SIMULATION OF NETWORKED CONTROL SYSTEMS USING TRUETIME

Ľ. Farkas, J. Hnát***

* Systémy priemyselnej informatiky s.r.o., Kopčianska 14, 851 01 Bratislava, Slovak Republic, www.syprin.sk

** Institute of Control and Industrial Informatics, Faculty of Electrical Engineering and Information Technology, Ilkovičova 3, 812 19 Bratislava, Slovak Republic

Abstract

The installation of industrial fieldbuses in new industrial applications and plants calls for new approaches to the designing of the control system. The modern networked control systems are usual decentralized systems interconnected by industrial network cables or wirelessly. In this paper we describe networked control in industrial applications and the possibility to simulate the control systems by TrueTime.

1 Networked Control Systems

Nowadays we can see the trend of passing from the traditional centralized control to distributed control systems. This calls for a change of the usual design approaches to the control systems. The traditional automation is fusing with the technologies known from informatics and computer networks [1]. According to [2], a Networked Control System (NCS) is a distributed control structure where the communication between the nodes of the control system is provided by a communication network. The basic elements of a NCS are sensors, controllers, actuators and the communication network.

A NCS can be a one-level or a multi-level NCS. The one-level NCS is a system where the communication nodes are interconnected in one level of the whole manufacturing process of the company. An example of a one-level NCS is in Fig. 1. Multiple interconnected levels of the company's manufacturing process form a multi-level NCS. There are the fieldbuses on the bottom levels of the process and the company's LAN on the higher levels, where the data from the industrial layer are handled for economical or technological reasons.

The reasons for using a network infrastructure similar to computer networks are exactly the same as for using networks in buildings. To these reasons belongs the relatively cheap installation cost, use of standardized hardware, transparent infrastructure, ability to expand the system in the future, or a good resistance to interferences. The use of a network is very cost effective.

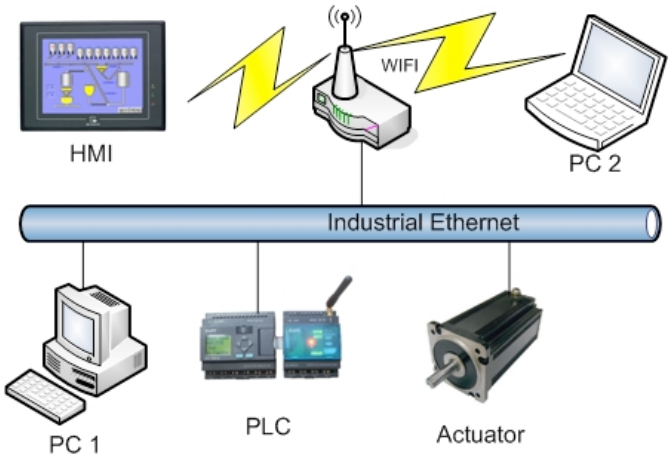

Figure 1: The bottom level of the company's manufacturing process (Hybrid NCS)

According to the communication point of view, the NCSs can be divided into straight and hierarchical [1]. In a straight NCS, the control loop consists of a controller, sensors and actuators connected through a network. In the hierarchical NCS the system consists of local straight control

loops connected to a superior controller. This type of NCS is used in more complicated systems, like in mobile robotics.

The nodes in the NCSs are either connected by cables (fieldbuses, Ethernet) or wirelessly (ZigBee, iWLAN).

1.1 Wired NCS technologies

The most common, most reliable and most secure way of connecting nodes communicating over an industrial network is by cables. There are several technologies for use in industrial applications. Each of them is supported by a number of device manufacturers. As we have presented, the benefits of the networks are mainly the cost effectiveness and reliability. Each technology respects these needs. The main differences are in the transfer rates and the communication protocols.

The older technologies like the CAN bus or Profibus were designed from scratch for the industrial needs (reliability, resistance to interferences) and their limits are unveiling in the last years, e.g. the maximum possible number of connected nodes, the maximum transfer rate. Because these limitations, there has been an initiative to use the existing Ethernet technology used in LANs. The Ethernet is perspective because of its high transfer rate, ease of installation, cheap hardware (compared to the hardware used in existing fieldbuses) and resistance to interferences. Some of the hardware manufacturers have been developing their own industrial implementation of the Ethernet. Examples of the implementations are Profinet by Siemens, EtherCAT by Beckhoff or Ethernet Powerlink by B&R. The differences of the technologies are especially in their transfer protocols and the medium access. Their common characteristic is the physical layer (cables, switches). An example of an EtherCAT network topology is in Fig. 2.

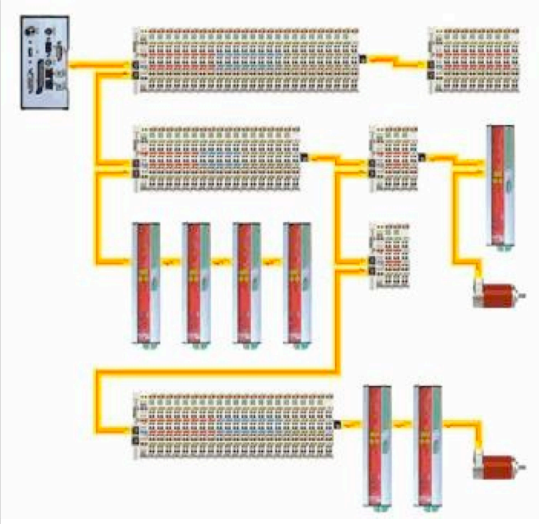

Figure 2: Example of an EtherCAT topology [3]

1.2 Wireless NCS technologies

Following the successful implementation of wired Ethernet technologies in industry the manufacturers have been developing also the industrial implementation of the wireless data transfer technologies. The reason for the use of wireless data transfers is similar to the use of fieldbuses and it is the cost effectiveness in some applications. The benefits are reduced installation cost, lower operating cost, installation flexibility, and scalability [4]. A proper industrial application and a proper wireless technology must be chosen for the right control behavior of the system. Factors influencing the functionality of the wireless system are for example the environment interference or the safety of the data transfer.

The typical industrial wireless technologies are ZigBee and industrial WLAN (an industrial modification of the 802.11 b/g standard). According to [5] the wireless technologies bring to users low cost access to additional measurements that would otherwise be too expensive to be installed. These

additional measurements sometimes contribute to increasing of the safety of the plant. According to [6] the wireless applications in industry can be used in the fields of

- Plant safety
- Increasing the reliability of the plant
- Optimization of the production process

1.3 The need for co-simulation during NCS development

Computer-based control systems and networked control systems are hybrid systems where continuous time-driven dynamics and discrete event-driven dynamics interact. The temporal non-determinism introduced by computing and communication in the form of delays and jitter can lead to significant performance degradation. Software tools are needed to analyze and simulate how the timing affects the control performance [7].

The wireless components of the industrial system are often moving around in the environment or a sensor is monitoring the presence of moving objects in the environment. These systems interact with their environment. These applications are usually very complex and a simulation tool during their development is very helpful.

It should be possible to simultaneously simulate the computations that take place within the nodes, the wired and wireless communication between the nodes, the sensor and actuator dynamics, the dynamics of the mobile robots, and the dynamics of the environment, including the physical systems under control [8].

2 TrueTime

TrueTime is a Matlab/Simulink-based simulator for networked and embedded control systems that has been developed at Lund University since 1999. The simulator software consists of a Simulink block library (Fig. 3) and a collection of MEX files. The kernel block simulates a real-time kernel executing user-defined tasks and interrupt handlers. The various network blocks allow nodes (kernel blocks) to communicate over simulated wired or wireless networks [9].

According to [8] the TrueTime blocks are connected with ordinary Simulink blocks to form a real-time control system. The main feature of TrueTime is the possibility of co-simulation of the interaction between the real-world continuous dynamics and the computer architecture in the form of task execution and network communication.

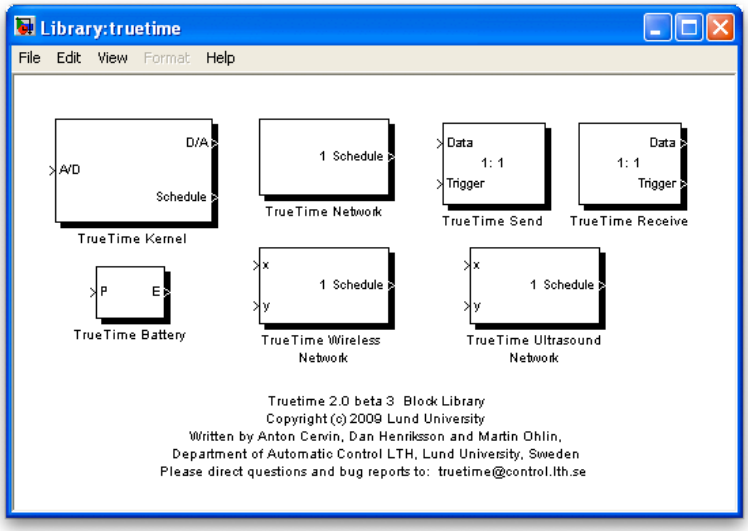

Figure 3: The TrueTime block library

The block library consists of the TrueTime Kernel block that simulates a real-time kernel executing user defined tasks and interrupt handlers, the Network block that allows nodes to communicate over simulated network, a couple of standalone interface blocks and of the Battery block that allows modeling of battery driven operation. The blocks are variable-step, discrete, MATLAB S-functions written in C++.

Before a simulation can be run, however, it is necessary to initialize kernel blocks and network blocks, and to create tasks, interrupt handlers, timers, events, monitors, etc for the simulation. The initialization code and the code that is executed during simulation may be written either as Matlab M-files or as C++ code.

2.1 The TrueTime Wired/Wireless Network Blocks

The TrueTime network block simulates medium access and packet transmission (physical and medium access layer) in a local area network. When a node tries to transmit a message, a triggering signal is sent to the network block on the corresponding input channel. When the simulated transmission of the message is finished, the network block sends a new triggering signal on the output channel corresponding to the receiving node. The transmitted message is put in a buffer at the receiving computer node [10].

The types of networks supported are CSMA/CD (Ethernet), CSMA/AMP (CAN), Round Robin (Token Bus), FDMA, TDMA (TTP), Switched Ethernet, WLAN (802.11b), and ZigBee (802.15.4). There is also a TrueTime Ultrasound network support in the TrueTime 2.0 beta version.

The network blocks are configured by their block mask dialogues. There are some common parameters for all networks like the network number, number of nodes, data rate or minimum frame size. The other parameters are network type specific like transmit power or receiver signal threshold in wireless networks. Wired and wireless network block masks are in the Fig. 4.

There can be a number of network blocks in a model. That is why each network is identified by an id number. A number also identifies the connected nodes. This node id number has to be network specific.

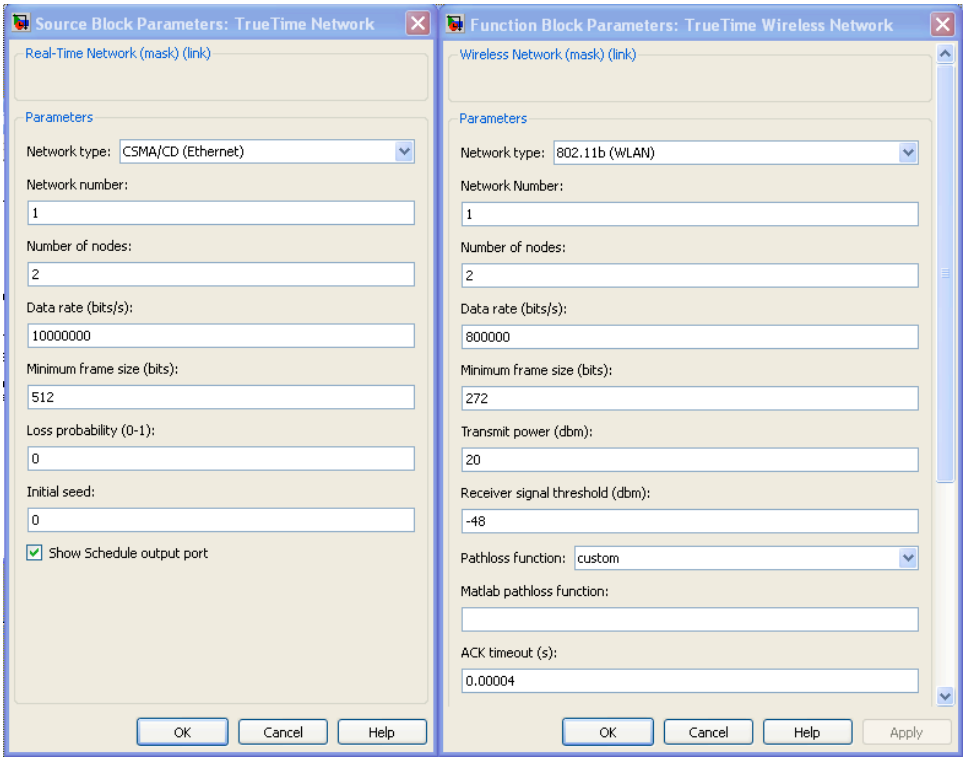

Figure 4: The TrueTime Wired and Wireless Network block masks

2.2 The TrueTime Kernel Block

The kernel block is a Simulink S-function that simulates a computer with a real-time kernel, A/D and D/A converters, a network interface, and external interrupt channels. The kernel executes user-defined tasks and interrupt handlers. Internally, the kernel maintains several data structures that are commonly found in a real-time kernel: a ready queue, a time queue, and records for tasks, interrupt handlers, monitors and timers that have been created for the simulation [11].

The block is configured by the block mask dialog (Fig. 5). The main parameter is the name of the initialization function, because each kernel block has to be initialized at the start of the simulation. An optional argument for the initialization script can be set, also the battery option, the clock drift and the clock offset can be set.

The initialization script, the task code and the interrupt handler code may be written either as m-file script or C++ functions. Control algorithms may also be defined graphically using ordinary discrete Simulink block diagrams [7]. An example initialization and task code will be listed below.

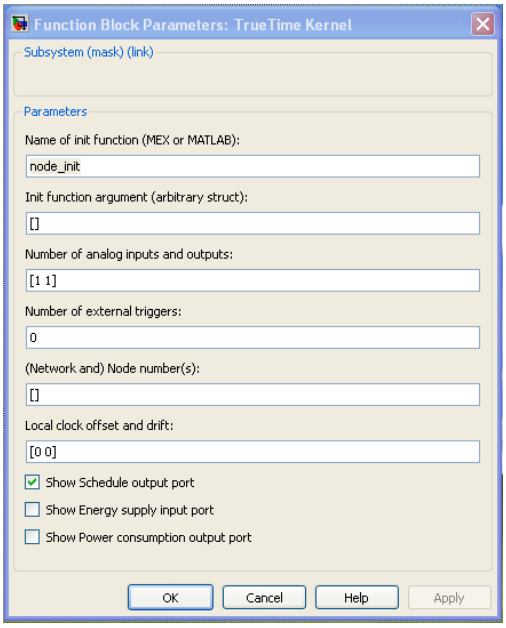

Figure 5: The TrueTime Kernel block mask

2.3 The TrueTime Standalone Network Interface Blocks

The standalone network interface blocks (TrueTime Send, TrueTime Receive) can be used to send messages through the network (using network blocks) without using kernel block. This means that no initialization code or task code must be written. According to [10] the whole network simulation can be created in Simulink without using any m-files, or C++ code.

It is possible to mix the standalone blocks with kernel blocks in a simulation. It means that some stations can send messages without m-file task codes (e.g. sensors) and some stations use kernel blocks (controllers).

The Send and Receive blocks are configured through their block mask dialogs (Fig. 6). The send block can be time-triggered or event-triggered. The trigger input port can be configured to trigger on raising, falling or either flanks.

2.4 The TrueTime Battery Block

The battery block acts as a power source for the battery enabled kernel blocks. It uses a simple integrator model so it can be both charged and recharged. The only one parameter in its block mask is the initial power. To use the battery, enable the check box in the kernel configuration mask and connect the output of the battery to the E input of the kernel block.

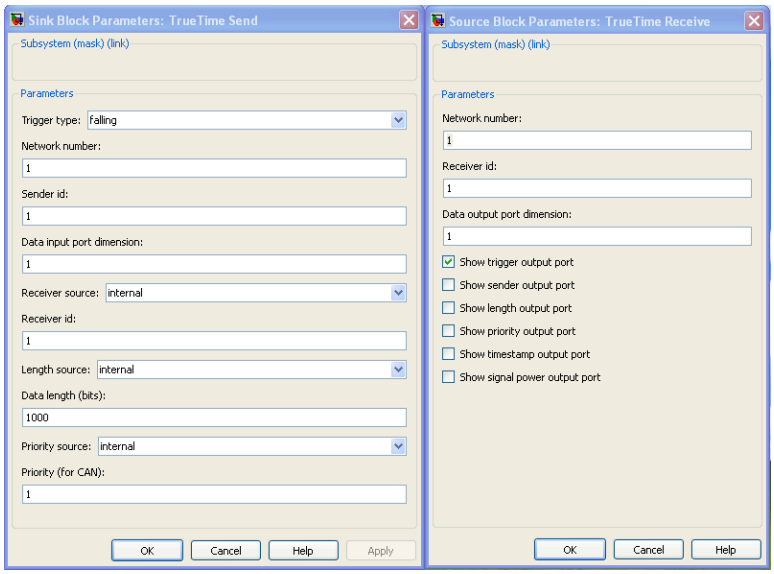

Figure 6: The TrueTime Send and Receive block masks

3 NCS examples simulated in TrueTime

We will demonstrate a few examples based on the simulation models from the TrueTime example directory and from [10]. We would like to show how to simulate a wired and a wireless NCS with the use of the standalone network interface blocks (the network message senders/receivers) on the controlled system side and also on the controller side. The third example will be a mixed environment with a kernel block on the controller side and standalone interface blocks on the controlled system side. This will be a wired NCS example.

3.1 The wired/wireless NCS with standalone interface blocks

The simulation models are in Fig. 7 and Fig. 8. The difference between the models is the network media type. The controlled system and the controller parameters are the same.

The wireless network block has 2 extra input ports. These ports are the x and y coordinates of the nodes to specify their location and compute the path-loss of the radio signal. With the use of standalone network interface blocks we must not use the initialization and task codes. This simplifies the making of some simple network simulations. Our simulation models consist of a DC motor controlled by a discrete PID controller interconnected by the CAN fieldbus or wirelessly by the 802.11 g standard.

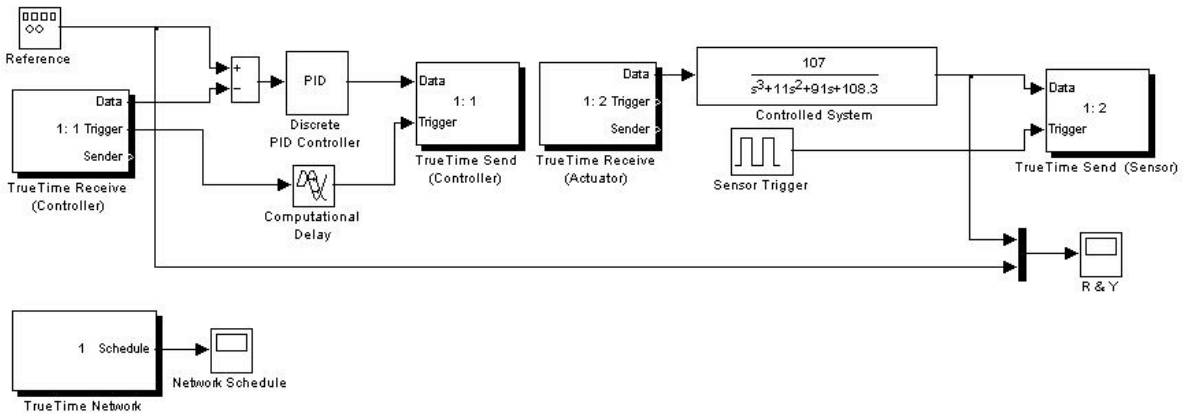

Figure 7: Wired NCS simulation model with standalone interface blocks

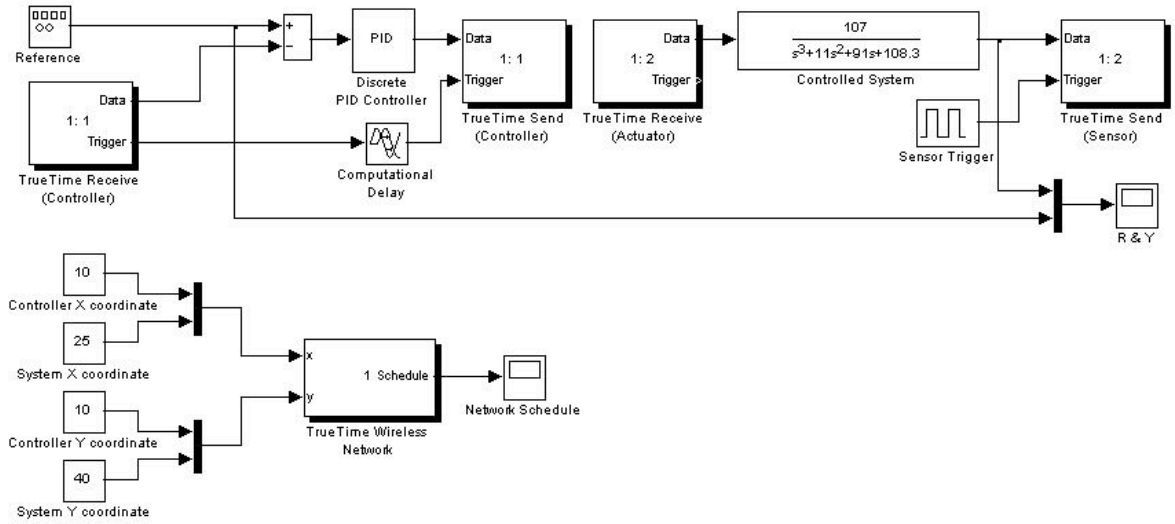

Figure 8: Wireless NCS simulation model with standalone interface blocks

The DC motor is described by the transfer function (1) in both cases. It is controlled by a discrete PID regulator with the parameters P = 1.2, I = 1.4, D = 0.1 and with the sample time 0.01 s. We control the rpm of the motor represented by a voltage value of the sensor, so the reference value is also represented in Volts. The simulation graph of the process values together with the reference values and the controller outputs are in Fig. 9. The simulation results are identical for both simulations and we can see, that the approach with the standalone interface blocks is good for modeling of simple control strategies and sensor data acquisition. A close-up of the network traffic (network schedule) of the wireless network is shown in Fig. 10.

$$G(s) = \frac{107}{s^3 + 11s^2 + 91s + 108.3} \tag{1}$$

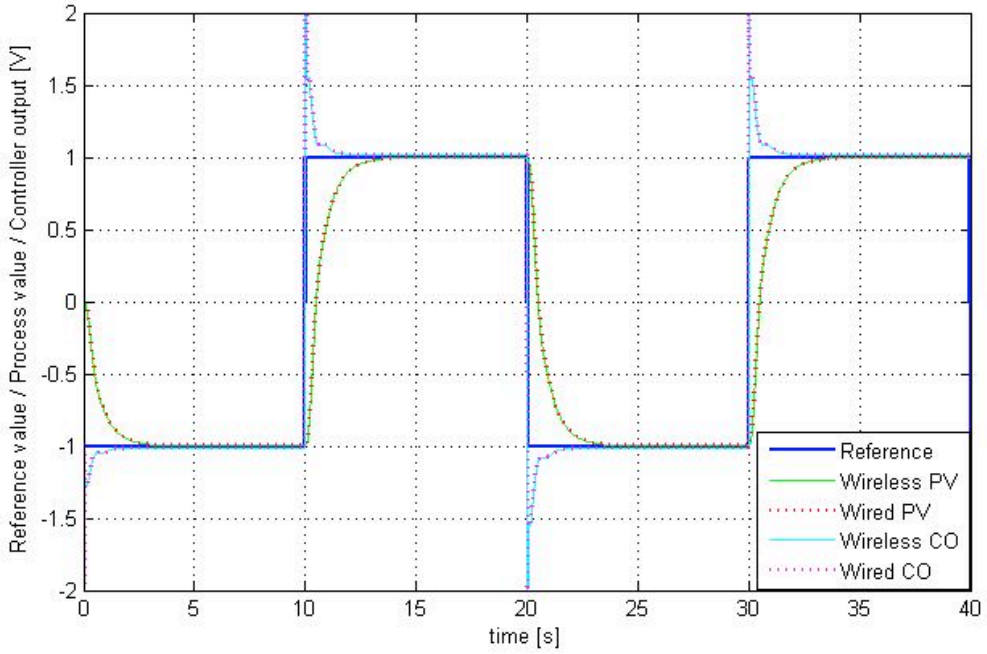

Figure 9: Wired and wireless NCS simulation results

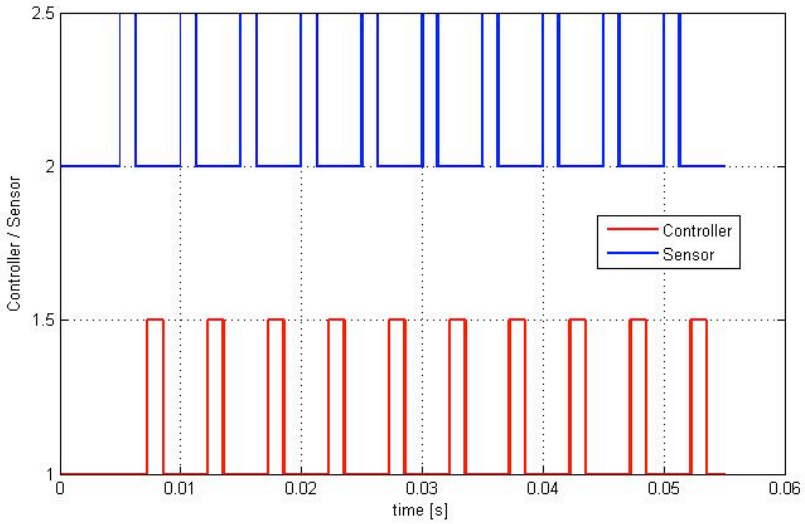

Figure 10: Wireless NCS simulation network schedule

3.2 Wired NCS simulation with kernel block and standalone blocks

We use a combination of a kernel block as the controller and standalone interface blocks as the sensor and actuator in this simulation example. We would like to show the possibility to mix the different kind of blocks and to show the basic concept of writing the initialization code and the controller task code. The controlled system is the same DC motor as in the previous examples. The blocks are interconnected by the CAN fieldbus. The simulation model is in Fig. 11.

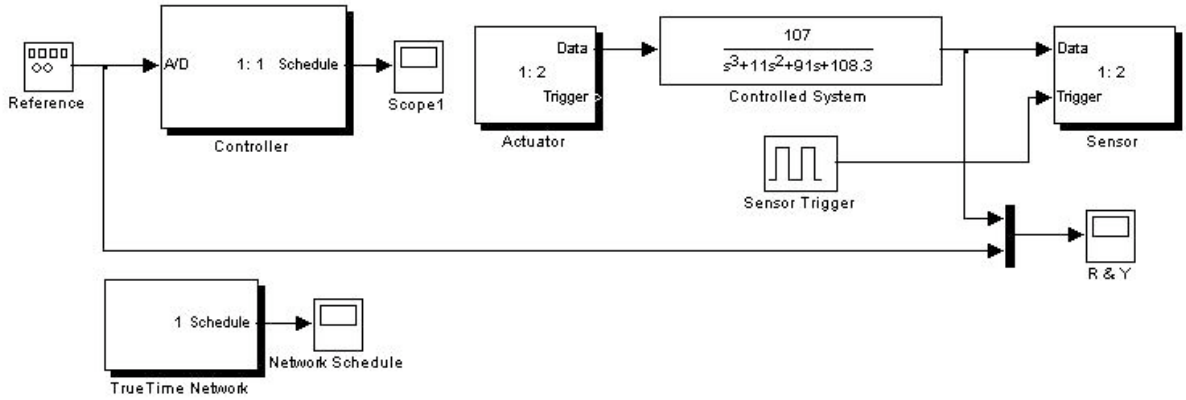

Figure 11: Wired NCS with a kernel block and standalone blocks

We have to write an initialization function and also a task code if we use a kernel block in the simulation. The initialization script defines the scheduling policy, creates tasks, interrupt handlers, events, monitors, etc. for the simulation. In our case the initialization script is written for the kernel block representing the controller. The script is shown in Listing 1. The kernel is initialized by providing the scheduling policy using the function *ttInitKernel* and by creating a periodic controller task using the function *ttCreatePeriodicTask.* The name of the task is *controller_task* and its task code is *controller_code*. The other parameters are the controller parameters in a structure passed to the task and the task period. As the task is periodic, we don't need to define an interrupt handler.

Listing 1: The initialization script for the controller network kernel block

```
function controller_init(arg)
ttInitKernel('prioFP')
% Create task data
data.MPn = 0;
```

```
data.MIn = 0;
data.MDn = 0;
data.MX = 0;
data.h = 0.010;
data.K = 1.2;
data.Ti = 1.1;
data.Td = 0.4;
data.yold = 0;
data.u = 0;
% Periodic controller task
period = 0.01;
ttCreatePeriodicTask('controller_task', period, 'controller_code', data);
```

The execution of the task is defined by the code function from the initialization script. It is the *controller_code* in our case. The code is shown in Listing 2. It is a function, which obtains the process value from the network, reads the reference value from the analog input, computes the controller output and sends it through the network to the actuator.

Listing 2: The discrete PID controller task code

```
function [exectime, data] = controller_code(seg, data)
switch seg
case 1
  y = ttGetMsg;          % Obtain sensor value

if isempty(y)
    disp('Error in controller: no message received!');
    y = 0.0;
end
  r = ttAnalogIn(1);     % Read reference value
  data.MPn = data.K * (r - y);
  data.MIn = data.K * data.h/data.Ti * (r - y) + data.MX;
  data.MDn = data.K * data.Td/data.h * (data.yold - y);
  data.u = data.MPn + data.MIn + data.MDn;
  data.MX = data.MIn;
  data.yold = y;
  exectime = 0.0005;
 case 2
  ttSendMsg(2, data.u, 100);  % Send the value to the actuator
  exectime = -1;              % finished
end
```

The close-up of the network schedule is in Fig. 12. The simulation results are in Fig. 13. The rpm is again represented by voltage. From the results we can see, that the approach of combining kernel blocks with standalone blocks is also appropriate for simulations. The sensors can be represented by standalone blocks because of no need for initialization or task codes and the computers running more complicated control tasks can be represented by kernel blocks.

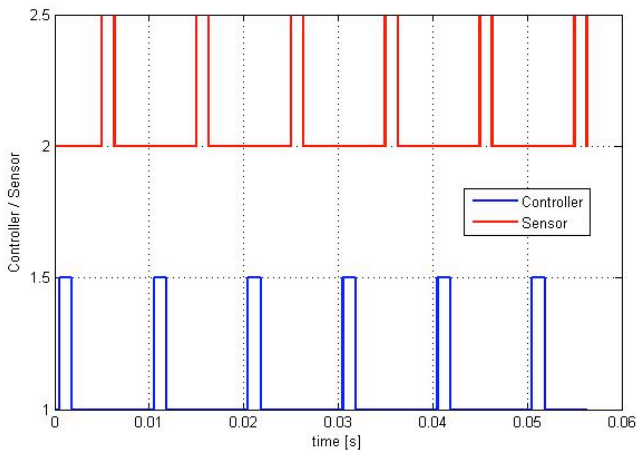

Figure 12: Wired NCS network schedule

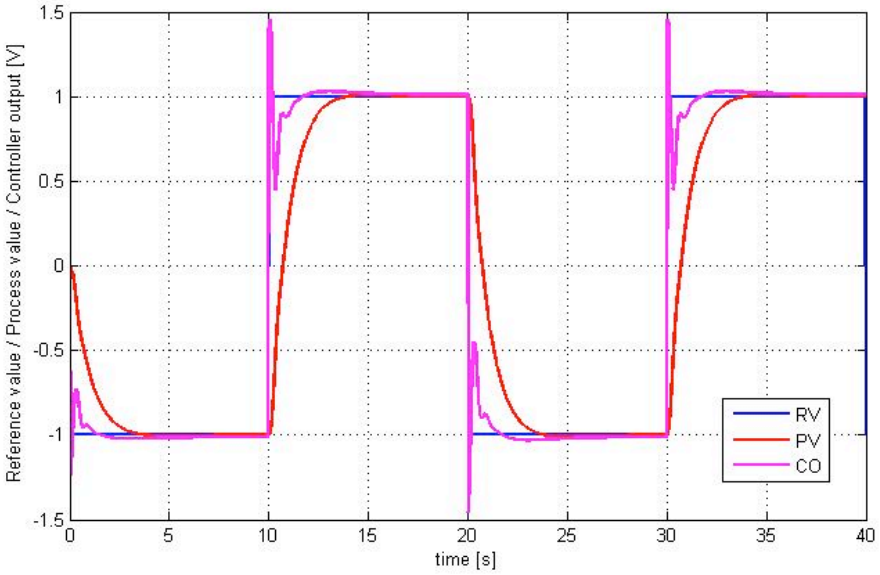

Figure 13: Simulation results of a NCS with a kernel block and standalone blocks

4 Conclusion

In this paper we described the Networked Control Systems, their characteristics and possibilities. We described the wired and also the wireless systems. A simulation tool is helpful during the development of NC systems because they are usually very complex. This is for example an environment with wirelessly communicating moving robots or a mixed wired-wireless environment.

The name of the tool is TrueTime and it is Matlab/Simulink based. The TrueTime blocks connected with ordinary Simulink blocks form a real-time control system. We described the individual TrueTime blocks and their setup. Then we made some example simulations with only standalone interface blocks communicating through wired and wireless network and after that an example with a kernel block communicating with standalone blocks. In the last example we have shown the writing of the initialization and the task code of a periodic task.

From the simulation results we can see, that the standalone blocks are appropriate for sending/ receiving data without additional handling (sensors, actuators) and the kernel blocks are appropriate for simulating computers with a generic real-time kernel, A/D and D/A converters and network interfaces executing more complicated data handling.

5 Acknowledgments

This work has been supported by the Slovak Research and Development Agency under the contract APVV-0337-37. It has been also supported by the Scientific Grant Agency VEGA under the contract 1/0544/09.

References

[1] M. Foltin, J. Murgaš. *Sieťové riadenie procesov – formulácia a trendy*, Elosys ’07, Trenčín 2007.
[2] M. Jamshidi. *Large-scale systems: modeling, control and fuzzy logic*, Prentice Hall, 1997
[3] *EtherCAT - Principle of operation* [online], http://www.beckhoff.com/, Beckhoff Automation GmbH, 2008.
[4] J. Slipp et al. *WINTeR: Architecture and Applications of a Wireless Industrial Sensor Network Testbed for Radio-Harsh Environments*, Communication Networks and Services Research Conference, 2008, 422-431, ISBN 978-0-7695-3135-9

[5] M. Menezes. *Bezdrôtové technológie a najlepšie riešenia na zníženie nákladov projektu*, In AT&P Journal 7/2008, 16 - 18
[6] Honeywell. *Honeywell Industrial Wireless Solutions*, [online] cit. 24.10.2009, available at: http://hpsweb.honeywell.com/Cultures/en-US/Products/wireless/default.htm
[7] D. Henriksson et. al. *TrueTime Simulation of Networked Computer Control Systems*, Preprints of the 2nd IFAC Conf. on Analysis and Design of Hybrid Systems (Alghero, Italy), 7-9 June 2006
[8] M. Andersson. *Simulation of Wireless Networked Control Systems*, Proceedings of the 44th IEEE Conference on Decision and Control, and the European Control Conference 2005, Seville, Spain, December 12-15, 2005, 476 – 481
[9] A. Cervin et. al. *Simulation of Networked Control Systems Using TrueTime* [online], cit. 24.10.2009, available at: http://www.control.lth.se/documents/2007/cer+07ncs.pdf
[10] A. Cervin et. al. *TrueTime 2.0 beta – Reference Manual*, Department of Automatic Control, Lund University, January 2009
[11] A. Cervin et. al. *Control Loop Timing Analysis Using TrueTime and Jitterbug* [online], cit. 23.10.2009, available at: http://www.md.kth.se/RTC/ARTIST2/publications/cervin+CACSD06.pdf

---

Ľudovít Farkas
ludovit.farkas@syprin.sk

Juraj Hnát
juraj.hnat@stuba.sk