technische Universiteit Eindhoven
Dept. of Electrical Engineering, Control Systems Group
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
Email: j.h.sandee@tue.nl, p.p.j.v.d.bosch@tue.nl
[†]Embedded Systems Institute, Email: maurice.heemels@esi.nl

Abstract— **Severe requirements on embedded controllers ask for new approaches that can effectively bridge the existing gap between the disciplines Software and Control Engineering. Event-driven control is presented as an opportunity to create a negotiable environment for the sample frequency, control performance and the usage of processing power. Simulations show that event-driven control can reduce the average processor load without influencing the control performance too much. Some first ideas for analysis and synthesis techniques look promising and future research will focus on the extension of these techniques.**

I. Introduction

Increasing numbers of processes utilize embedded controllers nowadays. Many examples can be found in e.g. cars, mobile phones or copying devices. The complexity of these processes as well as the complexity of the controller hard- and software is increasing fast, while time-to-market has to decrease. To lower the cost price, multiple control algorithms are often implemented on a single processing unit.

An obvious result is the need for design methods for control that take the requirements of the software implementation into account. On the other hand, the reverse is also needed; design methods for software that also deal with the control requirements. Although the need for close cooperation between control engineers and software engineers seems obvious, the gap between control theory and computer science has only become larger over the last couple of years. The main reason for this is that both disciplines are developing new and more difficult methods and theories fast. It is almost impossible to keep up with each others state-of-the-art practice. This article describes an event-driven controller design as an opportunity to reduce the gap between both disciplines.

Little research has been done on this multidisciplinary problem. The need for such theories however, is addressed in various articles (e.g. See [2] and [10] and the references therein). In this context the term 'implementation aware control' is also well known [8]. The work in this paper forms a contribution to this challenging field of research.

Control algorithms are based on many assumptions. In practice, many of those assumptions cannot be satisfied completely in software. For example, standard control algorithms are based on a constant sample time for both measurement and actuation. This eases the analysis a lot, but it is often hard to implement in software, as it heavily constrains the scheduling of tasks in software. A second assumption is that delays are assumed to be constant and known or even non-existing. When implementing a control algorithm, all calculations take time, so e.g. a delay between measurement and actuation will be present. This delay will hardly ever be constant due to other tasks interrupting the control algorithm or simply by a small change in the required calculations, like e.g. activated saturation. This problem can be overcome by introducing a constant delay between measurement and actuation, e.g. by means of a hardware buffer. A drawback is that this delay needs to be as large as the biggest possible delay. Another problem where software engineers often run into at the implementation is that complex algorithms require too much resources, like processing power. A direct result is that the allowed sample time is limited and therefore not only dependent on the bandwidth of the controlled system. This assumption is often made by the control engineer, disregarding the limitations of the implementation.

One of the possible solutions to take better care of the described problems is event-driven control. In contradiction to time-driven control, these methods are not based upon a constant sample time. Already in 1962, the need for such methods was addressed [4], but few research has been spent in this subject since. Besides being able to vary the sample time each sample, it also enables an easier way to deal with (known) delays by using time-stamping [7]. The benefits of an event-driven controller over a time-driven one are mainly influenced by the way in which the events are generated. One possible choice is to generate less events at times that the process does not have to be controlled very accurately. Another choice could be to shortly delay an event generation when the processor load is very high. In this case, the

[1]This work has been carried out as part of the Boderc project under the responsibility of the Embedded Systems Institute. This project is partially supported by the Netherlands Ministry of Economic Affairs under the Senter TS program.

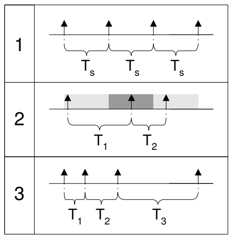


Fig. 1. 1 - synchronous, 2 - semi-synchronous, 3 - asynchronous.

control algorithm is directly coupled to the scheduler of the operating system [3].

Event-driven controllers are becoming increasingly commonplace, particularly for distributed real-time sensing and control. For example, for sensors in sensor networks, TinyOS, an event-based operating system, is rapidly emerging as the operating system of choice. Characteristic of applications running on an event-based operating system is that state variables will typically be updated asynchronously in time, e.g., when an event of interest is detected, of because of delay in computation and/or communication.

This paper presents an event-driven controller as a means to make a compromise between processor load and control performance. Also some first promising ideas ideas for analysis and synthesis techniques are presented.

The paper is outlined as follows: Section II gives a classification needed to explain event-driven control. This section is followed by the description of an event-driven PID controller. In section IV results of various simulations with this event-driven controller are described. Section V describes some preliminary analysis techniques and finally, the conclusions are given.

II. Classification of real-time control

Before discussing how an event-driven controller should be implemented and what could be gained with this type of control algorithms, a classification of possible controllers is given. Three classes are defined for the way in which actions can occur in time. Considering a control algorithm, these actions can be for instance taking measurements or updating the actuator signal. The three classes are displayed in figure 1.

These three classes, respectively called synchronous, semi-synchronous and asynchronous, are defined as follows:

- Actions occur synchronous in time if the length of the time interval in between each two successive actions is the same.
- Actions occur semi-synchronous in time if there exists a partitioning of the time scale into periods with equal lengths, so that each period contains exactly one action.
- Actions occur asynchronous in time otherwise.

With the above given definitions it is now possible to classify all possible control algorithms in the 3x3 matrix, depicted in figure 2. In rows and columns, the different ways of actuation and measurement are set out respectively. Two classes in this figure are marked as 'theory'. Most control algorithms nowadays are still based on both measurement and actuation occurring synchronous in time. However, reality often lies in the semi-synchronous world, as discussed in the introduction. By making use of asynchronous observers, control algorithms can deal with asynchronous measurement, and therefore also semi-synchronous measurement (see e.g. [9] and [5]). If a control theory combines asynchronous measurement with asynchronous actuation, this would handle all defined situations. Obviously, event-driven control belongs to this specific class. When events are e.g. generated according to the size of the measured error, event generation will most probably not occur at a constant rate. If these events trigger both measurement and actuation, the algorithm is classified as being asynchronous in both measurement and actuation. For this class, event generation should be handled in hardware. When this is however implemented in software, the measurements will in most cases still occur synchronous in time. Due to implementation limitations, it is also possible to end up with semi-synchronous measurement in combination with asynchronous control.

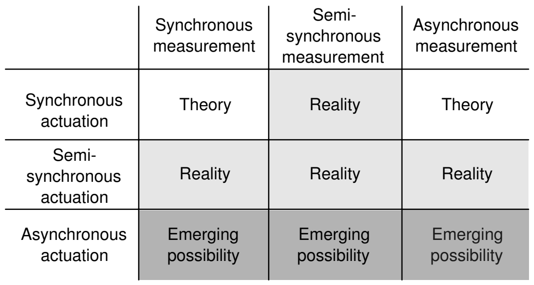
III. Event-driven PID controller

The most common controller in industry is still the PID (Proportional Integral Derivative) controller or a derived version of it. For this reason this controller is taken as a first example. This section describes the transformation of the basic time-driven control algorithm into an event-driven control strategy. In [1] a similar approach is presented but in a different setting. Their controller has two alternating sample frequencies, while the presence of noise is disregarded. Nevertheless [1] as well as this paper discuss the subject of implementing event-driven controllers to reduce processor utilization.

A standard transfer function for a continuous time PID algorithm is (1).

$$C(s) = K_p + K_i/s + K_d s L(s) \tag{1}$$

with $L(s)$, a low-pass filter to deal with high frequency measurement noise. The transfer function of this filter with a bandwidth ω_d is given in (2).



	Synchronous measurement	Semi-synchronous measurement	Asynchronous measurement
Synchronous actuation	Theory	Reality	Theory
Semi-synchronous actuation	Reality	Reality	Reality
Asynchronous actuation	Emerging possibility	Emerging possibility	Emerging possibility

Fig. 2. Classification of control algorithms.

$$L(s) = \frac{\omega_d}{s+\omega_d} \quad (2)$$

To use this controller in a discrete-event environment, the first step is to discretize the transfer function of the controller. This can be done by means of approximation formulas, e.g. Euler and bilinear transformation (Tustin). A common choice for approximating the integral part is to use Forward Euler. To approximate the derivative part in combination with the filter $L(s)$, the Tustin approximation is used. This transformation maps the entire left half of the s-plane into the unit circle in the z-plane, guaranteeing preservation of stability. The resulting transfer function of the PID controller in discrete time is given in (3).

$$C(z) = K_p + K_i \frac{T_s}{z-1} + K_d \frac{2}{T_s}\frac{z-1}{z+1} L(z) \quad (3)$$

with: $L(z) = \frac{\frac{\omega_d T_s}{2+\omega_d T_s} z + \frac{\omega_d T_s}{2+\omega_d T_s}}{z + \frac{\omega_d T_s - 2}{\omega_d T_s + 2}}$

The resulting transfer function shows a pole in z = -1 for the derivative part and a zero in z = -1 for the filter. After multiplication of the transfer functions, these are canceled out. It is therefore more efficient to rewrite the transfer function into formula (4), before implementing in software. Less computations are needed at the cost of a little less separation of functionality.

$$C(z) = K_p + K_i \frac{T_s}{z-1} + K_d \frac{2\omega_d}{2+\omega_d T_s} \frac{z-1}{z+\frac{\omega_d T_s - 2}{\omega_d T_s + 2}} \quad (4)$$

Controller (4) is suitable as an event-driven controller, with T_s as a varying parameter. Each time a control action is requested, the actual sample time T_s is determined and used. This requires accurate timing information from the points in time at which measurements are taken and actuator signals are updated. Many platforms on which the control software is executed are already equipped with this feature, called time-stamping.

The benefits of an event-driven controller over a time-driven one are mainly influenced by the way in which the events are generated. Many criteria can be chosen on the bases of which an event is generated. One obvious choice, which is used for the simulation presented in the next section, is to make the event generation dependent on the size of the tracking error. The larger the error, the more often events are generated. A second option is to let the operating system decide when the next control action should be taken. This gives the scheduler of the operating system more freedom to dynamically schedule the tasks. For stability reasons however, this freedom is limited. If we are able to make estimations of or measure process disturbances and reference changes, we can compensate these variations very fast by adjusting the sample time.

For the simulation presented in the next section, the events are generated dependent on the error. An event is generated when both the measured error and the elapsed time since the last control action satisfy the criterium in (5). K_e and e_T are positive numbers. $T_{s,min}$ is the minimum sample time. The present time and the time the last event was generated are respectively called t_p and t_l. T_s is calculated by subtracting t_l from t_p. With this method, the sample time T_s decreases when the absolute value of the measured error increases. This can also be seen in figure 3, which displays the calculated sample time versus the error $e(t)$. No event will be generated as long as $t_p - t_l$ versus $e(t_p)$ is below the depicted characteristic. When the error is smaller than e_T, no event will be generated as well. Therefore, both conditions in (5) should be satisfied, before a control action will be taken.

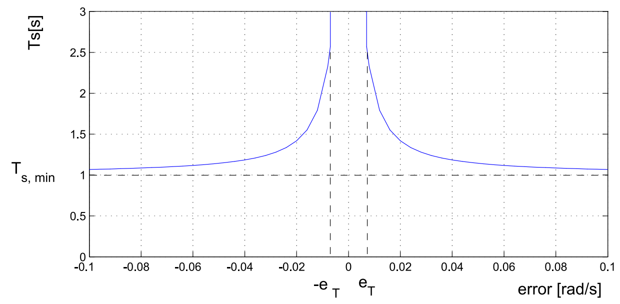


Fig. 3. Depiction of criterium (5).

$$t_p - t_l > \frac{0.5\pi \cdot T_{s,\min}}{\arctan(K_e \cdot |e(t_p)|)} \; and \; |e(t_p)| > e_T \quad (5)$$

The choice of parameters K_e and e_T influences both the number of control events and the controller performance. The aim is to find values for K_e and e_T for which the required control performance is met, while using as less control actions as possible. Parameter e_T mainly influences the maximum error and oscillatory behavior in the error. By increasing e_T the total number of generated events generally decreases, but the maximum error increases and the oscillatory behavior occurs more often. This is because these oscillations only occur when the error decreases within the bounds $|error| \le e_T$. If these bounds are tight, the error reaches the bounds less often. The value of K_e influences the sample time at points in time at which the error is larger than e_T. Increasing K_e decreases the sample time which improves the control performance but also increases the number of generated events. Decreasing K_e will worsen the control performance which will lead to bigger oscillations in the error. These oscillations can as well lead to more generated events.

A (heuristic) algorithm to choose the right values for K_e and e_T is: First choose an appropriate e_T, while keeping the value for K_e relatively high, so that the maximum error is within specification. Then decrease the value of K_e so that the number of generated events is minimal, while still satisfying the requirements of the maximum error. As will be seen in the next section, the choice of these values depends on the noise level. One of our future

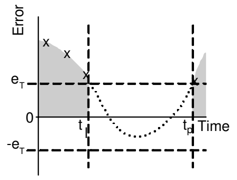


Fig. 4. Example of an error signal.

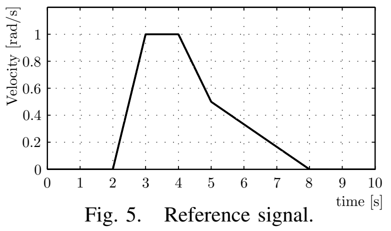


Fig. 5. Reference signal.

goals is to find systematic methodologies for tuning these controller parameters. These methodologies will take into account performance and stability criteria by means of existing analysis methods from the field of hybrid systems and sampled-data systems.

As was said before, each time an event is generated and a control action is requested, the value for T_s needs to be determined. This is done by subtracting the last stored time-stamp (t_l) at which an event was generated from the current time (t_p). A problem now arises at the first event at which the second condition of (5) becomes true. See figure 4 for an example. If time-stamps t_l and t_p and corresponding samples would be used to calculate the contribution of the integral term of the actuator signal at time t_p, the result would be larger than it should be. For this reason the integral term is not updated at time t_p but kept constant during the period $[t_l, t_p]$. For the derivative term it also holds that the obtained value when using $T_s = t_p - t_l$ for calculating the derivative term, does not give an accurate value. It could therefore be considered to set the derivative term to zero at time t_p. However, in the situations where the signs of the error signal are opposite (e.i. $e(t_l) \cdot e(t_p) < 0$), while entering and leaving the zone where $|e(t)| \leq e_T$, the reasoning is different. In this case making use of $T_s = t_p - t_l$ when calculating the derivative term, gives a better prediction compared to skipping this contribution for one sample. For this reason, the actuator signal calculated at time t_p consists of:

- the proportional term,
- the derivative term (including filter), calculated with $T_s = t_p - t_l$,
- the integral term, which is the same as calculated at t_l.

IV. Simulations

The controller described in section III is used in simulations to control the angular velocity of a DC-motor. A simplified model of the motor is selected. The input of the model is the motor voltage, the output is the speed of the motor axis. The transfer function $H(s)$ is given in (6). In this function, two time constants can be distinguished: τ_1 and τ_2 chosen 1/3 [s] and 1/6 [s] respectively. The static gain A of the motor is chosen to be 10 [rad/Vs].

$$H(s) = \frac{A}{(\tau_1 s + 1)(\tau_2 s + 1)} \tag{6}$$

With loop-shaping, the gains of the continuous controller are calculated, resulting in: $K_p = 30$, $K_i = 40$, $K_d = 2$. In industry the sample frequency of the time-driven controller is often chosen approximately 20 times the bandwidth of the system. This bandwidth, defined as the zero-dB crossing of the open loop amplification, is 57 Hz in the considered example. The sample frequency is therefore chosen to be 1 kHz. To improve the performance of the controller, a feed forward term was added that feeds-forward the set-point speed multiplied by a gain of $\frac{1}{A} = 0.1$. Furthermore, the output of the controller is saturated at +10 and -10 Volt. The bandwidth of the low-pass filter f_d is chosen to be 200 Hz.

Various simulations have been carried out with the reference velocity shown in figure 5. In the first simulation, the standard time-driven PID controller was used. This is the same controller as described in section III, but the variable T_s was chosen constant at 1 kHz. A time-driven controller is a specific class of event-driven controllers with events generated at a constant frequency. The result of this simulation is shown in figure 6. This figure also presents the results of a simulation with the event-driven controller, with events generated as described in the previous section.

For comparison, the parameter e_T of equation (5) is chosen in such a way that the maximum error of the event-driven simulation approximates the maximum error obtained from the time-driven simulation. K_e is chosen to minimize the amount of events needed to control the system. The value of $T_{s,min}$ is chosen the same as the sample time of the time-driven controller. The values are: $e_T = 5 \cdot 10^{-4}$, $K_e = 500$, $T_{s,min} = 0.001$. As can be seen from figure 6, the event-driven simulation does not control the error to zero. In most industrial applications however, there are only requirements given for the maximum value of the error. Although, it should be noted that oscillations in the error signal can cause inconvenient side affects in certain applications.

The third plot in figure 6 shows the number of samples that are needed for both control algorithms. The amount of samples that are needed for the time-driven controller is 10,000 as it is running on a constant sample frequency of 1 kHz for 10 seconds. The number of samples needed for the event-driven controller is reduced to about 1100, so a reduction of 89%. This number depends on the chosen reference signal and the level of the noise, like e.g. measurement noise. For the reference signal in this simulation, the chosen way of generating interrupts is very beneficial, because the controller does not have to perform many actions when the velocity is constant and the noise is zero. The main reason is that the system is open-loop stable.

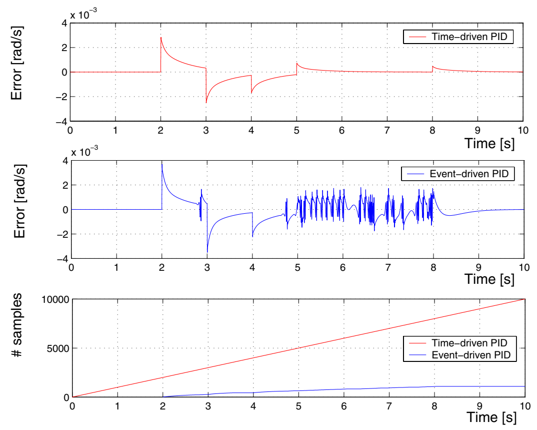


Fig. 6. Simulation results of time-driven and event-driven simulation.

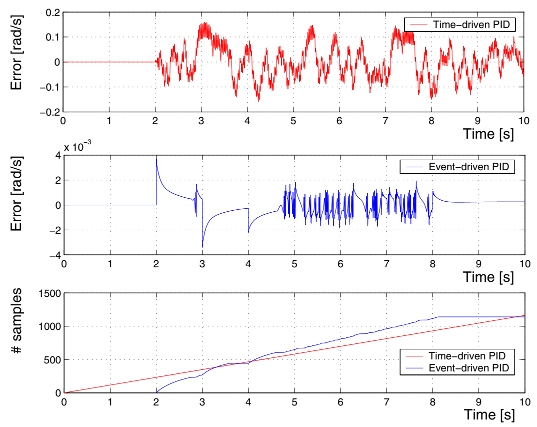


Fig. 7. Simulation results of time-driven and event-driven simulation.

Figure 7 shows the same simulation as figure 6, but here the sample time of the time-driven controller is chosen equal to the resulting average sample time of the event-driven controller. Compared to the event-driven controller, the time-driven controller performs a lot worse. The size of the error increased with a factor of approximately 100.

In the next simulations uniformly distributed noise is added to the output of the process, with a maximum value of 3% of the measured velocity. To obtain the best results with the event-driven controller, the value of e_T needs to be increased to make sure it will not be triggered continuously by the noise. After this, K_e has to be optimized as well. The new values are: $e_T = 7 \cdot 10^{-3}$, $K_e = 100$. The results of the simulations with measurement noise included are depicted in figure 8. To demonstrate the benefits of the event-driven controller, the parameters are optimized for the number of generated events. A slightly worse performance of the event-driven controller compared to the time-driven controller is to be accepted here. With these values it is still possible to control the system with less than 1700 events (83% reduction) during this simulation as can be seen in the third graph of figure 8.

The event generator (5) can be implemented in both hardware or software. When implemented in software, a time-driven sampler needs to measure the error at a constant frequency. The choice of the anti-aliasing filter (implemented in hardware) is straightforward, depending on the sample frequency of the sampler. An open question is how to choose the anti-aliasing filter for filtering the input signal for the controller when the event generation is handled in hardware. When the event generator is implemented in software, the fact that these additional calculations require time and introduce unknown delays, needs to be considered. In the presented example these delays are neglected as their influence is expected to be of minor importance. Further simulations and experiments have to validate this statement.

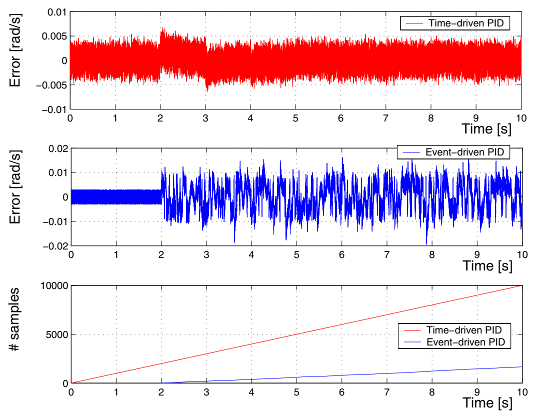


Fig. 8. Simulation results with measurement noise added.

V. Some preliminary analysis techniques

We will present in this section some preliminary ideas for analysis of a simplified event-driven control scheme. The way to do this is to consider a continuous-time plant like (6) in state-space description

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{7}$$

and to consider a discrete-time controller (+zero-order hold) with a fixed sample time $T_s = T_{s,min}$ as a state-feedback

$$u(t) = Fx(kT_s) \text{ for } t \in [kT_s, (k+1)T_s) \tag{8}$$

The sample time T_s is the lower bound in (5) and the matrix F is tuned for this particular sample time, so that the system behaves according to the requirements. We study the "stabilization problem" of controlling the system's state towards a region close to the origin and keep it there, as we cannot expect asymptotic stability. A term that is used in this context is ultimate boundedness (see e.g. [6]).

The scheme (5) is simplified by removing the first condition in (5). The condition $|e(t_p)| > e_T$ in (5) will be replaced by $x(t) \notin \mathcal{B}$, where $\mathcal{B} \subset \mathbb{R}$ is an open set containing the origin. Mathematically the simplified switching rule can be represented as

$$u(t) = Fx(t), \quad \text{if} \tag{9a}$$

- $x(t) \notin \mathcal{B}$ and $t - t_l = T_s$ or
- $x(t) \notin \mathcal{B}$ and for all $\varepsilon > 0$ there exists $x(\tau) \in \mathcal{B}$ for all $\tau \in [t - \varepsilon, t]$,

$$u(t) = Fx(t_l), \quad \text{otherwise ,} \tag{9b}$$

where t_l denotes as before the time instant at which the last control update was performed.

Loosely speaking, this means that if the system's state $x(t)$ is evolving outside $\mathcal{B}$ the synchronous sampled-data system, defined by

$$\begin{aligned}\dot{x}(t) &= Ax(t) + Bu(t) \\ u(t) &= Fx(kT_s)\text{for } t \in [kT_s, (k+1)T_s)\end{aligned} \tag{10}$$

is active. When $x(t)$ is evolving inside $\mathcal{B}$ the control values are not updated and kept constant until the boundary of $\mathcal{B}$ is hit at which the control value is updated again (i.e. (10) is directly switched on again). Hence, one could say that inside $\mathcal{B}$ no control updates happen, while outside $\mathcal{B}$ (10) is active. Note that control updates are not synchronous for the system (7)-(9). The duration that the state of the system remains inside $\mathcal{B}$ causes asynchronicity, but T_s is a fixed sample time outside $\mathcal{B}$.

We call the system (7)-(9) ultimately bounded to the set Ω, if for each $x_0 \in \mathbb{R}^n$ there exists a $T > 0$ such that the state trajectory x of (7)-(9) with initial condition $x(0) = x_0$ satisfies $x(t) \in \Omega$ for all $t > T$. The ultimate bound can be used to see if the accuracy specifications for the controller are met. We present a Lyapunov-like approach to the estimation of the ultimate bound and a way to see how the ultimate bound depends on the choice of $\mathcal{B}$.

Theorem 5.1: Consider the system (7)-(9) with $\mathcal{B}$ an open set containing the origin. Let $V : \mathbb{R}^n \to [0, \infty)$ be a continuously differentiable function that satisfies

1) $V(x) > 0$ for all $x \neq 0$ and $V(0) = 0$ (positive definite)
2) There exists $\varepsilon > 0$ such that $\frac{d}{dt}V(x(t+)) = \lim_{\tau \downarrow t} \frac{d}{dt}V(x(\tau)) < -\varepsilon V(x(t))$ at points $x(t) \notin \mathcal{B}$ along trajectories of the *synchronous sampled-data system* (10).
3) $V(x) \to \infty$, when $\|x\| \to \infty$ (radial unboundedness).

Let $\alpha^* := \sup_{x \in \mathcal{B}} V(x)$. Then the system (7)-(9) is ultimately bounded to the set $\Omega_{\alpha^*} := \{x \in \mathbb{R}^n \mid V(x) \leq \alpha^*\}$.

Proof: Using standard Lyapunov arguments it can be shown that from each initial state $x(0) = x_0$, we reach the set $\mathcal{B}$. Indeed, since $V(x(t)) \leq e^{-\varepsilon t}V(x_0)$ as long as $x(t) \notin \mathcal{B}$, it is clear that (at least) the boundary of $\mathcal{B}$ will be reached in finite time. Hence, the set $\mathcal{B} \subseteq \Omega_{\alpha^*}$ is reached in finite time. Since Ω_{α^*} is a positive invariant set due to statement 2) in the theorem, the results follows. ■

Note that the standard Lyapunov conditions hold only outside $\mathcal{B}$ and that it has to satisfy these properties for the synchronous sampled-data system (10) and not for the event-driven controlled system (7)-(9). Studying these functions and constructing these (for linear time-invariant systems quadratic Lyapunov-like functions might be used) are typical issues for future research. Next we have a theorem that shows how the set $\mathcal{B}$ has to be constructed to guarantee certain ultimate bounds on the state of the system.

Theorem 5.2: Consider the system (7)-(9) with $\mathcal{B}$ an open set containing the origin. Let $V : \mathbb{R}^n \to [0, \infty)$ be a function that satisfies the properties of theorem 5.1. Moreover, suppose there exists a ρ such that for all $\beta > 0$ and all x, $V(\beta x) = \beta^\rho V(x)$ (This replaces the radial unboundedness condition). If we replace $\mathcal{B}$ by $\lambda\mathcal{B}$ for some $\lambda > 1$ in (9), then the statement of theorem 5.1 holds for Ω_{α^*} replaced by $\Omega_{\lambda^\rho \alpha^*} = \lambda\Omega_{\alpha^*}$.

Proof: Since $\sup_{x \in \lambda\mathcal{B}} V(x) = \lambda^\rho \alpha^*$, where α^* is as in Theorem 5.1, the result follows. ■

This theorem gives a means to tune the ultimate bound on the state and can be used to design the event driven controller. Indeed, if we have a Lyapunov-like function V outside $\mathcal{B}$, then scaling $\mathcal{B}$ with a constant $\lambda > 1$ leads to a "stabilization error" that is λ times larger.

VI. CONCLUSIONS

This article presents an event-driven controller as an opportunity to reduce the gap there is between control theory and computer science. Event-driven control creates a negotiable environment to make a compromise between processor load and control performance. This can make the implementation of control algorithms on processor boards more flexible. Moreover, a possibility has been created for control engineers to deal with implementation difficulties, like e.g. unforeseen delays, by means of time stamping.

If the criterium on which events are generated is chosen in the right way, event-driven control can reduce the average processor load. The reason for this is that in many controlled systems, a high actuation frequency is only needed on limited points in time. In the example explained in section IV, the average sample frequency was reduced with 89% in the situation without noise. When noise was added, a reduction of the sample frequency of 83% was obtained.

This paper discussed and showed the potential of event-driven control. However, more validation and analysis is needed to substantiate these first findings. The first ideas for analysis and synthesis techniques look promising and future research will focus on the extension of these techniques.

REFERENCES

[1] Årzén, Karl-Erik (1999). A simple event-based PID controller. In: *Preprints 14th World Congress of IFAC*. Beijing, P.R. China.

[2] Årzén, Karl-Erik, Anton Cervin and Dan Henriksson (2003). Resource-constrained embedded control systems: Possibilities and research issues. In: *Proceedings of CERTS'03 – Co-design of Embedded Real-Time Systems Workshop*. Porto, Portugal.

[3] Cervin, Anton (2003). Integrated Control and Real-Time Scheduling. PhD thesis. Department of Automatic Control, Lund Institute of Technology, Sweden.

[4] Doff, R. C., M. C. Fatten and C.A. Phillips (1962). Adaptive sampling frequency for sampled-data control systems. In: *IRE Transactions on Automatic Control*. Vol. AC-7. pp. 38–47.

[5] Grewal, Mohinder S. and Angus P. Andrews (1993). *Kalman filtering: theory and practice*. Englewood Cliffs: Prentice Hall.

[6] H.K. Khalil (2002). Nonlinear Systems. *Second edition, Prentice Hall, chapter 5.*

[7] Lincoln, Bo (2002). Jitter compensation in digital control systems. In: *Proceedings of the 2002 American Control Conference*.

[8] Network of Excellence. HYbrid CONtrol. [online] http://www.ist-hycon.org.

[9] Phillips, A.M., and Tomizuka, M. (1995). Multirate estimation and control under time-varying data sampling with applications to information storage devices. In: *Proceedings of the 1995 American control conference*. Vol. 6. pp. 4151–4155.

[10] Sanz, Ricardo and Karl-Erik Årzén (2003). Trends in software and control. *IEEE Control Systems Magazine* **23**(3), 12–15.