Hardware in the loop networked control and diagnosis of a quadrotor drone [1]

Cédric Berbra * **Daniel Simon** ** **Sylviane Gentil** *
Suzanne Lesecq *

* *Gipsa-lab, Control Systems Department, INPG-UJF-CNRS, BP 46, St Martin d'Hères, 38402, France(firstname.name@gipsa-lab.inpg.fr).*
** *INRIA Rhône-Alpes, Inovallée, 655 avenue de l'Europe,Montbonnot,38 334 Saint Ismier Cedex France(Daniel.Simon@inrialpes.fr)*

Abstract: This paper deals with the design, real-time implementation and testing of the control and diagnostic functions of a quadrotor drone, implemented using a Network Control System (NCS). For diagnostic purpose, an indicator is used to make the difference between packet losses due to the network and sensor faults. The control and diagnostic algorithms are implemented as a multitask and multirate real-time software using a design environment called Orccad. A hardware-in-the-loop simulator has been set up, connecting with a CAN bus the controller, running on the embedded target, and a real-time simulated model of the drone. The results of the real time implementation are compared with the simulation results using Matlab/Simulink and Truetime toolbox. This allows the validation of the proposed architecture.

Keywords: Quadrotor, Real time, Networked Control Systems, Co-design, Hardware in the loop, Network.

1. INTRODUCTION

Embedded systems are taking growing importance in high technology industry such as automotive industry or aeronautics, and span many research problems for control scientists. The fact that most of these systems are network controlled is probably the newest one and plants interesting theoretical control and diagnosis problems. Analyzing, prototyping, simulating and guarantying safety of these systems are also very challenging. We need models for the mechatronic continuous system, for the discrete controllers and diagnosers and for the network behavior. Implementation constraints such as tasks response time, multi-rate sampling, synchronization and various sources of delay make the run-time behavior difficult to predict. The network Quality of Service (delays, data loss, jitters) influences the controlled system properties (Quality of Control).

In this paper, a quadrotor is taken as an example of an embedded system whose safety is critical. This quadrotor is controlled and diagnosed through a network. The interactions between the controller, the diagnoser and the system are first studied with on one hand Matlab/Simulink, which are standard tools for simulating the physical system, the controller, the diagnoser, and on the other hand Truetime for the network. Then, an approach named hardware-in-the-loop is proposed. In this case, the physical system is still numerically simulated, but the control and diagnostic algorithms are running on the embedded target as real-time tasks, and they communicate with the numerical simulator through a real network. This is possible with a tool named Orccad and allows the validation of the proposed real time architecture.

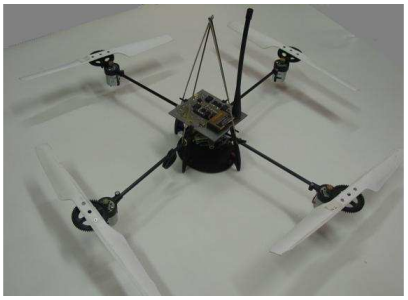

Fig. 1. The quadrotor benchtest.

Section 2 presents the quadrotor and its Inertial Measurement Unit (IMU) models. Section 3 summarizes the control and the diagnosis implemented to isolate IMU sensor faults. Section 4 deals with the real time architecture implementation with Orccad. Section 5 compares experimental results obtained with Matlab/Truetime with those obtained with hardware in-the-loop. Section 6 gives a short conclusion and future research directions.

2. THE QUADROTOR AND IMU MODELS

2.1 Description of the quadrotor structure

Figure 1 depicts the experimental quadrotor designed for this project. A quadrotor (drone with 4 rotors, each one

[1] This work is partially supported by the Agence Nationale de la Recherche (France) grant ANR-05-SSIA-0015-03

driving a blade) is regarded as a composition of two PVTOL (Planar Vertical Take-Off and Landing) whose axes are orthogonal, allowing a movement of six degrees of freedom. Two frames are considered (see Figure 2): the *inertial frame* $\mathbf{R}(e_x, e_y, e_z)$ and the *body frame* $\mathbf{B}(e_1, e_2, e_3)$ attached to the structure with its origin at the center of mass of the quadrotor. In the following paragraphs, the model is focused on the quadrotor orientation (or attitude). The orientation can be represented by three angles yaw-pitch-roll (ϕ, θ, ψ) or by a unitary quaternion that is used in this paper: $q = [q_0 \ \overrightarrow{q}^T]^T$, $\|q\|_2 = 1$ (Chou [1992]). The rotation matrix $C(q)$ can be expressed with $q \in \Re^4$

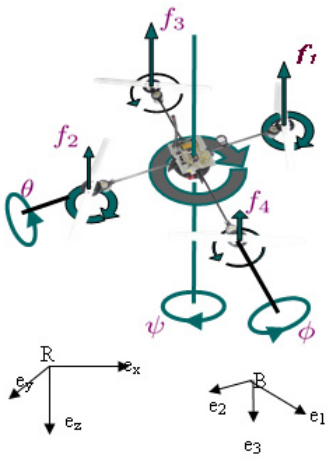

Fig. 2. Definition of the coordinate frames.

$$C(q) = (q_0^2 - \overrightarrow{q}^T\overrightarrow{q})I + 2(\overrightarrow{q}\overrightarrow{q}^T - q_0[q^\times]) \quad (1)$$

The advantage of using q for the attitude representation is to avoid singularities that appear with classical angular representations (Euler angles or Cardan angles). Moreover, q is an elegant and efficient attitude representation from a computational point of view, which is of great importance for embedded systems. Note that a coordinate change from $\overrightarrow{r}$ in the reference frame to $\overrightarrow{c}$ in the body frame is expressed with

$$c = q^{-1} \otimes r \otimes q = \overline{q} \otimes r \otimes q \quad (2)$$

where $r = [0 \ \overrightarrow{r}^T]^T$, $c = [0 \ \overrightarrow{c}^T]^T$. $\overline{q} = [q_0 \ -\overrightarrow{q}^T]$ is the conjugate of q and $\otimes$ is the quaternion multiplication.

With $\omega \in \Re^3$ the angular velocity of the quadrotor measured by the rate gyros in the $\mathbf{B}$ frame, the rotational quaternion dynamic equation is

$$\dot{q} = \frac{1}{2}\Omega(\omega)q = \frac{1}{2}\Xi(q)\omega \quad (3)$$

with $\Omega(\omega)$ given by

$$\Omega(\omega) = \begin{pmatrix} 0 & -\omega^T \\ \omega & -[\omega^\times] \end{pmatrix} \quad (4)$$

$[\omega^\times]$ is the self cross product

$$[\omega^\times] = \begin{pmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{pmatrix} \quad (5)$$

The rotational motion of the quadrotor is given by

$$I_f\dot{\omega} = -[\omega^\times]I_f\omega - G_a + \tau_a \quad (6)$$

where $I_f \in \Re^{3\times3}$ is the symmetric positive definite constant inertia matrix of the quadrotor with respect to frame $\mathbf{B}$. The gyroscopic torques G_a are modeled as

$$G_a = \sum_{i=1}^{4} I_r(\omega \times e_z)(-1)^{i+1}\omega_{Mi}, \quad (7)$$

The components of $\tau_a \in \Re^3$ (i.e. the torque generated by the four rotors) are given by

$$\begin{aligned} \tau_{roll} &= d \cdot b \cdot (\omega_{M2}^2 - \omega_{M4}^2), \\ \tau_{pitch} &= d \cdot b \cdot (\omega_{M1}^2 - \omega_{M3}^2), \\ \tau_{yaw} &= k \cdot (\omega_{M1}^2 + \omega_{M3}^2 - \omega_{M2}^2 - \omega_{M4}^2) \end{aligned} \quad (8)$$

where d is the distance from the rotors to the center of mass of the quadrotor; b and k are two parameters depending on the air density, the radius, the shape, the pitch angle of the blade and other factors; ω_{Mj}, $(j = 1..4)$ are the four motors speed.

2.2 Inertial Measurement Unit (IMU)

The estimation of the attitude (or orientation) and of the rotational speed of the quadrotor is a pre-requisite for its attitude control. An Inertial Measurement Unit is embedded in the quadrotor in order to provide measurements that will be fused to estimate the attitude. The IMU consists of three rate gyros (g_1, g_2, g_3), a tri-axis accelerometer (a_1, a_2, a_3), and three magnetometers (m_1, m_2, m_3).

Rate Gyro modeling. The angular velocity ω is measured in body frame $\mathbf{B}$ with three rate gyros mounted at right angles. The measurements delivered by these sensors are usually affected by noise. Theoretically, the integral of ω could give the relative orientation but the presence of noise generates errors that are accumulated over time. The sensor measurements are modeled as

$$\omega_g = \omega + \eta_1 \quad (9)$$

where ω_g are the sensor values and η_1 is assumed to be Gaussian zero-mean white noise.

Accelerometers. The 3-axis accelerometer senses the inertial forces and gravity in body frame $\mathbf{B}$. The transformation of accelerometer measurements from inertial frame $\mathbf{R}$ to body frame $\mathbf{B}$ is computed as follows

$$b_{acc} = C(q)(\dot{v} - g) + \eta_{acc} \quad (10)$$

where b_{acc} corresponds to the measurements in $\mathbf{B}$, and η_{acc} is Gaussian zero-mean white noise. The motion is supposed quasi-static so that linear accelerations $\dot{v}$ are neglected (i.e. $\dot{v} \approx 0$). Note that this assumption is fully valid because the quadrotor is controlled so as to obtain hover conditions $(\varphi \approx \theta \approx \psi \approx 0)$. Moreover, the Coriolis effect is not taken into account. In this way, accelerometers are only sensitive to the gravitational field g.

Magnetometers. The information provided by the three magnetometers mounted orthogonally is added to the inertial measurements. The magnetic field is sensed in body frame $\mathbf{B}$. It is defined by

$$b_{mag} = C(q)h_m + \eta_{mag} \quad (11)$$

where $h_m = [h_{mx}\ 0\ h_{mz}]^T$ and b_{mag} are the three components of the magnetic field in $\mathbf{R}$ and $\mathbf{B}$, respectively.

Remark. Note that the accelerometer and magnetometer measurements are modeled by **static non linear equations** that depend on constant known vectors g and h_m and on matrix $C(q)$ which is a non-linear function of q.

3. CONTROL AND DIAGNOSIS

3.1 Attitude control

The rotational speed and quaternion estimation (ω and $\hat{q}$) are used in a feedback loop. The attitude reference is given by a quaternion reference q^{ref}. The controller implemented for the attitude stabilization is detailed in (Guerrero-Castellanos et al. [2007]) and supplies for each motor the speed reference that is managed by a local PI control loop.

3.2 Attitude observer

The measurements provided by the IMU feed a non linear observer whose output $\widehat{q}$ is used by the controller. The observer proposed in (Guerrero-Castellanos et al. [2007]) is depicted in Figure 3.

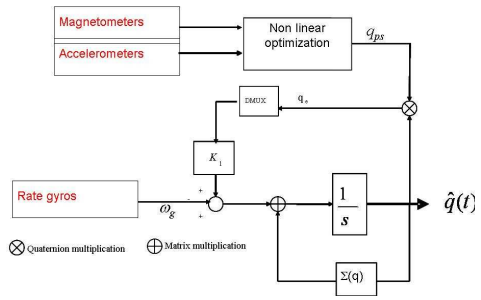

Fig. 3. Non-linear observer scheme.

The non-linear observer principle is as follows. A "pseudo-measurement" quaternion q_{ps} is computed from b_{mag} and b_{acc}, based on the nonlinear static measurement equations

$$q_{ps} = \arg\min [\frac{1}{2} \parallel [b_{acc}^T\ b_{mag}^T]^T - h(q) \parallel_2^2] \qquad (12)$$

where $h(q)$ is derived from (10) and (11). A Sequential Quadratic Programming (SQP) algorithm is used at this step (e.g. "fmincon" in the Matlab environment). $\hat{q}$ is then obtained by propagating the kinematic equation (3) using ω_g in (9), and the discrepancy between $\hat{q}$ and q_{ps} is computed as follows

$$q_e = \hat{q} \otimes q_{ps}^{-1} = [q_{e_0}\ \overrightarrow{q_e}^T]^T \qquad (13)$$

3.3 Generalized observers scheme

A well-known diagnostic method consists in designing state observers that do not use all the system outputs but only a subset of them. Therefore, the estimated state and outputs are independent of the discarded measurements and of the corresponding sensors possible fault (Isermann [2006]). A bank of generalized observers has thus been designed for the quadrotor diagnosis, based on the observer in section 3.2. Each generalized observer uses eight out of the nine IMU sensors. Its estimated quaternion is compared to the quaternion computed with the theoretical model fed by the four motor speed references. The resulting difference is thus insensitive to the faults in the sensor that is not used for the attitude estimation. The signature table obtained in this way is strongly isolable (Berbra et al. [2008]). To choose more easily the thresholds to decide if the quaternion difference is not zero, this one is further transformed into the corresponding differences in the three attitude angles. These angle discrepancies constitute the residual for the diagnosis, each residual r_i being a 3-dimensional vector. Practically, each angle threshold is fixed to 2°, value deduced from the noise level.

Two different techniques for this generalized observers design have been implemented, depending on the subset of sensors considered. The first technique computes $\widehat{q}$ using two rate gyros and all the magneto and accelero measurements. The second technique is only based on the static non linear measurement equations (10) and (11). The principle of the observer bank is depicted in Figure 4 and each technique is shortly presented hereafter.

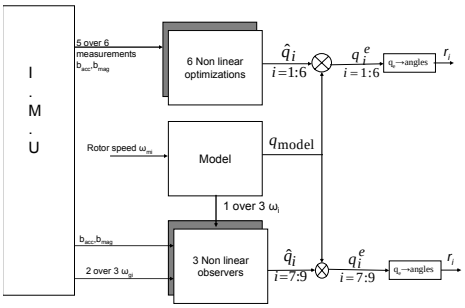

Fig. 4. Multi-observer scheme.

Attitude estimation with acceleros, magnetos and without rate gyros measurements. To estimate the residuals $\hat{q}_i$ ($i = 1, \cdots, 6$) in the top of Figure 4, only five over six accelero and magneto measurements are considered, satisfying (10) and (11). The quaternion estimation based only on these equations is obtained by solving again a non-linear optimization problem similar to the one in (12), but now with 5 equations instead of 6. Six estimators are designed in this way, and the differences q_i^e ($i = 1, \cdots, 6$) are sensitive to faults in all the accelerometers and magnetometers except the discarded one.

Attitude estimation using acceleros, magnetos and two rate gyros measurements. As can be seen in the bottom of Figure 4, the measurements provided by the IMU feed three nonlinear observers based on the scheme presented in section 3.2. The discarded rate gyro measurement is replaced with the value that is computed with the mechanical model (6). Therefore, the attitude estimated with each observer in this bank is sensitive to faults in all the acceleros, magnetos and in two rate gyros over three. From these observers, three residual vectors, denoted r_i ($i = 7, \cdots, 9$) are obtained.

4. HARDWARE-IN-THE-LOOP ARCHITECTURE

In the first step, the simulation of the quadrotor with its control and its diagnosis has been implemented with Matlab/Simulink.

Then, the simulation of the network has been introduced, using the Truetime toolbox (Anderson et al. [2006]), a toolbox for simulation of distributed real-time control systems. The Truetime library provides specific blocks for the network interface modeling. This library is developed in C++ language. All the developed files are compiled in Matlab by using an external C++ compiler. Truetime provides a few types of networks which can be used for simulation of the NCS (Ethernet, CAN, Round Robin, TDMA, FDMA, Switched Ethernet and WLAN or ZigBee Wireless networks).

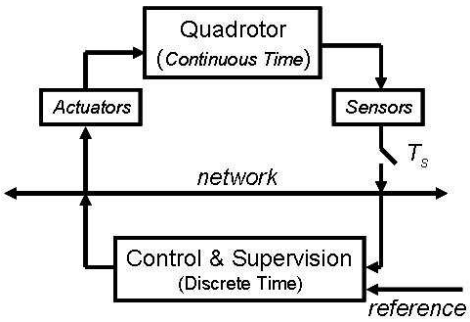

Fig. 5. Closed loop of the Networked Control Quadrotor.

The network is operated in closed loop (see Fig.5)

(1) between the sensors and the controller; the sensors are associated to the numerical information provided by an AD converter that generates the sensor flow (this is called sensor task); the controller task generates the controller flow;
(2) between the controller and the actuators (the four motors) driven through a DA conversion.

The network influence (jitter, delays, packet loss) has been studied in (Berbra et al. [2008]).

However, using Matlab/Simulink with Truetime is only a simulation. It is why, in the third step, a hardware-in-the-loop experimentation has been made (see Fig.6). Now, the mechanical behavior of the quadrotor is still simulated. The quadrotor model is handled by a numerical integrator running on an external PC under Linux. This computer must be fast enough so that the simulated model is faster than real-time and the induced disturbances are negligible w.r.t. computing and networking delays. The control, observation and diagnostic algorithms are implemented in the real embedded hardware, a Phycore MPC5200B-tiny [2] embedded board running Linux on a Freescale MPC603e CPU. The two computers communicate via a CAN network.

In the drone particular case, hardware-in-the-loop experiments provide a safe environment for both algorithms and software validation, prior to experiments with the real (expensive and fragile) quadrotor. In the last step, the Power-PC will be used in the real structure of the quadrotor (Fig.1).

In the next subsection, a description of the real time language used is presented.

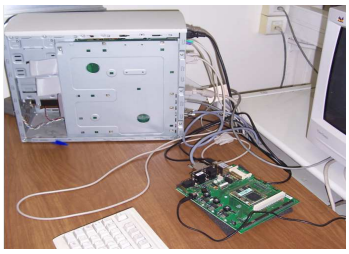

Fig. 6. The real implementation.

4.1 The Orccad *approach*

Orccad is a software environment dedicated to the design, the verification and the implementation of real-time control systems. Besides control law design, it also allows for the specification and validation of complex missions involving the logical and temporal cooperation of various controllers along the life of a control application(Borrelly et al. [1998], Törngren et al. [2006]).

The Orccad methodology is bottom-up, starting from the design of control laws by control engineers, to the design of more complex missions.

The first step in designing a control application is to identify all the necessary elementary tasks involved. Then, for each of the tasks, various issues are considered, either with an automatic control viewpoint (such as defining the regulation problem, control law design, design of reactions to relevant events) or with an implementation aspect (such as the decomposition of the control law into real-time tasks, and selection of timing parameters). Finally, all the real-time tasks are mapped on a target architecture. During this design, the control engineer has a lot of degrees of freedom to meet the end-user requirements and Orccad aims at allowing the designer to exploit these degrees of freedom.

Orccad promotes a controller architecture which is naturally open since it allows access to every level by different users: the application layer is accessed by the end-user, the control layer is programmed by the control expert, and the system layer is accessed by the system engineer. Orccad provides formalized control structures, which are coordinated using the synchronous paradigm, specifically using the Esterel language: while the control laws are periodic (or more generally cyclic) and are programmed using tasks and an RTOS, the discrete-event controller manages these control laws and handles exceptions and mode switching.

The main entities used in the Orccad framework are: module tasks (MT), the real-time tasks which implement functions; robot tasks (RT), the control tasks representing basic control actions encapsulated in a discrete-event controller; robot procedures (RP), a hierarchical composition

[2] `http://www.phytec.com/`

of RTs and other existing RPs, forming more complex structures.

The RT characterizes continuous-time closed-loop control laws, along with their temporal features and the management of associated events. From the application perspective, the RT's set of signals and associated behaviors represent the external view of the RT, hiding all specification and implementation details of the control laws. More complex actions, the RPs, can then be composed from RTs and other RPs in a hierarchical fashion leading to structures of increasing complexity. At the top level, RPs are used to describe and implement a full mission specification. At mid levels they can be used to fulfil a single basic goal through several potential solutions, e.g. a nominal controller supplemented by the recovery substitutions associated with diagnosis and fault detection.

Once a control application has been entirely designed, and for some parts formally verified, a runtime code can be automatically generated for various real-time operating systems, such as Linux in this particular case.

4.2 Quadrotor simulation setup

Figure 7 describes the control and diagnostic setup used for testing purpose. In this block-diagram the blue boxes represent the user-provided modules (i.e. functions) interconnected by their input/output ports (respectively blue/red).

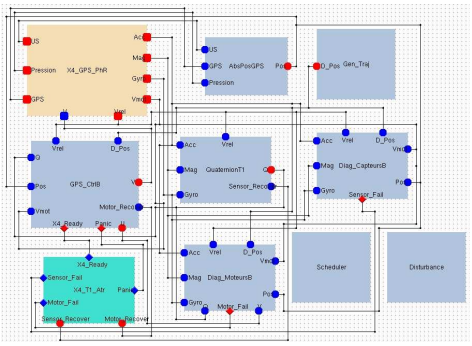

Fig. 7. Control and diagnosis block-diagram.

The main parts of the functions network are now described.

- The attitude control path starts from the quadrotor sensors (accelerometers, gyrometers and magnetometers), module XA-GPS-PhR. The raw measurements are used by the Quaternion module (QuaternionT1) to estimate the drone attitude, which is forwarded to the GPS_CtrlB module to perform the attitude control. The computed desired motor velocities are then sent to the quadrotor via the V port.
- Provision is given for future enhancements of the sensor set, as a GPS-like position sensor and ultrasonic sensors are expected to be integrated in the future. Therefore a trajectory generator GEN_Traj and position estimator AbsPosGPS are integrated in the control architecture to evaluate position control.
- The Diag_CapteursB module runs the diagnostic algorithms that isolates sensor failures. A failure is signaled by the Sensor_Fail weak exception to the X4_T1_Atr module and it is forwarded to the Quaternion module, so that the quaternion estimation algorithm can be adapted according to the reported failure;
- Similarly the Diag_MoteursB module forwards motor failures to be handled by the X4_T1_Atr module;
- The Scheduler module implements a feedback scheduler: it monitors the controller's real-time activity and may react by setting on-the-fly the tasks scheduling parameters, e.g. their firing intervals. For example it has been used to implement a (m,k)-firm dropping policy (Jia et al. [20O7]), and to dynamically adapt the priorities of messages on the CAN bus (Juanole et al. [2008]).
- A Disturbance task allows to generate an extra load either on the CPU or on the CAN bus.

From the real-time point of view, each module is implemented by a real-time task possessing its own programmable timer. Therefore all the modules can be run asynchronously at their own (possibly varying) sampling frequency. The task priorities are set according to their relative importance. Data integrity between asynchronous modules is provided by asynchronous lock free buffers (Simpson [1997]).

The box labeled X4_GPS_PhR represents the quadrotor: sending or reading data on its ports actually calls the drivers, i.e. the functions used to interface the real-time controller with the hardware or with the simulator. The two computers communicate via a CAN bus: the driver ports located on the X4_GPS_PhR interface send and receive data using the Socket-CAN protocol [3].

5. EXPERIMENTS AND RESULTS

In this section, hardware-in-the-loop results are presented and compared with those obtained with Matlab/Simulink and Truetime. In order to illustrate the behavior of the quadrotor in a more intuitive representation, the attitude q is transformed into Euler angles, roll (ϕ), pitch (θ) and yaw (ψ). As explained previously,, the observer and the control law are implemented using the quaternion representation of the attitude. The CAN network (Controller Area Network) is configured with the following characteristics:

- A bit rate of 1 Mb/s in the physical layer.
- A sampling period equal to $h = 10ms$ for the sensor task and the controller task (the system application).
- The sensor and actuator flows are L=64 bits long.

Three scenarios are now presented.

5.1 Basic attitude control

In this scenario, the fault-free case is considered and the quadrotor stabilization is shown. In this case, the quadrotor starts with an initial attitude equal to $[120°; -10°; 50°]$ and the reference attitude is equal to $[0°; 0°; 0°]$.

The hardware-in-the-loop result is shown in Fig.8, left. The red curve represents the roll angle, the green one the pitch angle and the blue one the yaw angle. The time response

[3] `http://developer.berlios.de/projects/socketcan/`

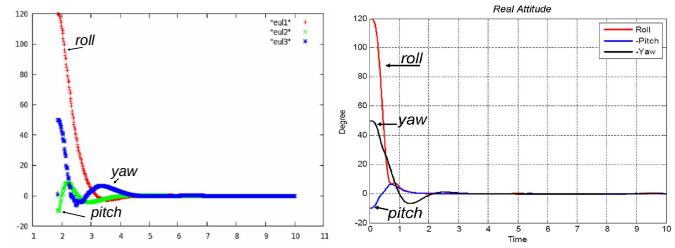

Fig. 8. Attitude of the quadrotor: Orccad (left); Matlab (right)

of the system is almost 2 seconds and is the same as the one obtained with Matlab/Simulink and Truetime (Fig.8, right).

5.2 Packet loss

In this scenario, the same initial and reference positions are used. The objective is now to study the influence of packet losses on the system behavior. A loss of 10% of data from accx is considered. In (Berbra et al. [2008]), a fault indicator has been proposed to make the difference between a sensor fault and a packet loss. This indicator is called $r_{network}$ and it is equal to 1 when the data is not received on time by the control and diagnostic modules (Fig.9, right). Moreover, when the data is lost at $t = (kT_e)$, the quaternion $\hat{q}(KT_e)$ is not computed and the control algorithm holds the value ω_{Mi}^{ref} computed at time $t = (k-1)T_e$.

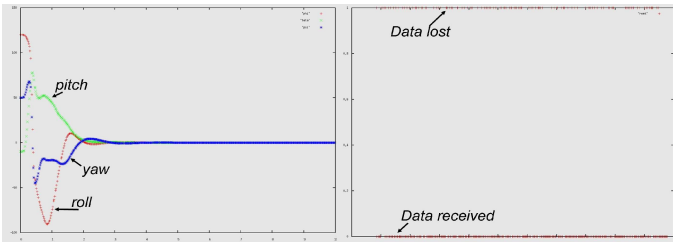

Fig. 9. Attitude of the quadrotor with 10 % of packet loss on accx, left; Indicator of packet loss, right.

The result is shown in Fig.9, left. Small differences can be noted with respect to Fig.8 but it can be seen that the control law is robust to 10 % of packet losses of this sensor. Several other simulations have been made with other packet loss scenarios, and results are quite similar.

5.3 Sensor failure

In this scenario (Fig.10), a bias failure in the rate gyro ω_{gx} is considered. The fault is simulated at time $t = 5$ seconds. Before the fault appearance, all the quaternion errors are close to zero. After $t = 5$ seconds, quaternion estimation $\widehat{q}_8$, $\widehat{q}_9$ are sensitive to the fault and quaternion estimation $\widehat{q}_7$, computed with the observer that discards this sensor value is still correct.

6. CONCLUSION AND PERSPECTIVES

In this paper, a hardware-in-the-loop experiment of a Networked Control System has been made. With Orccad,

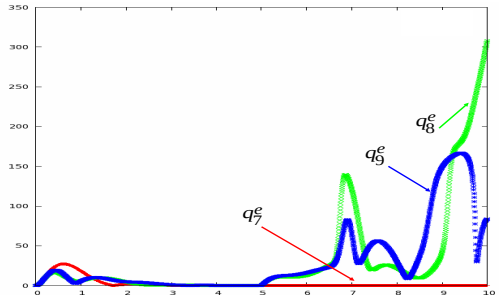

Fig. 10. Quaternion errors $q_i^e, i = 7..9$ with failure in the rate gyro on axis x at $t = 5$ seconds.

the control and diagnostic algorithms are implemented as a multitask and multirate real-time software. The communication between the control/diagnostic tasks and the quadrotor simulator is via a CAN bus. The results of the real time implementation were compared with a pure simulation based on Matlab/Simulink and Truetime. They confirm the robustness of the control with respect to data loss and the potentiality of the proposed diagnostic method. In perspective, the real application will be tested.

REFERENCES

M. Anderson, D. Henriksson, and A. Cervin. *Truetime 1.5 - reference manual*. Department of Automatic Control, Lund Institute of Technology, Swenden, 2006.

C. Berbra, S. Gentil, S. Lesecq, and JM. Thiriet. Co-design of a safe network control quadrotor. In *17th IFAC World Congress*, 2008.

J.J. Borrelly, E. Coste-Manière, B. Espiau, K. Kapellos, R. Pissard-Gibollet, D. Simon, and N. Turro. The ORCCAD architecture. *Int. Journal of Robotics Research*, 17(4):338–359, april 1998.

J.C.K. Chou. Quaternion kinematics and dynamic differential equations. *IEEE Transactions on Robotics and Automation*, 8:53–64, 1992.

J. F. Guerrero-Castellanos, A. Hably, N. Marchand, and S. Lesecq. Bounded attitude stabilization: Application on four-rotor helicopter. In *Proceedings of the 2007 IEEE International Conference on Robotics and Automation*, Roma, Italy, April 2007.

R. Isermann. *Fault-Diagnosis Systems: An Introduction from Fault Detection to Fault Tolerance*. 2006.

N. Jia, Y. Song, and F. Simonot-Lion. Graceful degradation of the quality of control through data drop policy. In *European Control Conference, ECC*, Kos, Greece, 20O7.

G. Juanole, G. Mouney, and C. Calmettes. On different priority schemes for the message scheduling in networked control systems. In *Proceeding of 16th Mediterranean Conference on Control and Automation*, 2008.

H.R. Simpson. Multireader and multiwriter asynchronous communication mechanisms. *IEE Proceedings-Computer and Digital Techniques*, 144(4):241–244, 1997.

Martin Törngren, Dan Henriksson, Karl-Erik Årzén, Anton Cervin, and Zdenek Hanzalek. Tools supporting the co-design of control systems and their real-time implementation; current status and future directions. In *2006 IEEE International Symposium on Computer-Aided Control Systems Design*, Munich, Germany, October 2006.