Chapter 3

Issues on Communication Network Control System Based Upon Scheduling Strategy Using Numerical Simulations

Oscar Esquivel-Flores, Héctor Benítez-Pérez and Jorge Ortega-Arjona

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/48578

1. Introduction

Nowadays, industry has successfully used Network Control Systems (NCS) therefore several lines of research have arisen. A NCS is a current application of a Real-Time Distributed Systems (RTDS), composed of a number of nodes capable of developing a complete control process. In these systems several nodes exchange information through a communication network to achieve specific control goals, nevertheless network traffic increases. This affects the overall system performance. Several approaches have been developed to satisfy requirements of both control and communication performance. Particularly, some methodologies focus on saving bandwidth, one of such methodologies is network scheduling. The objective of this methodology is the accurately use of the computing resources. NCS research is categorized into two main parts [5]:

1. *Control of network:* Study and research on communications and networks to make them suitable for real-time NCS, e.g. routing control, congestion reduction, efficient data communication, networking protocol.
2. *Control over network:* This area deals with control strategies and control systems design over the network trying to minimize the effect of adverse network parameters on NCS performance such as network delay.

These systems have many challenges to maintain the the Quality of Service (QoS) and Quality of Control (QoC). In the networks, QoS is the idea that transmission rates, error rates, and other characteristics can be measured and improved. The QoS can be degraded due to congestion and interference.

1.1. Architecture and overview of a NCS

There are two general NCS configurations, *Direct structure* and *Hierarchical structure* [14]. Direct structure consists of a controller and a remote system containing a physical plant, sensors and actuators. The controller and the plant are physically located at different points and are directly linked by a data network in order to perform remote closed-loop control. Fig. 1 presents a schematic diagram of a direct structure of a NCS [15].

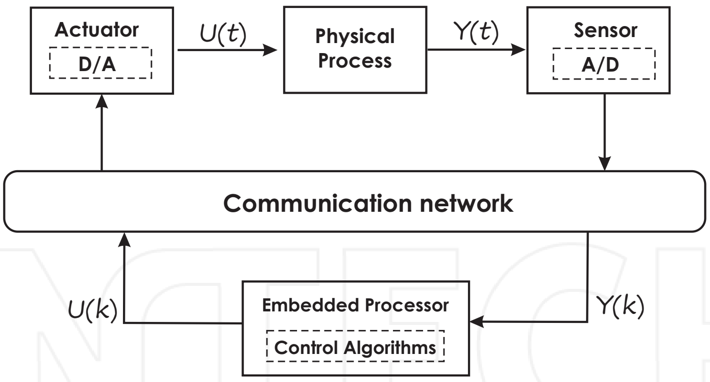

Figure 1. Schematic diagram of a direct structure of a NCS

1.2. Effect of sampling period on NCS

Network scheduling deals with to elect a sampling rate, aiming to reduce the number of data transmitted over the network . The effectiveness of the control system depends on such a sampling rate [8–10]. A region is acceptable in networked control performance terms if it is contained within two sampling rate boundaries, which can be statistically determined.

The use of a common-bus network architecture and a particular network protocol introduces different forms of time delay uncertainties between, sensors, actuators and controllers [8]. Hence, it is quite important to explore different network protocols and network scheduling strategies, before to implement the RTDS, in order to obtain a desired control performance.

For networked control, the minimum transmission frequency (f_m) is necessary to guarantee good system performance without decreasing the network performance. As the transmission frequency increases the system performance improves; however, the load on the network also increases until a maximum transmission frequency (f_h) is reached, then the system performance decreases because the network performance is overloaded.

This phenomenon represents non-linear situations with respect to sudden changes in state of the network, failure situations, or saturation in the channel or traffic, among others. However; it is possible to propose a linear model in the context of proper use of the network, thereby deferring the modeling of nonlinearity in these systems until future work.

The main objective of this work is to explore several issues on communication network scheduling of a RTDS, besides to implement a particular network scheduling strategy to evaluate its effectiveness using numerical simulations. This is presented using a simulated case study, based upon a 2-DOF helicopter simulation benchmark [13]. This simulation provides an approximation to system response, in which, for demonstration purposes, the main results are obtained for a typical fault scenario. Thus, for this simulation, a scheduling strategies is implemented using TrueTime [1, 2] performing dynamic scheduling. Several researches have focused on control over network, shared-network control systems have special interest.

2. Real-time simulation tool TrueTime

This section gives a brief overview of TrueTime simulator and exposes basic examples to initialize typical TrueTime blocks.

According to Cervin *et al.*, nowadays simple embedded control systems often contain a multi-tasking real-time kernel and support networking, besides time control algorithm and control software designs need to be considered together. Thus new computer-based tools for real time and control design are needed [1]. Networked control loops consist of sensor, actuator and control calculations residing on different nodes; within the individual nodes the controllers are implemented as one or several tasks on microprocessor with a real-time operating system, this operating system typically uses multiprogramming to execute various tasks. Communication bandwidth and CPU time can be considered as shared resources for which the task compete. Different sources of temporal nondeterminism as execution times of the tasks or communications delays affect the control performance, nevertheless this nondeterminism can be reduce by the accurate choice of implementation platforms. The constraints of the implementation platform must be considered in systems with limited computer resources [1], therefore some tools are available to analyze and simulate the effects of temporal constrains affects control performance.

TrueTime [1, 2, 6, 7, 12] is a simulator for networked and embedded control system based on Matlab/Simulink, it has been developed at Lund University since 1999 [2]. TrueTime can be used as an experimental platform for research on dynamic real-time control systems. For instance, it is possible to study compensation schemes that adjust the control algorithm based on measurements of actual timing variations [1]. TrueTime make it possible to study more general and detailed timing models of computer-controlled systems. TrueTime can be used: to investigate how timing nondeterminism affects the system behavior, to develop new outlines to adjust control parameters dynamically, to experiment new approaches as codesign of control and network scheduling and to simulate control systems based on event-driven task [1].

The simulator software consists of a Simulink block library, the kernel block simulates a Real-Time kernel executing user-defined task and interrupt handlers. To communicate kernel blocks (nodes) several network blocks may be used, thus it makes quite simple to develop networked control system simulations.

2.1. The kernel block

A computer node is simulated using TrueTime kernel block, this node has a generic real-time kernel, A/D and D/A converters, and network interfaces. An initialization scrip is used to configure the block, in this script it is possible to create several objects as task, timers, interrupt handlers, semaphores, etc., this objects establish the software executing in the computer node. The kernel continuously calls the code functions of the tasks and interrupt handlers. Either Matlab m-files code or C++ language may be used to write initialization scripts and the code functions, the main advantage to use C++ is the speed, nevertheless m-file code is very easy to use. Several scheduling policy is able to use in TreTime kernel block, these can be fixed-priority scheduling and earliest-deadline-first scheduling and custom scheduling policies [2].

The *task* is the main construction in the TrueTime environment, this object is used to simulate periodic and aperiodic activities, for example controller and I/O tasks can be periodic and communication and event-driven controller can be aperiodic tasks. A set of attributes and a code function define a task; attributes as name, release time, worst-case execution time, budget, relative and absolute deadlines, priority (if fixed priority scheduling is used), period (if the task is periodic). Release time and absolute deadline are attributes constantly updated bye the kernel during simulation, while period and priority are kept constant, although can be changed by callas to kernel primitives trough execution [1]. An example of the definition of a task is shown below:

```
function sensor_init(arg)
% Initialize TrueTime kernel
  ttInitKernel(1, 0, 'prioFP'); %Inputs,Outputs,FixedPriority
% Create sensor task
  offset = 0;
  prio = 1;
  period = 0.010;
  ttCreatePeriodicTask('sens_task', offset, period, ...
  prio, 'senscode', data);
```

The kernel primitive `ttInitKernel()` initializes a sensor node. The kernel is initialized by specifying the number of A/D and D/A channels and scheduling policy. The built-in priority function `prioFP` specifies fixed-priority scheduling. Rate monotonic `prioRM`, earliest deadline first `prioEDF`, and deadline monotonic `prioDM` scheduling are additional predefined scheduling policies [6].

Interrupts can be generated in two ways: An external interrupt is associated with one of the external interrupt channels of the computer block; when the signal of the corresponding channel changes value the interrupt triggers. The usefulness of this type of interrupt lies to simulate distributed controllers that execute when measurements arrive on the network. Internal interrupts work to construct timers, when a timer expires the interrupt is triggered. A user-defined interrupt handler is scheduled when an external or internal interrupt occurs. An interrupt, as a task, handles works but it is scheduled on a higher priority level. An interrupt handler is defined by name, a priority and a code function [1]. An example of a definition of a interrupt handler is as follows:

```
%Initialize the network
 ttCreateInterruptHandler('nw_handler1', prio, 'msgRcvSensor');
 ttInitNetwork(4, 'nw_handler1'); % node #4 in the network
```

Cervin *et al.* [1, 2] mentions that simulated execution occurs at three distinct *priority* levels: the interrupt (highest priority), kernel and task (lower priority) levels. The execution may be preemptive or non-preemptive. At interrupt level, interrupt handlers are *scheduled* according to fixed priorities. At task level, dynamic-priority scheduling may be used. At each scheduling point, the priority of task is given by user-defined priority function which is a function of the task attributes, this makes it easy to simulate different scheduling policies. Predefined priority functions exist for most of the commonly used scheduling schemes.

The *code* associated with task and interrupt handlers is scheduled and executed bye the kernel while simulation progresses. The code may be divided in segments, which can interact with other tasks and with the environment at the beginning of each code segment. The simulated execution time of each segment is returned by the code function and can be modeled as constant, random, or even data-dependent [1]. During the simulation the kernel saves the current segment and calls the code functions with proper arguments. Execution resumes in the next segment when the task has been running for the time associated with previous segment [7]. An example of a sensor code is given bellow:

```
function [exectime, data] = senscode(seg, data)
switch seg,
 case 1,
  % Receive data from analog input
  data.y = ttAnalogIn(1);
  exectime = 0.0005;
 case 2,
  % Shows the current time
  ttCurrentTime
  % Send message (80 bits) to node 3 (controller)
  ttSendMsg(3, data, 80)
  exectime = 0.0004;
 case 3,
  exectime = -1; % finished
end
```

This function implements a simple sensor node. In the first segment, the plant is sampled using a execution time of .5 ms. In the second segment, the control signal is sent to the controller node. The third segment indicates the end of execution by returning a negative execution time. The structure `data` represent the local memory and is used to store the measured variable between calls to the different segments. The kernel primitives `ttAnalogIn` and `ttAnalogOut` can be perform A/D and D/A conversion. Besides A/D and D/A conversion, a large set of kernel primitives exist which can be called from code function [12].

Monitors and events support *sincronization* between tasks. Monitors are used to guarnatee mutual exclusion when accessing comon data. Events are associated with monitors to represent condition variables [1].

Different *output graphs* are generated by truetime blocks. Each computer block will produce two graphs: A computer graph will display the execution trace of each task and interrupt handler during the simulation. If the signal is high, it means that the task is running. A medium signal indicates that the task is ready but not running, a low signal means that the task is idle. Otherwise a monitor graph shows which tasks are holding and waiting on the different monitors during simulation [1].

2.2. The network block

The TrueTime network block simulates the physical layer and the medium-access layer of several local-area networks. CSMA/CD (Ethernet), CSMA/AMP (CAN), Round Robin

(Token Bus), FDMA, TDMA (TTP), Switched Ethernet, WLAN (802.11b), and ZigBee (802.15.4) are some types of network supported [2]. The network blocks are mainly configured using blocks dialogs. Some parameters common to all types of networks are bit rate, the minimum frame size, and the network interface delay. For each type of network there are some parameters that specifie the number of nodes, data rate (bits/s), preprocessing delay, minimum frame size, maximum frame size, frame overhead, etc. The network blocks may be used having one kernel block for each node in the network. The tasks into the kernels can send and receive arbitrary Matlab structure arrays over the network using `ttSendMsg` and `ttGetMsg` kernel primitives. This way is to quite flexible but requires to program some routines to configure the system. An useful network scheduler viewer shows the network activity for all nodes involved. An overview of all Truetime's primitives can be found on [12].

3. Frequency transmission scheduling

In this section a formal definition of task model is provided and it gives an overview of the control of frequency transmission in a distributed systems and how it impacts over quality performance.

3.1. Task model

Chen *et al.* [3] gives a formal definition of system model of a typical distributed systems consisting of a set of processors and a set of tasks. A distributed system are characterized as follows. A set of processors $\Omega = \{S_1, S_2, ..., Sm\}$ where Ω is the processor set, S_i is the $i - th$ processor and m is the total number of processors. In this model, all processors are assumed to be identical to assure same execution time for each task on different processors. It is also supposed that enough processors are provided. A set of primary copy of real-time tasks $\Phi = \{\tau_1, \tau_2, ..., \tau_n\}$, where $\tau_i = \{c_i, p_i\}$, $i = \{1, 2, ..., n\}$. Here Φ is the set of tasks, τ_i is the $i - th$ task, n is the number of tasks which are periodic, independent and preemptive, c_i denotes the execution time of τ_i, p_i denotes the period of task τ_i.

3.2. Control of frequency transmission

An approach to schedule a real time distributed system based upon modifications on frequency transmission of individual components in the system is presented in [4], this shows that scheduling of a distributed system can be accomplished through modifications on transmission frequencies into a region where the system performance is not affected. A linear time invariant model in which the coefficients of the state matrix are the relations between the transmission frequencies of each agent and through a feedback controller to modify transmission frequencies bounded between maximum and minimum values of transmission. This approach drives the frequency transmission based on three parameters: minimum frequency (f_m), maximum frequency (f_h) and real frequency (f_r). Frequency transmission dynamics can be modeled as a linear time-invariant subsystem which state variables are transmission frequencies of the sensor nodes involved on the system. Note that for each primary task of a sensor S_i, $i = 1, ..., m$ frequency can be expressed as $f_i = 1/p_i$. There is a relationship between nodes' frequencies an external input frequencies which serves as coefficients of the linear system. Therefore it is possible to control the NCS through the input vector u such that the outputs y are nodes' frequencies into a region L bounded by maximum and minimum transmission frequencies, see Fig. 2.

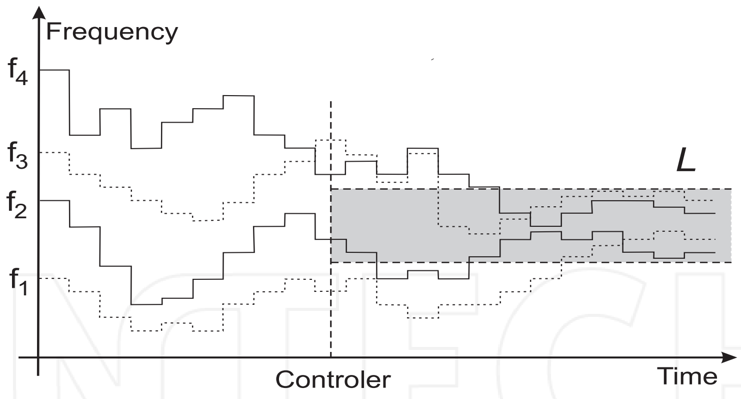

Figure 2. Transmission frequencies bounded by a schedulability region

The objective of controlling the frequency is to achieve coordination through the convergence of values. Each sensor S_i knows its minimum and maximum frequencies based upon messages sent to controller and it could be estimated its own real transmission frequency. Let a NCS with a set $\Omega = \{S_1, S_2, ..., S_l\}$ nodes that performs a set of task $\tau_i = \{c_i, p_i\}$ for $i = \{1, 2, , n\}$, a subset of Ω is sensor nodes subset $\Omega_s = \{S_1, S_2, ..., S_m\}$.

3.3. Network scheduling based on frequency transmission

An approach that modifies the frequency transmission uses f_m, f_r, f_x frequencies. RTDS dynamics, is modeled as a linear time-invariant system, whose state variables are frequencies transmission rates of the n nodes that compose the RTDS [4]. Frequency rates of a node are affected by some external input frequency rates, minimal frequencies of all nodes and particular ratios serve as coefficients of the linear system. So, it is possible to control the NCS using the input vector u, such that the output vector y contains the frequency rates of all nodes within a nonlinear region L, bounded by the maximum and minimum transmission frequency rates. Let we assume that there is a relationship amongst real frequencies $f_r^1, f_r^2, ..., f_r^m$ and external input frequencies $u_1, u_2, ..., u_m$ which serve as coefficients of the linear system:

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k, \\ y_k &= Cx_k. \end{aligned} \tag{1}$$

A, B, C are matrices with appropriate dimensions. A is the matrix of relationships between frequencies of sensor nodes, B is the scale frequencies matrix, C is the matrix with frequencies ordered, x_{k+1} is a real frequencies vector in time $t = k + 1$, y_k is the vector of output frequencies. The input u_k is a function of reference frequencies and real frequencies of the nodes in the distributed system, hence $u = K(f_m - f_r)$ where f_m, f_r are vectors. Then, the state vector in equation (1) can be written as:

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k, \\ x_{k+1} &= Af_{rk} + B\left(K(f_{mk} - f_{rk})\right). \end{aligned} \tag{2}$$

Note that K is the control gain defined as the basics of a LQR algorithm. Matrices A, B and C has the proper dimensions and matrix A include the following restriction [11]:

$$U = \sum_{i=1}^{n} c_i / p_i \leq 1,$$

as a new state of the system (1). For simplicity, frequency transition model will be described using a sensor node subset $\Omega_s = \{S_1, S_2, S_3, S_4\}$. The elements of the matrices for system 2 are defined as follows:

$$\bar{a}_{ij} = \begin{cases} \frac{\Lambda\left(f_m^1, f_m^2, f_m^3, f_m^4\right)}{f_m^i} & i = j \\ \\ \frac{f_m^j}{f_m^i} & i \neq j \end{cases},$$

$$\bar{b}_{ij} = \begin{cases} f_h^i & i = j \\ \\ 0 & i \neq j \end{cases},$$

$$\bar{c}_{ij} = \begin{cases} 1 & i = j \\ \\ 0 & i \neq j \end{cases}.$$

Note that $\Lambda\left(f_m^1, f_m^2, f_m^3, f_m^4\right)$ is the greatest common divisor of minimum frequencies, that is the planning cycle Γ expressed in terms of frequencies of the backup task, for shortening, it will be written only as Λ. f_m, f_h, and f_r are vectors of respective frequencies.

Considering the execution time c_i of each task executing in respective sensor nodes in Ω_s as an additional state, we can rewrite (2) as follows:

$$\begin{bmatrix} x_{k+1}^1 \\ x_{k+1}^2 \\ x_{k+1}^3 \\ x_{k+1}^4 \\ x_{k+1}^5 \end{bmatrix} = \begin{bmatrix} \frac{\Lambda}{f_m^1} & \frac{f_m^2}{f_m^1} & \frac{f_m^3}{f_m^1} & \frac{f_m^4}{f_m^1} & 0 \\ \frac{f_m^1}{f_m^2} & \frac{\Lambda}{f_m^2} & \frac{f_m^3}{f_m^2} & \frac{f_m^4}{f_m^2} & 0 \\ \frac{f_m^1}{f_m^3} & \frac{f_m^2}{f_m^3} & \frac{\Lambda}{f_m^3} & \frac{f_m^4}{f_m^3} & 0 \\ \frac{f_m^1}{f_m^4} & \frac{f_m^2}{f_m^4} & \frac{f_m^3}{f_m^4} & \frac{\Lambda}{f_m^4} & 0 \\ c_1 & c_2 & c_3 & c_4 & 1 \end{bmatrix} \begin{bmatrix} f_r^1 \\ f_r^2 \\ f_r^3 \\ f_r^n \\ x_c \end{bmatrix} + \begin{bmatrix} f_h^1 & 0 & 0 & 0 & 0 \\ 0 & f_h^2 & 0 & 0 & 0 \\ 0 & 0 & f_h^3 & 0 & 0 \\ 0 & 0 & 0 & f_h^n & 0 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$\cdot \left(\begin{bmatrix} K_1 & 0 & 0 & 0 & 0 \\ 0 & K_2 & 0 & 0 & 0 \\ 0 & 0 & K_3 & 0 & 0 \\ 0 & 0 & 0 & K_4 & 0 \\ 0 & 0 & 0 & 0 & K_c \end{bmatrix} \left(\begin{bmatrix} f_m^1 \\ f_m^2 \\ f_m^3 \\ f_m^4 \\ x_c^r \end{bmatrix} - \begin{bmatrix} f_r^1 \\ f_r^2 \\ f_r^3 \\ f_r^n \\ x_c \end{bmatrix} \right) \right),$$

$$\begin{bmatrix} y_k^1 \\ y_k^2 \\ y_k^3 \\ y_k^4 \\ y_k^5 \end{bmatrix} = \begin{bmatrix} 1\,0\,0\,0\,0 \\ 0\,1\,0\,0\,0 \\ 0\,0\,1\,0\,0 \\ 0\,0\,0\,1\,0 \\ 0\,0\,0\,0\,1 \end{bmatrix} \begin{bmatrix} f_r^1 \\ f_r^2 \\ f_r^3 \\ f_r^4 \\ x_c \end{bmatrix}.$$

x_c is a real execution time and x_c^r is a reference execution time.

Thus

$$\begin{bmatrix} x_{k+1}^1 \\ x_{k+1}^2 \\ x_{k+1}^3 \\ x_{k+1}^4 \\ x_{k+1}^5 \end{bmatrix} = \begin{bmatrix} \frac{\Lambda}{f_m^1} & \frac{f_m^2}{f_m^1} & \frac{f_m^3}{f_m^1} & \frac{f_m^4}{f_m^1} & 0 \\ \frac{f_m^1}{f_m^2} & \frac{\Lambda}{f_m^2} & \frac{f_m^3}{f_m^2} & \frac{f_m^4}{f_m^2} & 0 \\ \frac{f_m^1}{f_m^3} & \frac{f_m^2}{f_m^3} & \frac{\Lambda}{f_m^3} & \frac{f_m^4}{f_m^3} & 0 \\ \frac{f_m^1}{f_m^4} & \frac{f_m^2}{f_m^4} & \frac{f_m^3}{f_m^4} & \frac{\Lambda}{f_m^4} & 0 \\ c_1 & c_2 & c_3 & c_4 & 1 \end{bmatrix} \begin{bmatrix} f_r^1 \\ f_r^2 \\ f_r^3 \\ f_r^n \\ x_c \end{bmatrix}$$

$$+ \begin{bmatrix} K_1 f_h^1 (f_m^1 - f_r^1) \\ K_2 f_h^2 (f_m^2 - f_r^2) \\ K_3 f_h^3 (f_m^3 - f_r^3) \\ K_4 f_h^4 (f_m^4 - f_r^4) \\ K_1(f_m^1 - f_r^1) + K_2(f_m^2 - f_r^2) + \\ K_3(f_m^3 - f_r^3) + K_4(f_m^4 - f_r^4) + \\ K_c(x_c^r - x_r) \end{bmatrix},$$

$$\begin{bmatrix} y_k^1 \\ y_k^2 \\ y_k^3 \\ y_k^4 \\ y_k^5 \end{bmatrix} = \begin{bmatrix} 1\,0\,0\,0\,0 \\ 0\,1\,0\,0\,0 \\ 0\,0\,1\,0\,0 \\ 0\,0\,0\,1\,0 \\ 0\,0\,0\,0\,1 \end{bmatrix} \begin{bmatrix} f_r^1 \\ f_r^2 \\ f_r^3 \\ f_r^4 \\ x_c \end{bmatrix}.$$

4. Real-Time Distributed System

This section exposes the RTDS used for implementation purposes, it is a 2-DOF helicopter prototype [13]. The following section briefly introduces and describes this 2-DOF helicopter prototype and its controller design.

The case of study is a prototype of a helicopter system integrated to a CanBus network with two propellers that are driven by DC motors. The front propeller controls the elevation of the helicopter nose about the pitch axis (θ) and the back propeller controls the side to side motions of the helicopter about the yaw axis (ψ). The pitch and yaw angles are measured using high-resolution encoders. A brief description of the helicopter model is presented, however detailed information can be found in [13]. The dynamics of the helicopter is developed based on kinetic and potential energy, this model is used to design a position controller. The helicopter centre of mass is described in xyz cartesian coordinates with respect the pitch and yaw angles, see Fig. 3.

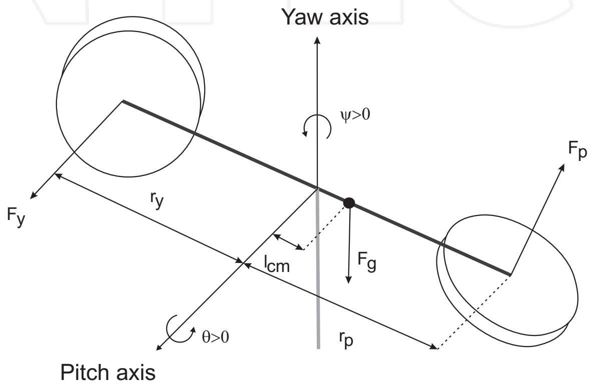

Figure 3. Dynamics of the 2DOF helicopter

The Euler-Lagrange equations are used to obtain nonlinear equations of motion for the 2 DOF Helicopter, which are used to derive the linear state model, and subsequently, to design the position controller. As the helicopter represents a non-linear system, it is required to perform a linearization around a point, this linearization point is

$$\left[\theta_0 = 0, \psi_0 = 0, \dot{\theta}_0 = 0, \dot{\psi}_0 = 0\right].$$

From this, the linearization of the motion equation is obtained as follows:

$$\left(J_{eq,p} + m_{heli} l_{cm}^2\right) \ddot{\theta} = K_{pp} V_{m,p} + K_{py} V_{m,y} - B_p \dot{\theta} - m_{heli} g l_{cm}, \tag{3}$$

$$\left(J_{eq,y} + m_{heli} l_{cm}^2\right) \ddot{\psi} = K_{pp} V_{m,y} + K_{yp} V_{m,p} - B_p \dot{\psi} - 2 m_{heli} l_{cm}^2 \theta \dot{\psi} \dot{\theta}. \tag{4}$$

Substituing $x = \left[\theta, \psi, \dot{\theta}, \dot{\psi}\right]'$ in (3) and (4) and solving for $\dot{x}$ the following linear model of state space is obtained:

$$\dot{x} = \begin{bmatrix} 0\,0 & 1 & 0 \\ 0\,0 & 0 & 1 \\ 0\,0 & -\frac{B_p}{J_{eq,p}+m_{heli}l_{cm}^2} & 0 \\ 0\,0 & 0 & -\frac{B_y}{J_{eq,y}+m_{heli}l_{cm}^2} \end{bmatrix} x + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \frac{K_{pp}}{J_{eq,p}+m_{heli}l_{cm}^2} & \frac{K_{py}}{J_{eq,p}+m_{heli}l_{cm}^2} \\ -\frac{K_{yp}}{J_{eq,y}+m_{heli}l_{cm}^2} & -\frac{K_{yy}}{J_{eq,y}+m_{heli}l_{cm}^2} \end{bmatrix} u,$$

$$y = \begin{bmatrix} 1\,0\,0\,0 \\ 0\,1\,0\,0 \\ 0\,0\,1\,0 \\ 0\,0\,0\,1 \end{bmatrix} x.$$

where K_{pp}, K_{yy}, K_{py}, K_{yp} are the torque-constants used to obtain coupled torques acting on the pitch and yaw axes; for the state space model the input u and output y vectors are $u = \left[V_{m,p}, V_{m,y}\right]'$ and $y = [x_1, x_2, x_3, x_4]'$, V_{mp} is the input pitch motor voltage and V_{my} is the input yaw motor voltage. Notice that, since the output matrix is the identity matrix, all states are measurable.

The model makes use of several Simulink and Matlab programs to develop the helicopter basic dynamics, by running a simulation of the closed-loop response, using the position controller. Regarding control issues, two controllers are designed: a FF-LQR and a FF+LQR+I. The FF+LQR regulates the pitch axis of the helicopter, using feed-forward (FF) and proportional-velocity (PV) compensators, while the yaw axis only makes use of a PV control. The FF+LQR+I controller uses an integrator in the feedback loop to reduce the steady-state error, by a feed-forward and proportional-integral-velocity (PIV) algorithms to regulate the pitch, and only a PIV to control the yaw angle. This work focuses on the FF+LQR+I controller, as follows. The FF+LQR control converges $(\theta, \psi, \dot{\theta}, \dot{\psi}) \rightarrow (\theta_d, \psi_d, \dot{\theta}_d, \dot{\psi}_d)$ where θ_d is the desired pitch angle and ψ_d is the desired yaw angle, such that:

$$\begin{bmatrix} u_p \\ u_y \end{bmatrix} = \begin{bmatrix} K_{ff}\frac{m_{heli} g l_{cm} cos\theta_d}{K_{pp}} \\ 0 \end{bmatrix}.$$

The addition of an integrator requires to introduce the states $\dot{x}_5 = \theta$ and $\dot{x}_6 = \psi$, so the linear state-space model is augmented as:

$$\dot{x} = \begin{bmatrix} 0\,0 & 1 & 0 & 0\,0 \\ 0\,0 & 0 & 1 & 0\,0 \\ 0\,0 & -\frac{B_p}{J_{eq,p}+m_{heli}l_{cm}^2} & 0 & 0\,0 \\ 0\,0 & 0 & -\frac{B_y}{J_{eq,y}+m_{heli}l_{cm}^2} & 0\,0 \\ 1\,0 & 0 & 0 & 0\,0 \\ 0\,1 & 0 & 0 & 0\,0 \end{bmatrix} x + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \frac{K_{pp}}{J_{eq,p}+m_{heli}l_{cm}^2} & \frac{K_{py}}{J_{eq,p}+m_{heli}l_{cm}^2} \\ -\frac{K_{yp}}{J_{eq,y}+m_{heli}l_{cm}^2} & -\frac{K_{yy}}{J_{eq,y}+m_{heli}l_{cm}^2} \\ 0 & 0 \\ 0 & 0 \end{bmatrix} u,$$

$$y = \begin{bmatrix} 1\,0\,0\,0\,0\,0 \\ 0\,1\,0\,0\,0\,0 \\ 0\,0\,1\,0\,0\,0 \\ 0\,0\,0\,1\,0\,0 \\ 0\,0\,0\,0\,1\,0 \\ 0\,0\,0\,0\,0\,1 \end{bmatrix} x.$$

Using the adequate Q and R weighting matrices, the control gain is as follows:

$$k = \begin{bmatrix} 18.9 & 1.98 & 7.48 & 1.53 & 7.03 & 0.77 \\ -2.22 & 19.4 & -0.45 & 11.9 & -0.77 & 7.03 \end{bmatrix}.$$

Thus, the FF+LQR+I controller is:

$$\begin{bmatrix} u_p \\ u_y \end{bmatrix} = \begin{bmatrix} K_{ff}\frac{m_{heli} g l_{cm} cos\theta_d}{K_{pp}} \\ 0 \end{bmatrix} - \begin{bmatrix} k_{11} & k_{12} & k_{13} & k_{14} \\ k_{21} & k_{22} & k_{23} & k_{24} \end{bmatrix}$$

$$\begin{bmatrix} \theta - \theta_d \\ \psi - \psi_d \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} - \begin{bmatrix} k_{15}(\theta - \theta_d) + \int k_{16}(\psi - \psi_d) \\ k_{25}(\theta - \theta_d) + \int k_{26}(\psi - \psi_d) \end{bmatrix}.$$

5. Experimental approach

In order to study the impact of network utilization on closed control loop, the 2-DOF Helicopter control model is built as a NCS. Several nodes are connected through a common communication network. The experiment focuses on network scheduling, and the main objective is to balance the amount of data sent through the network, in order to avoid latency and under sampling.

The NCS for the experiment consists of 8 processors. These real-time kernel processors and the network are simulated using TrueTime [2, 12] based on Matlab/Simulink. The network used is a CSMA/AMP(CAN) with a transmission rate of 80000 bits/second, and not data loss. The NCS model is shown in Fig. 4.

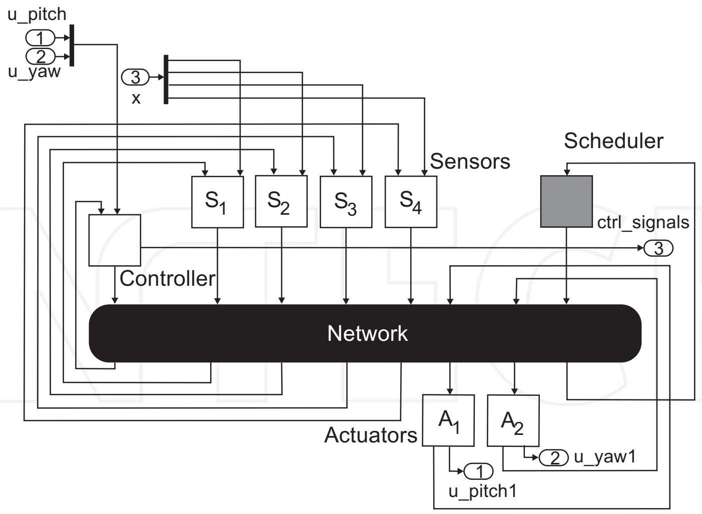

Figure 4. Networked Control System to analyze frequency transmission over the network

Four sensor nodes execute periodic tasks to sense control signals, as well as other additional periodic tasks. Each task has a period p_i and time consumption c_i (Fig. 4). The sensed control signals are $x = (\theta, \psi, \dot{\theta}, \dot{\psi})$. This model has a controller node, depicted on the left side (Fig. 4). This controller takes the control law from the FF+LQR module by means of a task, which activates by event. The time consumption of the controller task is the maximum average time it takes to compute the control law. The controller node uses the values from sensors, and sends control outputs u_p and u_y, that correspond to the pitch and yaw voltages. Two actuator nodes, located on the bottom right corner (Fig. 4), receive signals from the controller node. Finally, the scheduler node, located on the top right corner (Fig. 4), organizes the activity of the other seven nodes, and it is responsible for periodic allocation bandwidth, used by these nodes. Each node initializes, specifying the number of inputs and outputs of the respective TrueTime kernel block, defining a scheduling policy, and creating periodic tasks for the simulation. These tasks involve parameters about the periodic times and the consumption times. The task periodic times define the time interval between tasks, whereas the consumption times refer to the execution time of the task. Fig. 5 shows the 2-DOF Helicopter model, with a RTDS, where feedback control loop is closed through a communication network.

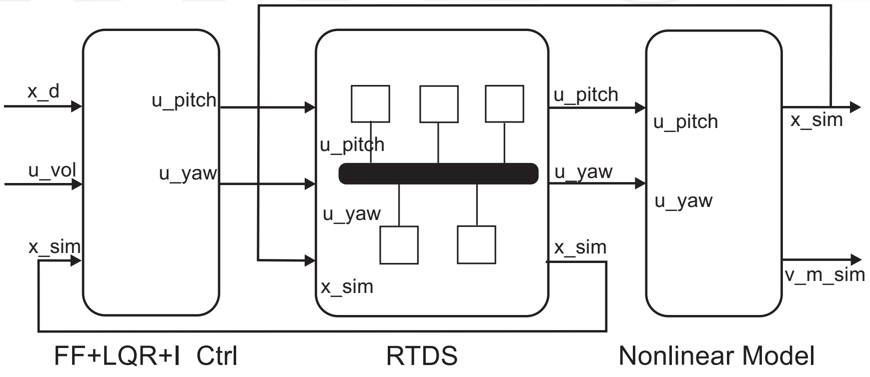

Figure 5. Networked control integrated into closed control loop of the 2DOF Helicopter

Changes on the real-time task parameters of the RTDS commonly impact on network utilization, and therefore, on the control performance [8, 9]. The problem to tackle, thus, is to find a proper way to schedule the common communication network of the RTDS, based on managing an accurate sampling period, capable of keeping both, the network load and required integrated performance.

A criteria to quantify the system's quality performance is the integral of the absolute value of the error (commonly expressed as IAE) is used:

$$IAE = \int_{t_0}^{t_f} |e(t)|\,dt \approx \sum_{k=k_0}^{k_f} |r(kh) - y(kh)| \tag{5}$$

where $r(t)$ is reference signal or setpoint, $y(t)$ is system output signal, $t_0(k_0)$ and $t_f(k_f)$ are the initial and final continuous(discrete) times of evaluation period [15].

5.1. Numerical simulations

In this section numerical simulations using the Network scheduling strategy based on Frequency Transmission are exposed. The network scheduling strategy dynamically adjusts the frequencies, considering the participation of several nodes of the NCS. These scheduling approach shows a way to manage the network resources, especially with a limited network bandwidth. These techniques avoid network delays during transmission. Numerical simulations shows that the dynamical changes of this strategy improve the RTDS response under fault scenarios.

The relationship amongst IAE and sampling periods of the primary sensing tasks is shown in Fig. (6). When the network transmits data without overloading or under sampling the output signals of pitch and yaw angles are similar to induced reference signal, so that the value of IAE is small. In contrast, when the transmission rate of sensor task set exceeds the upper bound data transmission rate the system becomes unstable, IAE increases accordingly.

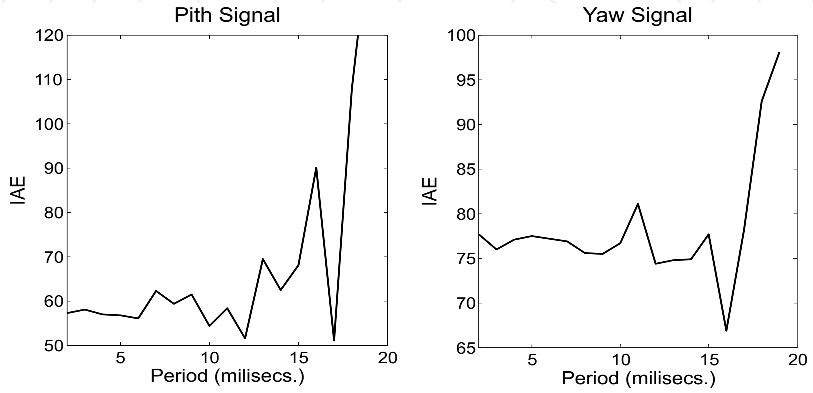

Figure 6. Values of the IAE using different sampling periods

The sensing periods was fixed in 10 miliseconds, sampling during the time simulation with this rate. In a fault-free scenario the pitch and yaw singnals track the reference signal, this behaviour is presented in Fig. 7

It is assumed that in a fault scenario the change in periods of tasks may be produced, this induces a network utilization that possibily can not be supported by the bandwidth or perhaps stops data tranmission continuity. The effect of a fault over the RTDS is shown in the Fig. 8. In this case the periods of sensor tasks produce a low sampling resulting in loss of control.

For the particular case study was made off-line analysis of the frequencies with which can be transmitted without increasing the value of the IAE for the sensing tasks. These ranges serve as parameters for the calculation of new transmission frequencies using the proposed model.

When fault occurs a signal is transmitted to the scheduler, this signal contains the *id* of sensor with fail and the parameters of the tasks on execution before the failure. The scheduler uses the frequency range and time of execution of the sensor tasks that has just entered, scheduler knows frequencies ranges of the tasks that continue without fail. This information feeds frequency transition model to compute a new frequency transmission range through a LQR

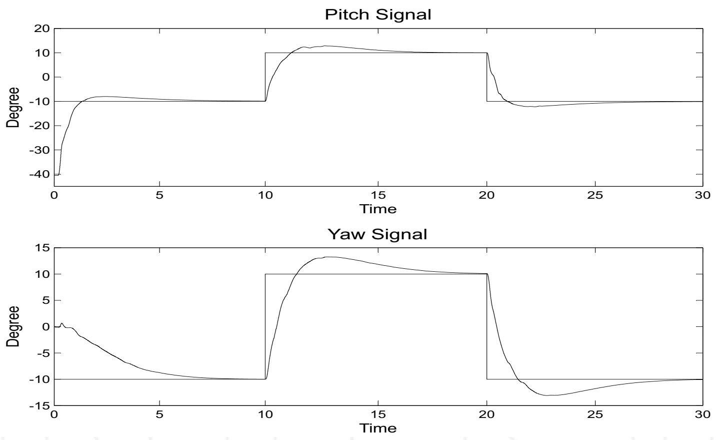

Figure 7. Pitch and yaw signals in a fault-free scenario

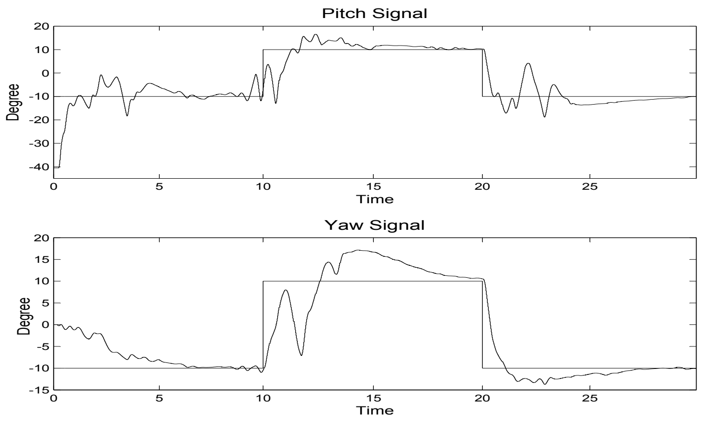

Figure 8. Pitch and yaw signals in a fault scenario

control scheme, new frequencies are assigned to the sensor nodes in which the task period is changed therefore the effect of fault scenario is minimized balancing the amount of data sent over the network.

Numerical simulations were performed using the values of the maximum, minimum, and real frequencies besides the computational time, the values used for the sensor nodes are shown in the table 1.

Figure 9 shows the frequencies controlled, this control bounds the frequency into a schedulability region where the IAE is low.

Figure 10 shows system behavior during 30 seconds using frequency transition model. Nominal frequencies start in schedulability region during 8 secs, system transmits with this

Node	Max. Freq.	Min. Freq.	Real Freq.	Consume
1	70	280	40	0.002
2	50	260	250	0.002
3	50	250	100	0.002
4	55	300	50	0.002

Table 1. Maximum, minimum, and real frequencies besides the computational time

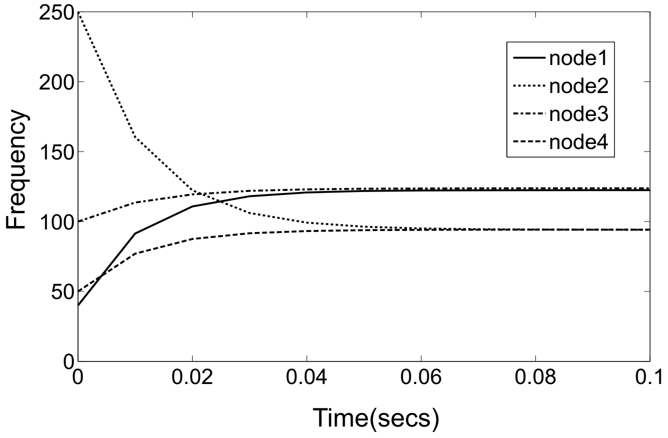

Figure 9. Frequencies bounded into a schedulability region usign a frequency transmission controller

rate. In second 9 fault appears with a change of context, system transmits with this frequency until second 18. In second 19 frequency transition model change frequencies of backup tasks.

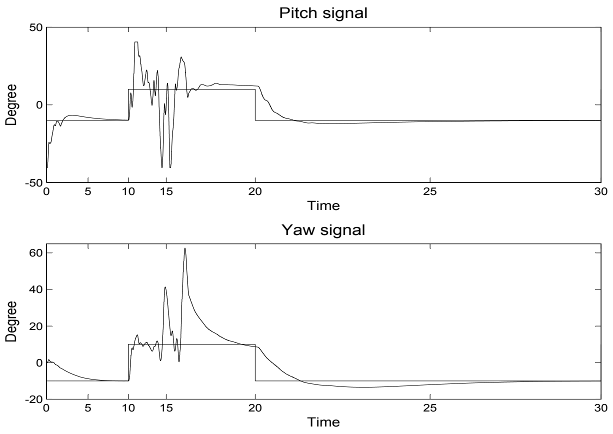

Figure 10. Pitch and yaw signals in a fault scenario using frequency transmission model

Figure 11 shows the computer network as well

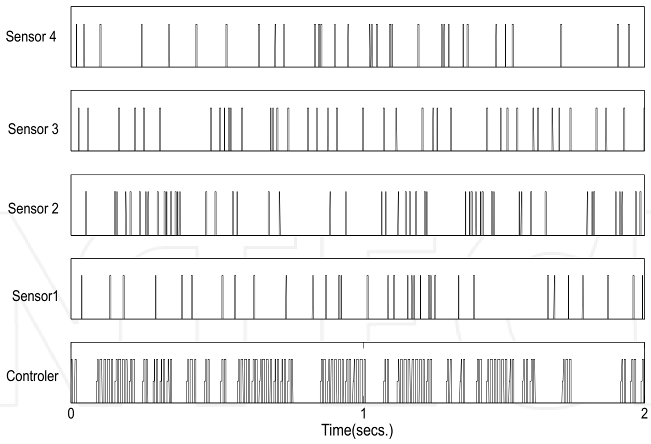

Figure 11. Sensing activiity and the related use of the computer networking using the Truetime toolbox

6. Conclusions

This work show a study of network scheduling strategy using numerical simulations. A simulation of a particular RTDS, a 2-DOF helicopter, is built using TrueTime as real-time simulation tool. A network scheduling strategy based on changes of frequency transmission rates is implemented in order to expose the advantages of using dynamic scheduling in an ad-hoc implementation for the network of a NCS. The use of numerical simulations aid to explore several considerations in design and analysis of a NCS.

Author details

Oscar Esquivel-Flores, Héctor Benítez-Pérez and Jorge Ortega-Arjona
Universidad Nacional Autónoma de México, México

7. References

[1] Cervin, A., Henriksson, D., Lincoln, B., Eker, J. & Årzen, K.-E. [2003]. How does control timing affect performance? analysis and simulation of timing using jitterbug and truetime, *Control Systems, IEEE* 23(3): 16 – 30.

[2] Cervin, A., Ohlin, M. & Henriksson, D. [2007]. Simulation of networked control systems using truetime, *Proc. 3rd International Workshop on Networked Control Systems: Tolerant to Faults*, Nancy, France.

[3] Chen, H., Luo, W., Wang, W. & Xiang, J. [2011]. A novel real-time fault-tolerant scheduling algorithm based on distributed control systems, *Computer Science and Service System (CSSS), 2011 International Conference on*, pp. 80 –83.

[4] Esquivel-Flores, O., Benítez-Pérez, H., Méndez Monroy, E. & Menéndez, A. [2010]. Efficient overloading techniques for primary-backup scheduling in real-time systems, *ICI Express Letters Part B: Applications* 1(1): 93–98.
[5] Gupta, R. & Chow, M.-Y. [2010]. Networked control system: Overview and research trends, *Industrial Electronics, IEEE Transactions on* 57(7): 2527 –2535.
[6] Henriksson, D., Cervin, A., Andersson, M. & Årzén, K.-E. [2006]. Truetime: Simulation of networked computer control systems, *Proceedings of the 2nd IFAC Conference on Analysis and Design of Hybrid Systems*, Alghero, Italy.
[7] Henriksson, D., Redell, O., El-Khoury, J., Cervin, A., Törngren, M. & Årzén, K.-E. [2006]. Tools for real-time control systems co-design, *in* H. Hansson (ed.), *ARTES – A network for Real-Time research and graduate Education in Sweden 1997–2006*, Department of Information Technology, Uppsala University, Sweden.
[8] Lian, F.-L., Moyne, J. & Tilbury, D. [2001]. Time delay modeling and sample time selection for networked control systems. *International Mechanical Engineering Congress and Exposition, Proceedings of ASME-DSC*, Vol. XX.
[9] Lian, F.-L., Moyne, J. & Tilbury, D. [2002]. Network design consideration for distributed control systems, *Control Systems Technology, IEEE Transactions on* 10(2): 297 –307.
[10] Lian, F.-L., Yook, J., Otanez, P., Tilbury, D. & Moyne, J. [2003]. Design of sampling and transmission rates for achieving control and communication performance in networked agent systems, *American Control Conference 2003, Proceedings of* , Vol. 4, pp. 3329 – 3334.
[11] Liu, C. L. & Layland, J. W. [1973]. Scheduling algorithms for multiprogramming in a hard-real-time environment, *J. ACM* 20(1): 46–61.
[12] Ohlin, M., Henriksson, D. & Cervin, A. [2007]. *TRUETIME 1.5 Reference Manual*.
[13] Quanser [2006]. *Quanser 2 DOF Helicopter. User and Control Manual*, Quanser. Innovate-Educate.
[14] Tipsuwan, Y. & Chow, M.-Y. [2003]. Control methodologies in networked control systems, *Control Engineering Practice* 11(10): 1099 – 1111.
[15] Xia, F. & Sun, Y. [2008]. *Control and Scheduling Codesign. Flexible Resource Management in Real-Time Control Systems*, Springer.