Draft paper submitted to the IFAC World Congress 1999.

A SIMPLE EVENT-BASED PID CONTROLLER

Karl-Erik Årzén

Department of Automatic Control,
Lund Institute of Technology,
Box 118, S-221 00 Lund, Sweden,
Email: karlerik@control.lth.se

Abstract: A simple event-based PID controller is presented. It is shown that it is possible to obtain large reductions in CPU utilization with only minor control performance degradation. Simulations on a double-tank process are presented.

Keywords: PID control, computer control, event-based sampling

1. INTRODUCTION

The majority of the work and research in automatic control considers periodic or time-triggered control systems where continuous time signals are represented by their sampled values at equi-distant sampling intervals. The major reason for this is the existence of a well established system theory for sampled data systems and sampled control systems, e.g., Åström and Wittenmark (1997).

However, there are cases when it is motivated to also consider event-based control systems where the sampling is event-triggered rather than time-triggered. Other names for these control systems are aperiodic or asynchronous control systems. In an event-based system it is the occurrence of an event rather than the passing of time, that decides when a sample should be taken. The nature of the event could vary. Examples could be that a measurement signal crosses a certain limit, or the arrival of a data packet to a node on a computer network. A comparison between periodic and event-based sampling for first order stochastic systems is found in Åström and Bernhardsson (1999).

One example of time-varying sampling intervals is control of internal combustion engines that are sampled against engine speed. Another example is manufacturing system where the sampling can be related to production rate. The event-based nature of the sampling can also be intrinsic to the measurement method used, or to the physical nature of the process being controlled. Alternatively, the event-based sampling can be a built-in feature of an intelligent sensor device. Event-based sampling is natural when encoder sensors are used or when the actuators are of an on-off nature, e.g., in satellite control with thrusters, Dodds (1981), or in systems with pulse frequency modulation, e.g., Sira-Ramirez (1989). Event-based sampling is also used in the process industry when statistical process control (SPC) is used in closed loop. In order not to disturb the process, a new control action is only calculated when a statistically significant deviation has occurred.

Modern distributed control systems also impose system architectural constraints that make it difficult to stick to the time-triggered paradigm. This is specially the case when control loops are closed over computer networks or buses, e.g., field buses, local area networks, ATM networks, or even the Internet, e.g., Nilsson (1998).

Another reason why event-based control is interesting is that it closer in nature to the way a human behaves as a controller. The human motion control system is event- or pulse-based rather than time-triggered, Mead (1989). Also, when a human performs manual control his behaviour is event-based rather than time-triggered. It is not until the measurement signal has deviated sufficiently enough from the desired set point that a new control action is taken.

The final reason why event-based control is of interest is resource utilization. An embedded controller is typically implemented using an real-time operating system that supports concurrent programming. The available CPU time is shared between the tasks in such a way that it appears as if each task is running independently. Occupying the CPU resource for performing control calculations when nothing significant has happened in the process is clearly an unnecessary waste of resources. The same argument also applies to communication resources. The available communication bandwidth in a distributed system is limited. To use this for sending data in a time-triggered fashion is clearly a waste of bandwidth.

What is then the reason why time-triggered control still dominates? A major reason is the difficulty involved with developing a system theory for event-based systems. Although much work was done in the 1960s and 1970s, the interest during recent years has been low. Another reason is the interface that has been established between the control community and the real-time computing community. The control community typically assumes that the real-time platforms used for implementing controllers are able to guarantee deterministic sampling intervals. In reality this is, however, not always true. The market push for using commercial off-the-shelf systems such as, e.g., Windows NT, also for real-time applications means that the determinism decreases. Modern hardware features such as caches and instruction pipelines also decreases the determinism, or worst-case performance, while drastically improving the average-time performance.

Similarly the real-time computing community assumes that all control loops are periodic, with fixed periods, and that they have hard deadlines. For these types of systems the rate monotonic scheduling theory and its extensions, Buttazzo (1997), have been derived. Using this theory it is in theory possible to formally decide whether a given task set will meet its computation deadlines are not. However, the weak point of the theory is the need to have a close upper bound on the worst case execution time (WCET). In practice this is very difficult to obtain. From the examples above it is clear that there are several control loops that are aperiodic. It is also easy to find examples of, e.g., heterogeneous or hybrid controllers, that switch among a set of control algorithms, each possibly with a different sampling interval. Several control loops have hard deadlines, e.g., controllers controlling open loop unstable processes. However, there are also a large number of examples where the deadlines are soft rather than hard. As long as the deadline is not missed too often, with too much, or under certain operating conditions, the controller can either be designed to be robust against this or to compensate for it.

In the Swedish research projects ARTES and DICOSMOS the interaction between control, computing, and communication is studied from different perspectives. In this paper a heuristic event-based PID controller is presented. The aim of the controller is to obtain comparable control performance with a conventional PID controller, with drastically reduced CPU utilization.

Outline of the paper

The approach taken is discussed in Section 2. The standard PID control algorithm is presented in Section 3, and the modifications needed in the event-based case are shown in Section 4. The proposed controller has been evaluated both in simulations and in laboratory experiments. In Section 5, some real-time simulation results are presented where a double-tank process is used as a test case.

2. EVENT-BASED CONTROL

The reasons for sampling in an event-based controller may vary. Here, we will consider systems where the decision to calculate a new control signal is based on level crossings of different signals. The basic setup is shown in Figure 1. The controller consists of two parts, an event detection part that uses time-triggered sampling with a sampling interval, h_{nom}, that is the same as the sampling interval of the corresponding time-triggered PID controller. The output of the event detection part is a request to the PID controller algorithm that a new control signal should be calculated. Included with the request is the length of the most recent sampling interval, i.e., the time since a new control signal was calculated last. This information can be used by the control algorithm to compensate for the sampling interval variations.

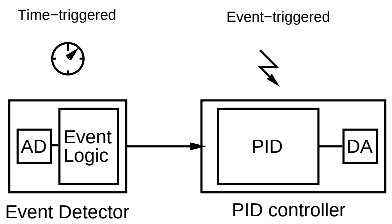

Fig. 1 Event-based PID structure

The signals involved in the detection logic can be of different types. One possibility is to sample only when the measurement signal y crosses a certain prespecified level. An alternative is to instead use relative measures, i.e., to sample when the change in the measurement signal crosses a certain limit. It is, however, not only changes in the measurement signal that should cause sampling. Sampling should also be performed when the set point is changed. Therefore it is an advantage to use the error signal rather than the measurement signal as the basis for the event condition.

The event detection logic could in principle be arbitrarily complex. One could, e.g., consider using error derivatives to predict error changes and cause samplict before the error change actually has occurred. Another possibility would be to also use the control signal in the logic, and only allow longer sampling intervals if the control signal indicates that steady state has been reached. However, in order to gain anything compared with a time-triggered PID controller it is important to keep down the complexity. The event condition used in the simulations in Section 5 is the following;

$$|e(t_k) - e(t_s)| > e_{lim} \text{ OR } h_{act} \geq h_{max}$$

A decision is made to calculate a new control signal if the absolute value of the difference between the current value of the error, $e(t_k)$, and the value of the error when a control signal was calculated the last time, $e(t_s)$, is greater than a limit, or when the time elapsed since the last sample, h_{act} exceeds the limit h_{lim}. The last condition is a simple safety measure. The effect of the condition will be that controller will execute at the nominal sampling time h_{nom} during transients, i.e., set point changes and load disturbances, and that the controller will execute at the maximal sampling interval during steady state conditions.

The event-based PID control structure can be viewed as a client-server architecture. A control may consist of a number of control loop clients that performs the high-frequency sampling. When one of them needs a new control signal calculation a request is sent to the PID server, that returns the newly calculated control signal.

The partition of a PID controller into two parts with different sampling intervals is nothing new. Similar structures are found in most commercial computer-based PID control system. Low-pass anti-aliasing filtering must be implemented on analog form. However, to avoid analog filters with varying filter parameters it is common to have a fixed analog filter combined with fixed fast sampling. A discrete filter is then used to perform the remaining low-pass filtering for the PID controllers that execute at a considerable slower frequency. The discrete filter parameters can easily be adjusted as the sampling interval of the PID controller is changed.

3. PID CONTROL

PID control is by far the dominating control structure in industrial practice. The textbook PID controller has the following basic structure, Åström and Hägglund (1995):

$$U(s) \quad = \quad K(E(s) \quad + \quad \tfrac{1}{sT_I}E(s) \quad + \quad T_D s E(s))$$

in the frequency domain. To avoid problems with high frequency measurement noise in the derivative part a low-pass filter is added. It is also quite common with set-point weighting in the derivative part. The derivative part is therefore realized as

$$U_D(s) = \frac{KT_D s}{1 + sT_D/N}(\gamma Y_{sp}(s) - Y(s))$$

In process control applications γ is often set to zero.

Set point weighting is also commonly used in the proportional part, i.e.,

$$U_P(s) = K(\beta Y_{sp}(s) - Y(s)).$$

The set point weighting terms can be interpreted as feedforward from the set point, thus giving the PID controller two dimensions of freedom.

3.1 *Discretization*

In order to implement a PID controller in a computer it has to be converted into digital form. A common way of doing this is to discretize the controller, i.e., to approximate the continuous time derivatives using, e.g., backward difference approximation, forward difference approximation, or Tustin approximation.

The proportional part $U_P(s)$ is straightforward to discretize by replacing the continuous variables with their sampled versions. For the integral part and the derivative part there are several possibilities. A common choice is to use forward differences for the integral part. The reason for this choice is that it is possible to pre-calculate the integral part for time t_{k+1} already at time t_k. This reduces the calculations that need to be done in between the sampling of the measurement signal y and the generation of the control signal u, thus reducing the control delay. For the derivative term it is common to chose a backward difference approximation. This approximation of the derivative part is stable for all values of T_D. The corresponding discrete parameters are also always positive, thus avoiding problems with ringing. Details about the discretization can be found in Åström and Hägglund (1995).

The total code for the PID controller is

```
(* Pre-calculated coefficients *)
bi := K*h/Ti;
ad := Td/(Td + N*h);
bd := K*Td*N/(Td + N*h);
(* Calculate control signal *)
ysp := ADIn(ch1);
y := ADIn(ch2);
up := K*(beta * ysp - y);
ud := ad*ud - bd*(y - yold);
u := up + ui + ud;
DAOut(u,ch3);
(* Update states *)
ui := ui + bi*(ysp - y);
yold := y;
```

It is assumed that $\gamma = 0$. In order for the code to be complete, an anti-reset windup mechanism has to be added, e.g., based on tracking.

The controller coefficients bi, ad, and bd are normally pre-calculated to minimize the computations needed to calculate the control signal. When a conventional PID controller is implemented the sampling period is assumed to be constant. In an event-based PID controller the sampling period will vary from sample to sample. However, this variation can be compensated for by adjusting the controller coefficients every sample. The price for this is increased computation time.

4. EVENT-BASED PID CONTROL

The code for the event-based PID controller has the following structure:

```
(* Pre-calculated parameter *)
bi := K / Ti;
(* Event detection *)
ysp := ADIn(ch1);
y := ADIn(ch2);
e := ysp - y;
hact := hact + hnom;
IF (abs(e - es) > elim) OR (hact >= hmax) THEN
  es := e;
  ad :=  Td/(Td + N*hact);
  (* Calculate control signal *)
  up := K*(beta * ysp - y);
  ud := ad*ud - ad*K*N*(y - yold);
  u := up + ui + ud;
  DAOut(u,ch3);
  (* Update states *)
  ui := ui + bi*hact*(ysp - y);
  yold := y;
  hact := 0.0;
ENDIF;
```

The code is executed at the nominal sampling frequency. The PID calculations are, however, only performed if the event condition is true. Compared with the time-triggered PID algorithms the proposed algorithm requires more calculations in the calculate output part of the code, mainly due to the fact that the PID coefficients are recalculated for every calculation. The algorithm also requires more state variables, e.g., `es` and `hact`, and two additional parameters, the error limit and the maximum sampling interval. The increase in complexity is, however, not drastic.

5. SIMULATIONS

The simulations have been performed on a double-tank process where the aim is to control the level of the upper or lower tank with the pump signal as the control signal, according to Fig 2. The simulations have been performed in real-time using the G2 programming environment, Moore *et al.* (1990).

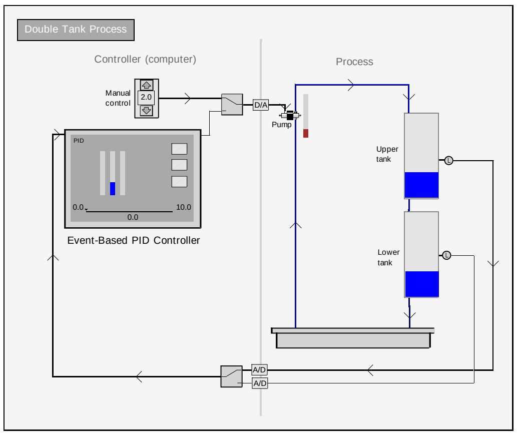

Fig. 2 Double tank process

In the first example we will control the level of the upper tank with a PI-controller. The parameters of the controller are $K = 4$ and $T_I = 20$. The nominal sampling interval is 1 second and the maximal sampling interval is 10 seconds. The process is simulated for 10 minutes. At time 0 (time $t - 10$) the set point is set to 2.0. At time 3 minutes the set point is changed to 5.0 and at time 5 minutes it is changed back to 2.0. An input load disturbance is introduced at time 8 minutes. The results of the simulation are shown in Fig. 3. The top plot shows the set point and the measured signal. The mid plot shows the control signal and the bottom plot shows how the actual sampling interval varies between the nominal value and the maximum value.

As a comparison the simulations with $h_{nom} = h_{max} = 1$, i.e., the corresponding time-triggered PID controller are shown in Fig. 4. The performance is comparable. For the case $h_{nom} = h_{max} = 10$, the system becomes unstable.

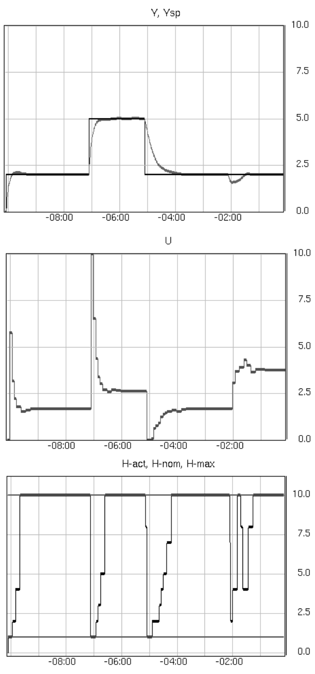

Fig. 3 PI control of upper tank with $h_{nom} = 1$ and $h_{max} = 10$

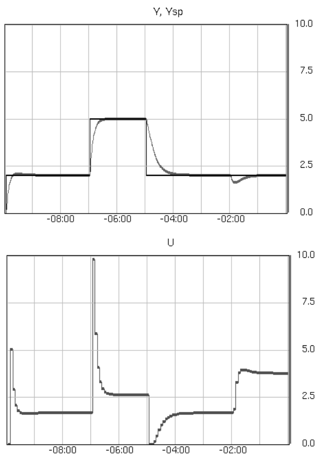

Fig. 4 PI control of upper tank with $h_{nom} = h_{max} = 1$

The necessity of compensating the PID coefficients for the varying sampling interval is shown in Figs. 5 and 6. In Fig. 5 the PID coefficients are kept constant as if the actual sampling period was equivalent to the nominal sampling period. In Fig. 6 we instead assume that the actual sampling period is equivalent to the maximum sampling period. A too high value of the sampling interval will have the same effect as if T_I is too small and a too low value of the sampling interval will have the same effect as if T_I is too large. The typical result of this is clearly seen in the figures.

With a time-triggered PID the control algorithm would be executed 600 times over a 10 minute simu-

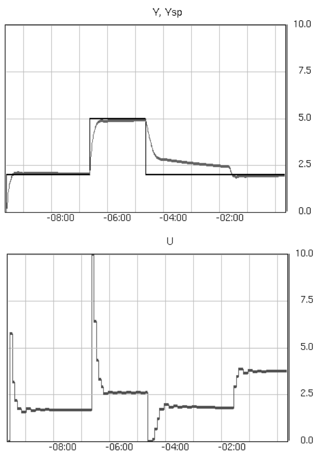

Fig. 5 No compensation for varying sampling interval. Assume that $h_{act} = h_{nom}$

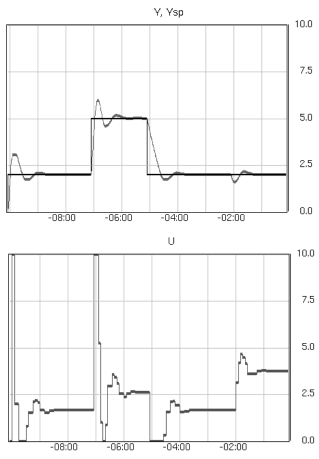

Fig. 6 No compensation for varying sampling interval. Assume that $h_{act} = h_{max}$

lation. In the event-triggered PID in the example the event detection logic is executed 600 times but the PID algorithm is only executed 103 times. One of the goals of the proposed approach is to reduce the processor utilization. An approximate analysis of this can be performed. The CPU utilization of a task i is defined as C_i/T_i where C_i is the WCET of task i and T_i is the period of task i. If we call the WCET for the time-triggered PID for C_{TPID} and assume that the WCET for the event detection part is equal to $C_{TPID}/4$ and that the WCET for the PID calculation in the event-triggered PID is equal to C_{TPID}, the following result can be derived. In the analysis the event-triggered PID is treated as two tasks: the event detection part with a period of 1 second and the PID calculation part with an average period of 6 seconds. This gives a utilization for the time-triggered PID of C_{TPID} and an average utilization for the event-triggered PID of

$C_{TPID}/4 + C_{TPID}/6 \approx 0.42C_{TPID}$, i.e., a reduction in utilization of about 58%. The upper and lower bound on the utilization are $1.25C_{TPID}$ and $0.35C_{TPID}$.

In Fig. 7 the results for PID control of the level of the lower tank are shown. Also here the approach works well.

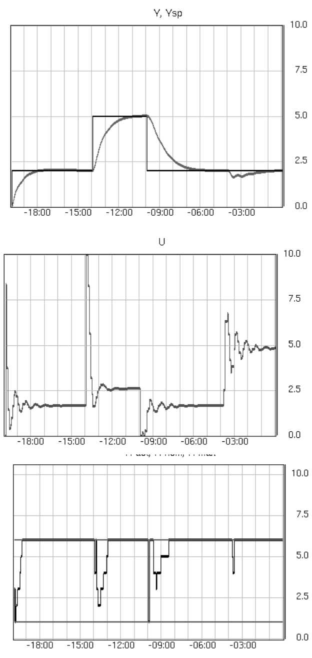

Fig. 7 PID control of the lower tank. $h_{max} = 6s$

The simulations presented have been noise-free. If measurement noise is present, the choice of the error limit becomes more important. In order to gain anything with theproposed method it is important to filter the measurement noise properly and to set the error limit sufficiently large to avoid sampling due to noise. The PID extended with the event detection logic becomes a nonlinear system of hybrid nature. Several interesting phenomena have been observed during the simulations. One example is limit cycles in the actual sampling interval. These are, however, hardly noticeable in the process output.

The approach can be extended in several directions. One example is to also make the DA-conversion conditional, i.e., to only send out a new control signal if it differs significantly from the latest control signal. This becomes equivalent to a dead band on the control signal increment.

6. CONCLUSIONS

A simple event-based PID controller has been presented. It has been shown through simulations that it is possible to obtain large reductions in the CPU utilization with only minor control performance degradation. It has also been shown that it is important to take the varying sampling interval into account in the control algorithm. The proposed controller has also been tested on a laboratory tank process with good results.

A problem with event-based control is that it becomes harder to verify and guarantee worst-case performance using, e.g., available scheduling theory, and that it is difficult to analyze the control system. Hence, there is a strong need for both a system theory for event-based control and for new results in scheduling.

This work has been supported by the ARTES project.

7. REFERENCES

ÅSTRÖM, K. J. and B. BERNHARDSSON (1999): "Comparison of periodic and event based sampling for first-order stochastic systems." In *Submitted to IFAC World Congress, Beijing*.

ÅSTRÖM, K. J. and T. HÄGGLUND (1995): *PID Controllers: Theory, Design, and Tuning*, second edition. Instrument Society of America, Research Triangle Park, NC.

ÅSTRÖM, K. J. and B. WITTENMARK (1997): *Computer-Controlled Systems*, third edition. Prentice Hall.

BUTTAZZO, G. C. (1997): *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers.

DODDS, S. J. (1981): "Adaptive, high precision, satellite attitude control for microprocessor implementation." *Automatica*, **17:4**.

MEAD, C. A. (1989): *Analog VLSI and Neural Systems*. Addison-Wesley, Reading, Massachusetts.

MOORE, R., H. ROSENOF, and G. STANLEY (1990): "Process control using a real time expert system." In *Preprints 11th IFAC World Congress*. Tallinn, Estonia.

NILSSON, J. (1998): *Real-Time Control Systems with Delays*. PhD thesis ISRN LUTFD2/TFRT--1049--SE, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

SIRA-RAMIREZ, H. (1989): "A geometric approach to pulse-width modulated control in nonlinear dynamical systems." *IEEE Transactions on Automatic Control*, **35:12**.