# Simulation of Process Control with WirelessHART Networks Subject to Clock Drift

Mauro De Biasi, Carlo Snickars, Krister Landernäs, and Alf Isaksson
ABB AB, Corporate Research
SE-721 78 Västerås, Sweden
{mauro.d.biasi,carlo.snickars, krister.landernas, alf.isaksson}@se.abb.com

***Abstract*— This work describes simulation of wireless control using the WirelessHart standard. Specifically issues concerning clock drift between controllers and wireless network are addressed. The simulations have been done using an extension of the Simulink package TrueTime.**

## I. Introduction

Wireless technologies are becoming more and more important both in public and in industrial environments. The apparent benefit of wireless communication is to remove the restriction of being attached to expensive and messy cables. The advantages given by wireless technology are several. First of all, it permits to carry the capability of wired networks to areas that cables cannot reach. Considering industrial plants, wireless technologies can significantly facilitate deployment and reconfiguration by eliminating the need for installing and maintaining cabling, reducing both cost and time. However, due to the lack of maturity and failure to provide real-time performance no wireless technology has been widely adopted for process automation. This may change with the introduction of WirelessHART.

WirelessHART is an optional HART Physical Layer that provides a low cost, relatively low speed (e.g. compared to IEEE 802.11g), wireless connection. It adopts the IEEE 802.15.4 physichal layer and it works in the 2.4GHz ISM radio band using 15 different channels. The communication between the devices is performed using Time Division Multiple Access (TDMA) with time slots of 10 ms. A series of time slots form a superframe which can be of arbitrary length. WirelessHART also enables channel hopping to avoid interferences and reduces multi-path fading effects. One or more sources and one or more destination devices may be scheduled to communicate in a given slot. The slot may be dedicated to communication from a single source device or a slot may support shared communication. In this last case, the MAC protocol used is CSMA/CA.

This work addresses the problem of clock drift between controller and the WirelessHART network that occurs when no synchronization exists between the two. Our theories have been verified using an extension to the simulation environment TrueTime [7]. This extension has added the possibility to simulate the newly released standard WirelessHART [1]. The rest of this paper is organized as follows: Section II is a general introduction to the WirelessHart protocol. Then Section III gives a brief introduction of TrueTime. In section IV the controller and the network is described in some detail, and in Section V some compensation methods are descried. This is followed by simulation examples in Section VI and some conclusions in the last section.

## II. WirelessHART Network Description

The Structure of a WirelessHART network is shown in Fig. 1. All communications of the WirelessHART Network pass through the gateway. Consequently, the gateway must route packets to the specified destination (network Device, host application, or network manager). The gateway uses standard HART commands to communicate with network devices and host applications. The plant automation network could be a TCP-based network, a remote IO system, or a bus such as PROFIBUS DP. The Network Manager creates an initial superframe and configures the Gateway. A detailed description of the components of a wirelessHART network is given in [2] and [3].

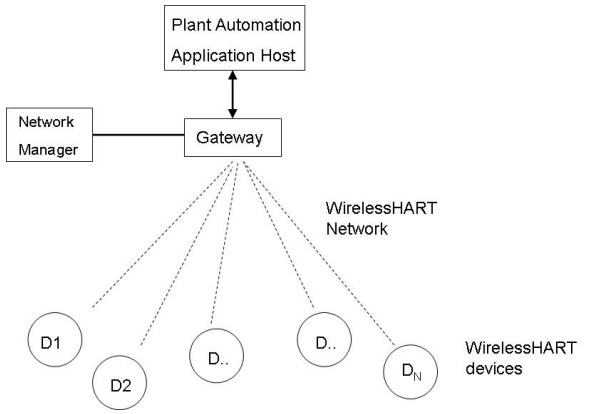


Fig. 1. The structure of a WirelessHART network.

### A. MAC Protocol Description

The main tasks of the Medium Access Control (MAC) protocol are:

- slot synchronization;
- identification of devices that need to access the medium;
- propagation of messages received from the Network Layer;

0730-3157/08 $25.00 © 2008 IEEE
DOI 10.1109/COMPSAC.2008.163

- to listen for packets being propagated from neighbors

WirelessHART uses Time Division Multiple Access (TDMA) and channel hopping to control access to the network. TDMA is a widely used Medium Access Control technique that provides collision free, deterministic communications. It uses time slots where communications between devices occur. A series of time slots form a TDMA superframe (see Figure 2).

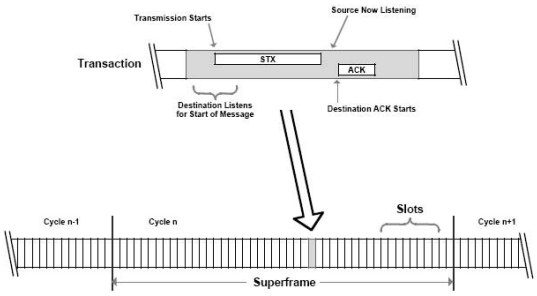


Fig. 2. The SuperFrame structure [1]

All devices must support multiple superframes. Slot sizes and the superframe length (in number of slots) are fixed and form a network cycle with a fixed repetition rate. Superframes are repeated continuously. For successful and efficient TDMA communications, synchronization of clocks between devices in the network is critical [8]. Consequently, tolerances on time keeping and time synchronization mechanisms are specified to ensure network-wide device clock synchronization. It is imperative that devices know when the start of a slot occurs. Within the slot, transmission of the source message starts at a specified time after the beginning of a slot. This short time delay allows the source and destination to set their frequency channel and allows the receiver to begin listening on the specified channel. Since there is a tolerance on clocks, the receiver must start to listen before the ideal transmission start time and continue listening after that ideal time. Once the transmission is complete and the destination device indicates, by transmitting an ACK, whether it received the source device data-link packet (DLPDU) successfully or with a specific class of detected errors. Communicating devices are assigned to a superframe, slot, and channel offset. This forms a communications link between communicating devices.

## III. Description of TrueTime

This section describes the use of the original Matlab/Simulink-based simulator TrueTime [7], which permits to design networked control systems simulating real-time kernels, network transmissions (using wired or wireless networks), and continuous plant dynamics. TrueTime is constituted by a six blocks library and by a collection of C++ functions with corresponding MATLAB MEX-interfaces. These functions are divided into two groups. One permits to configure the simulation by creating tasks, interrupt handlers, monitors, timers, etc. The other functions are real-time primitives that are called from the task code during execution and provides for AD-DA conversion, changing task attributes, entering and leaving monitors, sending and receiving network messages, etc.

### A. The TrueTime *Kernel* Block

One of the blocks contained in the library is the Kernel . It is a MATLAB S-function that simulates a CPU with a real-time kernel, A/D and D/A converters, a network interface, and external interrupt channels. The kernel is designed following a real-time model with a ready queue and a time queue. It is also characterized by records for tasks, interrupt handlers, monitors and timers that have been created for the simulation. The kernel executes an arbitrary number of user-defined tasks and interrupt handlers that may also be created dynamically at run-time. Tasks may be periodic to simulate activities such as controller and I/O tasks, or aperiodic to represent activities like communication tasks and event-driven controllers. Aperiodic tasks are executed by the creation of task instances (jobs).

### B. The TrueTime *Network* Block

The TrueTime network block permits to simulate medium access and packet transmission in a local area network choosing different communication protocols: CSMA/CD (e.g. Ethernet), CSMA/ AMP (e.g. CAN), Round Robin (e.g. Token Bus), FDMA, TDMA (e.g. TTP), and Switched Ethernet. Only packet-level simulation is supported. It is, in fact, assumed that the messages have been divided into packets at higher protocol levels. When a node tries to transmit a message (using the primitive `ttSendMsg`), a triggering signal is sent to the network block on the corresponding input channel. At the end of the transmission, the network block sends a new triggering signal on the output channel corresponding to the receiving node. Each receiving node has a buffer in which the transmitted message is put.

### C. The TrueTime *Standalone Network Blocks*

The standalone network blocks are two, the `ttSendMsg` and the `ttGetMsg`. They can be used to send messages using the network blocks without using kernel blocks. This permits (not having to initialize kernels, create and install interrupt handlers, etc.) to build quickly a simulation, without creating any M-files.

### D. The TrueTime *Wireless Network Block behaviour*

The wireless network block simulates medium access and packet transmission. Originally it implemented two kinds of communication protocols, 802.11b/g (WLAN) and 802.15.4 (ZigBee). The possibility to use also WirelessHART has been added. The block permits to simulate WirelessHART communication and study issues such as clock drift, delay and packet loss.

## IV. Description of the Control Network

Most industrial controllers, for example AC800M in the ABB 800xA distributed control system [4] [5], are not synchronized with the I/O communication. Each executing task typically consist of a number of control loop calculations and works according to a scenario like the one in figure 3. Each execution cycle is begun by reading the value of all

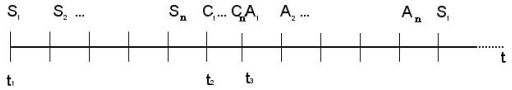


Fig. 3. AC800M typical scenario

the sensors from their respective registers. In the context of WirelessHART, then all sensor slots would have to be before any control tasks. AC800M then writes the control signals to the output registers as soon as the corresponding computation has been completed. Hence, one does not need to wait until all control signals have been computed before actuator communication can start. The reason for reading all sensor at once, is that when there are control loops with a direct connection to each other, they need to be executed together. An example of such a situation is cascaded loops. Here the outer, so called master controller, provides the setpoint to the inner slave controller, which in turn calculates the actuator signal. This means that two sensor signals are needed to calculate one actuator signal. Obviously both sensor readings need to be available before calculation can start. It is however, highly unlikely that with a large number of loops in one network they are all directly connected in this manner, why the requirement to have all sensor signals available up front is an unnecessary (but obviously convenient) restriction.

### A. Causes of clock drift in the Control Network

Consider a system like the one shown in the figure 4.

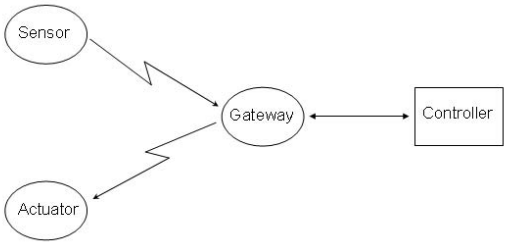


Fig. 4. A possible scenario.

The sensor sends the data to the gateway over a wireless network. The gateway writes the data in the I/O board of the controller through a wired connection. When the control signal is computed, the information is sent back to the gateway by wire and to the actuator using the wireless network. Using a wireless protocol like WirelessHART all devices that are part of that network, are synchronized. The controller, however, is not part of the wireless network and in the case in which no synchronization with the wireless nodes can be guaranteed. Thus, if periodic tasks are used to execute sensing, actuating and control calculation and if the controller is affected by clock drift, a delay can appear in the closed loop system. This is illustrated in figure 5. The

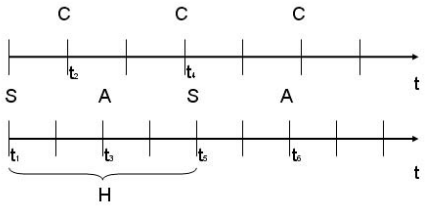


Fig. 5. Delay caused by clock drift.

control signal computated at time $t_4$ uses the sensor value from $t_1$, but will not actuate until $t_6$. The total delay in this case is not equal to the standard delay $L = t_3 - t_1$ but instead equal to $D$, where

$$D = t_6 - t_1 = H + L \tag{1}$$

From (1) it is clear that clock drift between the controller and the WirelessHART network can cause a delay equal to one superframe.

## V. Compensation Methods

In the simulations below we have studied two different controller types:

- A conventional PI controller because of its frequent use in industry.
- A special case of delay compensating controllers; the predictive PI (PPI) controller

### A. PI Controller

The continuous-time parametrization of the PI controller is

$$C(s) = K_c \left(1 + \frac{1}{T_I s}\right) \tag{2}$$

where $K_c$ and $T_I$ are the controller gain and integral time respectively. A digital implementation has been used where the control signal is calculated according to

$$u_k = u_{k-1} + K_c((1 + T_s/T_I)(r_k - y_k) - (r_{k-1} - y_{k-1})) \tag{3}$$

We will in this paper use controller settings in accordance with the *lambda tuning* technique [9], which assumes that the process is described by a first order system with time delay

$$G(s) = \frac{K}{Ts + 1} e^{-D_{sys} s} \tag{4}$$

Using an approximation of the deadtime and cancellation of the open-loop denominator the following tuning rules result

$$T_I = T, \qquad K_c = \frac{T}{K(\lambda + D_{sys})} \tag{5}$$

where $\lambda$ is the desired time constant of the closed-loop system.

### B. The PPI controller

The predicting PI controller (PPI) is a particular Smith predictor [9]. Consider:

$$G(s) = \frac{K}{Ts+1}e^{-Ds} \tag{6}$$

As in lambda tuning, the design method is to chose a controller that cancels the process pole and makes the other closed-loop pole equal to $s = -1/\lambda$, where $\lambda$ is the desired response time of the closed-loop system.

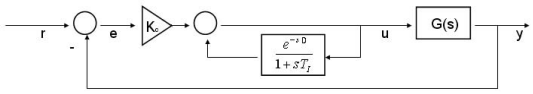


Fig. 6. The PPI controller.

The controller parameters obtained are the following:

$$K_c = \frac{T}{\lambda K}, \qquad T_I = T \tag{7}$$

The controller is:

$$C(s) = \frac{1+sT}{Ks\lambda}\frac{1}{1+\frac{1}{s\lambda}(1-e^{-Ds})} \tag{8}$$

The controller can be seen as a PI controller that acts on a predicted error, which is the actual error compensated for past control actions that have not yet appeared at the output. From that it takes its name, *Predicting PI controller*.
Choosing $\lambda = T$ gives the controller a particularly simple form also depicted in figure 6.

$$U(s) = K_c E(s) + \frac{1}{1+T_I s}e^{-sD}U(s) \tag{9}$$

Defining:

$$Z(s) = \frac{1}{1+T_I s}U(s) \tag{10}$$

then (9) becomes

$$U(s) = K_c E(s) + Z(s)e^{-sD} \tag{11}$$

To implement a digital PPI, it is necessary to convert to the discrete domain:

$$z_k = \frac{(1-a)q^{-1}}{1-aq^{-1}}u_k \tag{12}$$

where $a = e^{-\frac{T_s}{T_I}}$. The equation to update $z_k$ is :

$$z_k = az_{k-1} + (1-a)u_{k-1} \tag{13}$$

Considering (11) and (13) it is possible to obtain the digital control command:

$$u_k = K_c e_k + z_{k-1} \tag{14}$$

In figure 13 the reference tracking for this implementation of the PPI is compared with the PI performance. A similar simulation comparison for delay compensation in wireless communication also deploying the PPI has been presented in [6].

## VI. Simulations of systems with delay due to clock drift using wirelessHART protocol

In this section the problem of delay due to the clock drift in the controllers like the AC800M will be treated. All the tests have been made using simplified plants with one loop in which the devices communicates using the WirelessHART protocol.

### A. Simulation setup

The model considered is a stable first order system:

$$G(s) = \frac{0.25}{2s+1} \tag{15}$$

The global scheme of the control system is illustrated in figure 7. It has been constructed in a Simulink environment, using the TrueTime modified wireless network block.

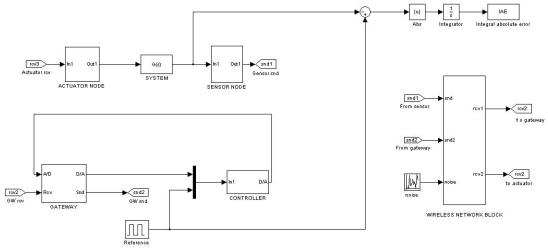


Fig. 7. Representation of the simulated control system

In this control system there is only one loop. The sensor sends the measurement to the gateway that is responsible to communicate with the controller using a wired protocol (like Profibus or Fieldbus) and to send the control signal to the actuator using the wireless network. The communication between sensor, gateway and actuator is provided by the WirelessHART protocol with a superframe size of 1 s and a time slot of 10 ms. The communication schedule is shown in figure 8.

Looking at this schedule, it is possible to notice that after the transmission between the sensor and the gateway ($S \rightarrow$

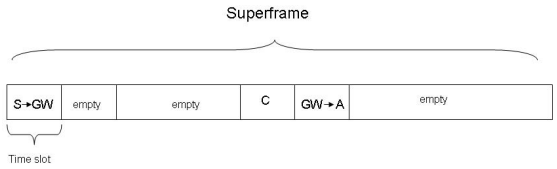


Fig. 8. Communication schedule

$GW$) there are some empties slots. These slots are left empty to simulate the same idle time of the case in which there are several loops. In this specific scenario the time elapsed between sensing and actuating (dead-time) is 0.5 s. If the controller is affected by the clock drift the dead-time increases, influencing the system performance. The clock drift in digital devices is usually around $10^{-5}$ but this value would require several hours of simulation before the delay appears in the system. Using a bigger value of the clock drift, the simulation needs less time and the results are not changed. For this reason the clock drift value has been fixed equal to 0.005 in the controller kernel block.
This setting means that the controller clock is faster than the simulation clock and it gains 5 s every 1000 s.

### *B. Simulation comparison of PI and Predictive PI control*

In this section the performances of a PI controller and a Predictive PI will be compared in order to understand how much improvement a PPI may lead to for the problem of the clock drift. Consider the first order model (15) and a square wave reference with a 50% duty cycle and period of 60 s. The controller parameters are computed following the *lambda tuning* rules. With $\lambda = T = 2$ for this plant the resulting setting is $K_c = 4$, $T_I = 2$. In this specific case has been chosen $K = 0.25$, $\lambda = T = T_I = 2$ and $T_s$=1.

In case the controller suffers from clock drift, the performance decreases as it is shown in figure 9, where it is compared to the case without drift.

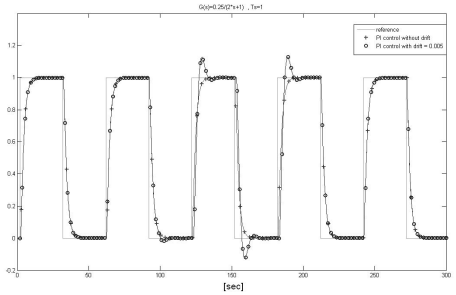


Fig. 9. Reference tracking using the PI controller both with drift and without

In fact, as it is explained in Section IV-A the drift of the clock causes a delay of one cycle (in this scenario 1 s) when the control signal is computed after the actuation so the dead-time becomes equal to 1.5 s instead of 0.5 s. Looking at figure 9, it is possible to notice an overshoot in the reference tracking when the system is affected by the extra delay. This overshoot disappears when the drift is equal to one cycle and the actuator task executes again after the controller.

Looking at figure 13, it is possible to notice that when the system is affected by the delay, the step response of the PPI is better than the one of the PI. However, it is of course worse when the system has no extra delay, since the PPI is implemented considering a fixed delay of 1 s.

## VII. A HYBRID PPI CONTROLLER

To increase the performance of the controller, it is possible to notice that the control signal of the PPI (9) in case of no delay (D=0) is equal to the PI control signal. Looking at (11) and considering the PPI implementation (14) the hybrid controller can be described with the following equations:

$$u_k = \begin{cases} K_c e_k + z_k, & \text{if D=0;} \\ K_c e_k + z_{k-1}, & \text{if D=1.} \end{cases} \tag{16}$$

When the system is not affected by the delay, the controller is a normal PI. Then, when the drift of the clock causes a delay, the control switches to the Predictive PI. Like it is shown in figure 10, the controller can be described as a finite state machine with two states: one state is D=0 (no delay) and the controller is a normal PI, the other state is D=1 (delay of one cycle) and the controller is the Predictive PI. The status changes depending on a variable that depends on the drift.

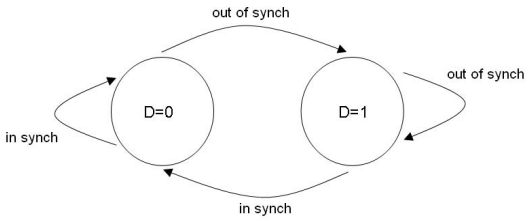


Fig. 10. Finite state machine: D=0 no delay, D=1 delay of 1 cycle

*1) Detection of the delay due to the drift:* The detection of the exact instant in which the drift introduces the delay is fundamental for the correct operation of the finite state machine. Consider a time stamp in the sensor data, $t_s$, and to compare it with the instant in which the gateway transmits the control signal to the actuator ($t_{GW}$) (see Fig. 11).
Notice that all the times are referred to the internal clock of each device. Considering these notations the detection of the delay can be explained with the following equations:

$$D = \begin{cases} 0, & \text{if } t_{GW} - t_s < t_d; \\ 1, & \text{else.} \end{cases}$$

This solution permits to detect the presence of the delay in the system. In this way the transition between D=0 and D=1 is detected a cycle later. This is due to the fact that when

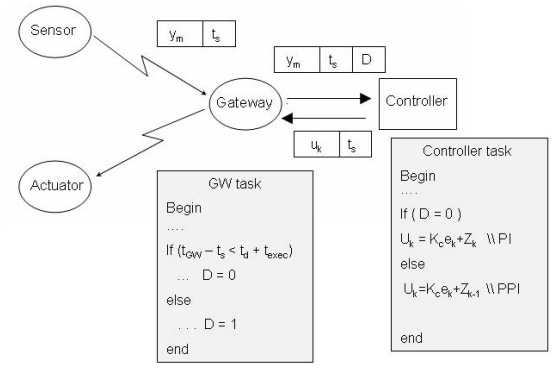


Fig. 11. Scheme of the detection of the delay due to the drift

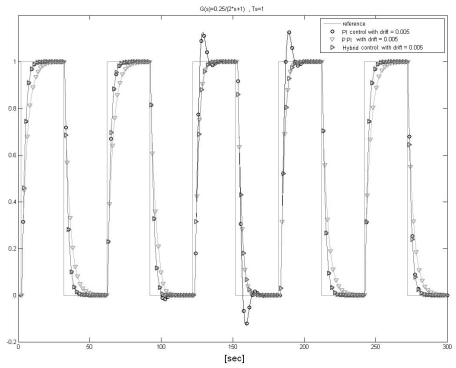


Fig. 13. Performance comparison between the hybrid controller, the PPI and the PI.

the gateway executes, the controller has already done the computation for that cycle. An improvement to this law is obtained considering the trend of the difference $t_c - t_s$.

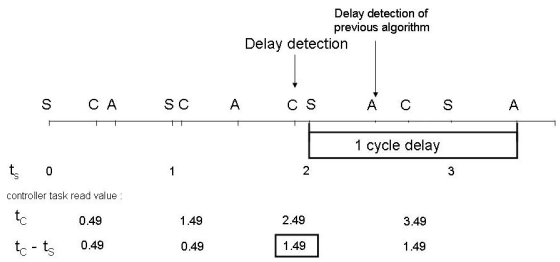


Fig. 12. Detection of the delay due to the drift improvement, an example.

Fig. 12 illustrates an example in which the controller clock is faster than the WirelessHart network one. The difference between $t_c$ and $t_s$ is constant and changes only in the instant when the drift introduces the delay. Defining:

$$\Delta_{cs}^{t} = t_c - t_s \tag{17}$$

the rule to switch between the two states becomes:

$$D = \begin{cases} 0, & \text{if } t_{GW} - t_{sc} < t_d; \\ 1, & \text{if } \Delta_{cs}^{t} \neq \Delta_{cs}^{t-1}. \end{cases}$$

Considering (16) the hybrid controller can be described by the following equations:

$$u_k = \begin{cases} K_c e_k + z_k, & \text{if } t_{GW} - t_s < t_d; \\ K_c e_k + z_{k-1}, & \text{if } \Delta_{cs}^{t} \neq \Delta_{cs}^{t-1}. \end{cases} \tag{18}$$

In this way the transition between the PI and PPI happens immediately without loosing any cycle. In figure 13 the performance of the hybrid controller is compared with the behaviour of the previous implementation of the PPI and with the performance of a normal PI.

## VIII. Conclusions

In this work we have investigated the impact of clock drift on control performance in WirelessHART network where no synchronization exist between the controller and the wireless network. The obvious solution to avoid clock drift is to have synchronized controller and wireless network. When this is not possible our work show that detecting when you are out of synch and using a predictive controller can significantly reduce the problem caused by clock drift.

## IX. Acknowledgments

The authors would like to thank the European Commission and the partners of the European IST FP6 project (SOCRADES - www.socrades.eu), for their support.

## References

[1] HART communication foundation, *TDMA Data Link Layer,* $HCF\_SPEC - 075$ *Revision 1.0*, 30 August 2007.
[2] HART communication foundation, *Wireless Devices Specification,* $HCF\_SPEC - 290$ *Revision 1.0*, 5 September, 2007.
[3] HART communication foundation, *Network Management Specification,* $HCF\_SPEC - 085$*, Revision 1.0*, 27 August, 2007.
[4] Industrial$^{IT}$ 800xA - System ( version 5.0) - *System guide: Technical data and configuration information*- ABB Automation Technology Products
[5] Industrial$^{IT}$ 800xA - Control and I/O ( System version 5.0) - *Application programming introduction and design*- ABB Automation Technology Products
[6] M. Nixon and T. Blevins, *Process Control Requirements for Wireless Communication*, Report from Emerson Process Management, submitted to HCF Wirelesss Hart Working Group, 2007.
[7] M. Ohlin, D. Henriksson, A. Cervin, Department of Automatic Control Lund University - *TRUETIME 1.5-Reference Manual* - January 2007.
[8] R. Tjoa, K.L Chee, P. K. Sivaprasad, S.V. Rao and J.G. Lim - *Clock Drift Reduction for Relative Time Slot TDMA-based Sensor Networks* - PIMRC 2004, 15-th IEEE symposium, Volume 2, pages 1042-1047, 2004.
[9] K. J. Åström, T. Hägglund, *Advanced PID control* - ISA, 2006