Flight control system rapid prototyping for the remotely-controlled elettra-twin-flyer airship

Flight Control System Rapid Prototyping for the Remotely-Controlled Elettra-Twin-Flyer Airship

M. Battipede[*], P. Gili[†], M. Lando[‡] and P.Gunetti[§]
Aeronautical and Space Department, Politecnico di Torino – 10129 Torino, ITALY

Nautilus S.p.A. is a small company investing in the design and development of a low-cost multipurpose multi-mission platform, known as Elettra-Twin-Flyer, which is a very innovative radio-controlled airship, equipped with high precision sensors and telecommunication devices. In the prototype phase, Nautilus policy is oriented towards a massive employment of external collaborators to reduce the development costs. The crucial problem of this kind of management is the harmonious integration of all the teams involved on the project. This paper describes the integration process of the PC-104 on-board computer with the avionic devices, which are electronic systems characterized by complex communication protocols. Attention is focused on the testing, verification, validation and final translation of the embedded control software into the on-board computer, through techniques derived from the automatic code generation, such as Rapid Prototyping and Hardware-In-the-Loop.

Nomenclature

CAS	=	Control Allocation System
ETF	=	Elettra-Twin-Flyers
FCC	=	Flight Control Computer
HIL	=	Hardware-in-the-loop
IMU	=	Inertial Measurement Unit
OS	=	Operative System
PIL	=	Processor-in-the-loop
RP	=	Rapid Prototyping
RPM	=	Rate Per Minute
SIL	=	Software-in-the-loop
WOW	=	Weight-On-Wheel
R	=	Correlation function
T	=	Thrust
act	=	Actual signal
com	=	Commanded signal
δ	=	Propeller orientation angle
δ_{comm}	=	Lateral and directional combined command
δ_{dir}	=	Directional joystick command
δ_{lat}	=	Lateral joystick command
δ_{lon}	=	Longitudinal joystick command
δ_{th}	=	Throttle input
τ	=	Correlation delay

[*] Ph.D. Researcher, Aeronautical and Space Department, manuela.battipede@polito.it, AIAA Member.
[†] Associate Professor, Aeronautical and Space Department, piero.gili@polito.it, AIAA Senior Member.
[‡] Ph.D. Researcher, Aeronautical and Space Department, marco.lando@polito.it.
[§] Graduate Student, Aeronautical and Space Department, paologunny@email.it.

I. Introduction

In the last years, the interest of various market sectors, involving military, civil and private organizations (security, commercial, defense, prevention, etc…) has more and more focused on unmanned aerial vehicles able to perform extremely different tasks, such as monitoring, telecommunications, advertising and reconnaissance. In this context, Nautilus S.p.A. is investing in the design and development of a low-cost multipurpose multimission platform, known as Elettra-Twin-Flyers (ETF), which is a very innovative radio-controlled airship equipped with high precision sensors and telecommunication devices[1,2]. Therefore, this remotely-piloted lighter-than-air platform is particularly suitable for inland, border and maritime surveillance missions and for telecommunication coverage extensions, especially in those areas which are either inaccessible or without conventional airport facilities and where the environment impact is an essential concern. In order to offer an innovative product at competitive prices in the restricted airship market sector, the development costs have to be necessarily cut down or at least limited. For this reason, hence, Nautilus policy is oriented towards a massive employment of external collaborators instead of enrolling internal employees, at least in the actual phase of the project. In this context, Nautilus has a collaboration with the Aeronautical and Space department of the Polytechnic of Turin for the research, development and design aspects and claims the cooperation with several firms on technical aspects concerning the manufacturing of many subsystems of this new concept airship. The crucial problem of this kind of management is the harmonious integration, especially in the prototype phase, of the teams belonging to different companies, which operate on specific systems, such as the structure, the avionics, the command system, the energetic and pneumatic systems and so on. Therefore, it is essential to establish very clear procedures, which can guarantee rapid evaluation feedbacks, the subsystem compatibility and the hardware/software integration among the several teams involved in the Nautilus project. Due to the unconventional command system and architectural solution, the Nautilus airship presents many innovative features in various sectors of classical aircraft design, therefore, almost all the subsystems have been purposely designed, assembled and tested because never employed in any other aeronautical application. Hence, great emphasis has been given to the prototype phase[3] in which the feasibility of this really innovative platform has to be proved. Concrete difficulties might rise in the avionics design, in which two different groups should proceed in parallel through an iterative procedure based on reciprocal feedbacks until the verification and validation of the final system is achieved. From one side there is the aeronautical engineer, who has the task of designing the flight control system using development tools such as Fortran, C/C++, *Matlab/Simulink*. From the other side there are the programmer and the electronic technician that translate the final code into the industrial programming language, i.e. ADA, of the on-board computer. These two groups pursue the same goal having to respect, however, different requirements, which might also be conflicting. If not automated, the entire process of iterative design and test induces high development times, which penalize enormously the airship costs. Nowadays, however, the availability of more and more powerful high-speed low-cost computers allows the implementation of new techniques of *Real-time Rapid Prototyping* (RP) and *Hardware In-the-Loop* (HIL) simulation[4] since the early stage of the development cycle, which is strategic to reduce the amount of time necessary to test the embedded control software and the hardware components. At present, the RP and HIL techniques are mainly adopted by the academic world. We reckon, however, that these tools could have a strong impact in the industrial reality, which is very sensitive in pursuing every means to shorten the new product development cycles.

For the above mentioned reasons, Nautilus invested the early stage of the project in the development of a complete and refined Flight Simulator[5-7], which proved to be essential for supporting the whole design process of this non conventional unmanned airship. In particular, the flight simulator provides also an effective tool for the design and test of the innovative flight control system and its following integration in the platform on-board computer. However, as the airship dynamic system simulation in the *Matlab/Simulink* environment is a non-real-time application and it has to be interfaced with an embedded real-time control system through I/O interfaces, it is necessary to convert the *Simulink* dynamic model into a real-time simulation. This operation is carried out by using an automatic code generator, such as the RT-LAB™ software package[8], which generates C code from *Simulink* block diagrams through *Real-Time Worshop*, compiles the C code in the *QNX* operative system and executes it on two PC processors running in parallel, which represent the *Targets*. The ETF Simulator, hence, is organized in four separate entities: the airship dynamics *Target*, the Flight Control Computer (FCC) *Target*, the ground station and the data-link device (receiver and transmitter).

RP and HIL techniques together with the ETF Simulator are both used for the testing, verification, validation and final translation of the embedded control software into the on-board computer. In addition, the most critical hardware components and interfaces are real-time tested through the embedded control software. This paper is focused on the application of these low-cost procedures, which facilitate the real-time interface between the software and hardware devices, used to accomplish ground and in-flight tests in the prototyping phase[9].

II. Beyond the Simulation

Simulating a model enables the designers to check that the system behavior satisfies the requirements well before the construction of the first prototype. Nowadays, this concept is deeply established among engineers, who are all more or less familiar with the standard simulation techniques. The development of a complete and refined Flight Simulator for the Elettra-Twin-Flyers has been essential to support the whole design process of this innovative unmanned airship. In particular, the ETF Simulator was implemented for the following reasons:

- to assist the airship design process from the early stages, in which it was necessary to evaluate the global dynamic behavior of the vehicle, up to the more advanced phases, in which the single components and subsystems of the airship had to be correctly analyzed, dimensioned and integrated in the final product;
- to support the design and test phases of a completely new flight control system;
- to supply a powerful tool for the pilot during the training phase, when the pilot was expected to acquire the ability to fly this non conventional remotely-piloted airship. In fact, even though the piloting has been thought to emulate as much as possible the helicopter concept, a short training phase is necessary in order to get used to the airship response and to obtain the best performance from its great capabilities;
- to support the design of the ground station, which has been realized as close as possible to the Flight Simulator in order to maintain the same operational environment used by the pilot during the training phase;
- to provide an expository platform to effectively highlight the peculiar characteristics and performance of this new concept unmanned airship during marketing operations.

Efforts and costs associated to the modeling and simulation activities, however, are still very high, even when low-cost hardware components and on-the-shelf visual systems are employed. For this reason, software engineers are extensively working on tools, which allow the exploitation of the simulation codes well beyond the early designing phases. All these tools are based on the automatic code generation technique and the development of software drivers, which enable the integration of standard PC-based hardware components with a very wide range of I/O devices. Beyond the simulation, the potentialities of these tools are incredible and it is almost impossible to list all the developments and test activities for which an automatically generated code can be effectively used. The key is to assume that each component of the system sketched in Figure 1 can manifest itself and connect to other components as software, hardware, or simply remain a model. Some applications are very well known among researchers but just a few are established and commonly employed in the aerospace industry. Some of them are briefly reviewed in the following subsections.

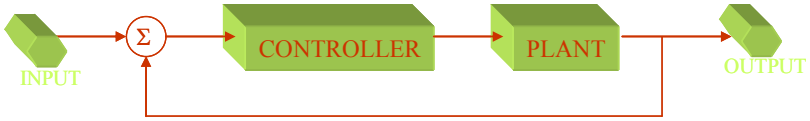

Figure 1. Control system block diagram.

A. Simulation Acceleration

Assuming that the system is reduced in the generic form of Figure 1, code can be generated and compiled for both the plant and controller models. It executes on the host computer and runs much faster than interpretive simulation. This is a popular way to do statistical analysis and parameter studies, such as those involving Monte Carlo methods.

B. Rapid Prototyping (RP)

RP is a very popular technique to develop interactive prototypes, which can be quickly replaced or changed in line with design feedback. This feedback may be derived from colleagues or users as they work with the prototype to accomplish set tasks. Thus, it can be used to adjust and tune the designer requirements. Code is generated just for the controller model. The code is then cross-compiled and downloaded to a high-speed, floating-point, rapid-prototyping computer, where it executes in real time. Prototypes created by this method usually have a high fidelity with the final product and give users and designers a tangible demonstration of the control system performance. The controller parameters can be

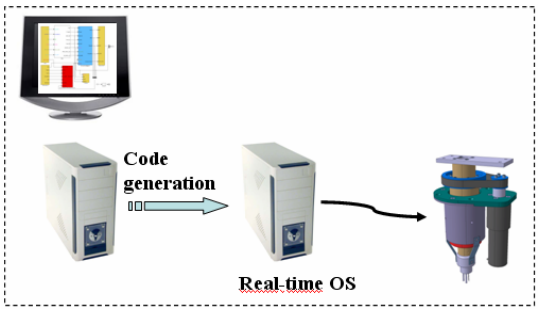

Figure 2. Rapid Prototyping scheme.

tuned "on-line" during tests involving the actual plant. The use of a commercial computer as a target platform allows the designers to test the software well before the hardware of the on board computer is actually available.

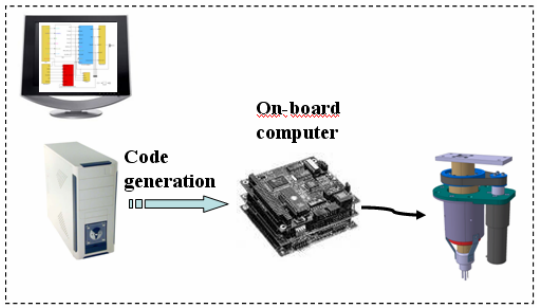

Figure 3. On-target Rapid Prototyping scheme.

C. On–Target Rapid Prototyping

As with RP, code is generated just for the controller model. However, the code is then cross-compiled and downloaded to the embedded microprocessor used in production, or perhaps to a close cousin of it, which is configured with a little more memory and the necessary Input/Output boards. As for the rapid prototyping the controller parameters can be tuned "on-line" during tests involving the actual plant. The on-target rapid prototyping can be seen as the last stage of the prototype development, when the software is almost definite, apart from adjustments or problems that might arise from the interaction with the computer hardware.

D. Production Code Generation

No simulation or development activity is associated with this application. Code is generated for the detailed controller model and downloaded to the actual embedded microprocessor as part of the production software build.

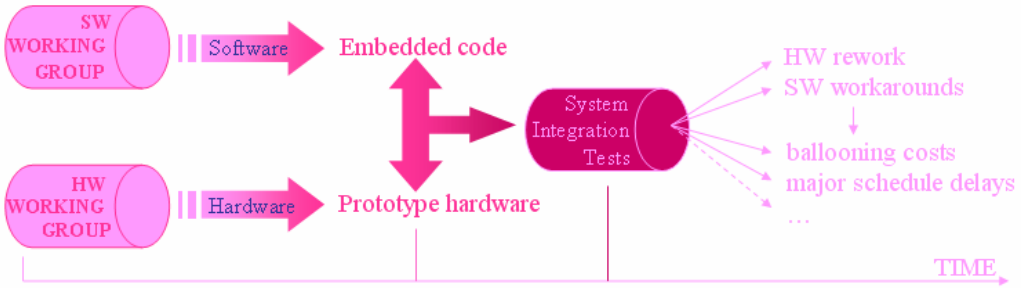

Figure 4. Controller production cycle without the aid of the automatic code generation techniques.

The key here is to ensure that the final build has fully integrated the automatically generated code with existing legacy code, I/O drivers, and real-time operating system. The process is appealing and definitely cost-effective, as it allows to reduce considerably the time associated to the development of the final product. Referring to Figure 4, in fact, it is clear that many factors may concur at a substantial delay in the accomplishment of the final product. Usually, when the controller is complex, the development is carried out by the synergy of two different groups: the software and the hardware working groups. The software manager has the task to produce the embedded code and make his/her staff work under specifications of the controller designer and the constraints of the hardware manager, who has the task to assembly the prototype hardware. The problems arise when the system integration tests begin. They might reveal that major hardware adjustments are required or that some unexpected software requirements necessitate the implementation of extra code lines or simply that minor bugs must be adjusted with workarounds.

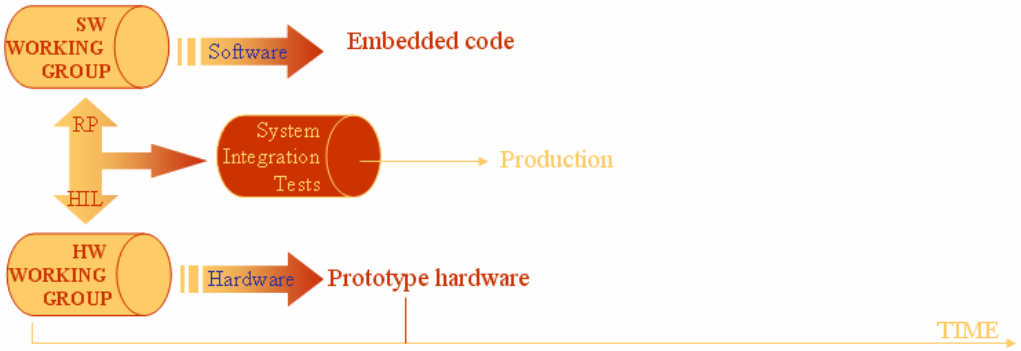

Figure 5. Production cycle with the aid of the automatic code generation techniques.

These problems are not anomalous and both software or hardware engineers are aware that the initially estimated program costs and lengths may rise at the extent that the program is eventually cancelled.

When automatic code generation techniques are employed the system integration tests can be scheduled starting from the very beginning and bringing the software and hardware working groups to communicate from the very early stage of the program, as shown in Figure 5. Software and hardware are developed and corrected step by step, moreover, requirements and constraints are verified in real-time, mitigating the risk of rework or major adjustments.

Among the automatic code generation techniques, the production code generation activity is by far the application that draws more benefits from this concept, for obvious reasons. Unfortunately, however, this tool is not always applicable, unless the final product does not require specific certification. In the avionic field, for instance, the regulation *DO-178B*[10] has become a de facto standard. The FAA's Advisory Circular AC20-115B, in fact, established *DO-178B* as the accepted means of certifying all new aviation software. The *DO-178B* is primarily concerned with development processes and no tool for automatic generation code is allowed to replace the job of at least two separate staff of software engineerings.

E. Hardware-in-the-Loop (HIL) Testing

HIL is another well known activity among engineers, who are aware that expensive, fragile, and unique systems are hard to test, especially if they do not really exist yet, as they are being developed together with the control system. Code is generated just for the plant model and runs on a highly deterministic, real-time computer. Sophisticated signal conditioning and power electronics are needed to properly stimulate the controller inputs (sensors) and outputs (actuator commands). Whereas rapid prototyping is often a development or design activity, hardware-in-the-loop testing serves as more of final lab test phase before road or track or flight tests begin. HIL is particularly meaningful when the plant is a dynamic systems, namely when the plant output is not simply a function of the present inputs, but is instead a function of the present inputs and some combination of past inputs. The problem is how to conduct meaningful tests of such a system, avoiding the risk of damaging the real plant with unexpected consequences. The use of real-time simulation is essential to determine whether the controller digital delays affect the operation of the actual system, well before the plant is irrecoverably damaged.

F. Software-in-the-Loop (SIL) Testing

This type of testing usually does not involve real time simulation. The production code for the controller is executed in an instruction-set simulator, debugger, or within the modeling environment itself, exercising the plant model and interacting with the user. The basic idea of the SIL is the same of the HIL, except that all run on standard PC hardware. The target hardware, together with sensor and actuators, is simulated and the software under test runs on that simulated hardware. The environment simulation runs in software as well. This approach allows the use of cheap PCs for testing the embedded software instead of costly HIL test-beds and in-circuit-emulators, which often also pose availability problems.

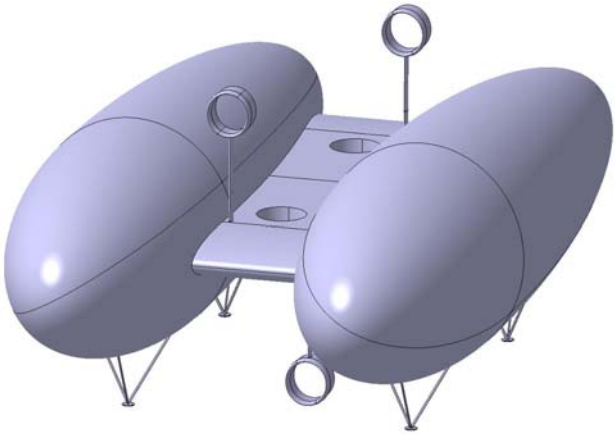

Figure 6. The Nautilus ETF, a new concept unmanned airship.

G. Processor-in-the-Loop (PIL) Testing

This technique is similar to the SIL in that it executes the production code for the controller. However, real I/O via CAN or serial devices is used to pass data between the production code, executing on the processor, and the plant model, executing in the modeling environment. As with SIL testing, PIL testing do not execute in real-time.

III. Plant Description: the Airship Prototype

The Nautilus new concept airship features an architecture and an appropriate command system, which should enable the vehicle to maneuver in forward, backward and sideward flight and hovering with any heading, both in normal and severe wind conditions. To achieve these capabilities the Nautilus airship has been conceived with a highly non conventional architecture based on a double hull with a central plane housing structure, propellers, on

board energetic system and payload. The resulting solution presents many innovative features, which cover many sectors of the aircraft design: almost all the subsystems have been purposely shaped and assembled and have never been tested on aeronautical applications. For this reason, great emphasis has been given to the prototype phase, which has been considered crucial in the final process of the feasibility assessment. The prototype is a reduced-scale platform which has been purposely assembled to test the most critical subsystems, such as the control system and the architectural solution. The ETF command system mainly consists of two vertical rotational axis ducted propellers and four thrust-vectoring propellers mounted on rotating vertical arms (Figure 6). The ducted propellers provide vertical thrust for steep rapid climb and descent and control the pitch attitude of the vehicle. In addition, their action is combined with the helium buoyancy to produce lift in hovering and also in forward flight, where the lift provided by buoyancy and vertical thrust is incremented by an aerodynamic component, developed by the double fuse-shaped body of the airship. The four thrust-vectoring propellers allow to control the lateral-directional attitude of the airship through the variation of the rotational speed (RPM) together with the rotation of the supporting vertical arms.

A. Control Allocation System (CAS)

Commands are generated by the pilot through an innovative cockpit, consisting of two throttles and a three-DOF joystick. The pilot inputs are processed and re-allocated by the Control Allocation System in order to generate the desired commands in terms of propeller rotational speeds and orientations of the four thrust-vectoring propellers. The Control Allocation System has been firstly designed and modeled in the FCC of the ETF Simulator and then integrated in the on-board computer of the airship prototype. The control strategies within the FCC have been developed for the two possible flight conditions: forward flight and hovering with/without wind. In forward flight, the joystick commands the orientation δ of the four thrust-vectoring propellers for the lateral and directional maneuvers, as well as the differential variation of the angular rate of all the six propellers, generating the differential thrusts ΔT_{ax} and ΔT_{axVT} for the longitudinal maneuvers. The allocation strategy of the longitudinal control, shared between the forward and vertical propellers, has been purposely designed and scheduled to improve both the efficiency and the potentiality of this command, optimizing the airship performance in the whole speed range[2]. The two throttles act on the collective rotational speed of the four thrust-vectoring propellers and the two vertical axis propellers. In particular, the variation Δn of the propeller rotational speed in rounds per minute (RPM) is proportional to the square root of the throttle input δ_{th}. This relationship has been imposed to obtain a linear relation between the command action and the generated thrust as the propeller thrust is proportional to the square root of the angular rate, according to the first *Rénard formula*[11]. All the six propellers can work in reverse mode with reduced efficiency. Moreover, the maximum collective thrust commanded by the throttle input δ_{th} is only a reduced percentage of the total available thrust, while the remaining available thrust is dedicated to the commanded maneuvers, which are thus always achievable even when the throttle command δ_{th} is maximum.

The general scheme of the control strategy in forward flight is illustrated in Figure 7, in which it is highlighted the position of each propeller with the corresponding control action generating positive pitching, rolling and yawing moments, respectively, for longitudinal, lateral and directional maneuvers. In particular, the forward and vertical collective thrusts T_{thFW} and T_{thVT} are linearly controlled by the two throttles, while the differential thrusts ΔT_{ax} and ΔT_{axVT}, as well as the propeller orientations δ are generated through the joystick action. For lateral and directional maneuvers, the joystick firstly commands the axial ΔT_{ax} and normal ΔT_n thrust increments with respect to the propeller rotational axis, successively, the FCC processes and re-allocates these signals in order to provide the corresponding orientation angles δ of the propellers. In Figure 7 the positive rotations δ of the thrust-vectoring propellers are assumed to be in a clockwise direction.

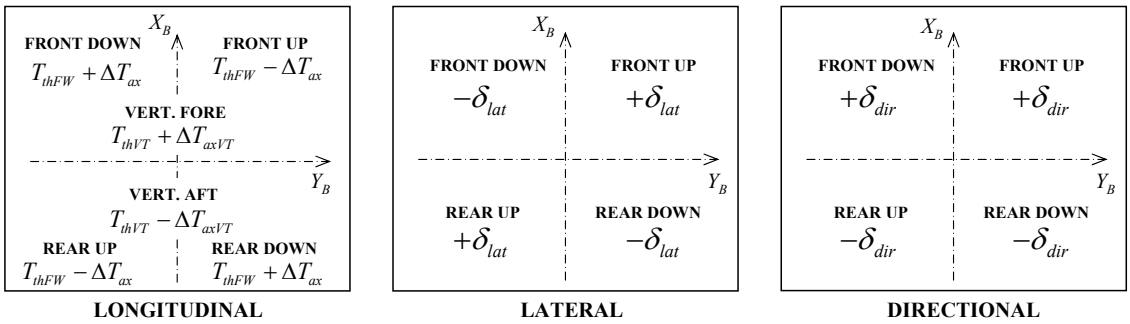

Figure 7. Control strategy for longitudinal, lateral and directional maneuvers in forward flight.

The control strategy in hovering condition is almost analogous to the forward flight except that all the thrust-vectoring propellers are collectively oriented in the wind direction through the angle δ_{coll}. In addition, the directional command is accomplished through a differential thrust of the right and left forward propellers with respect to the longitudinal wind axis X_w. In this way, the pilot continues to command the desired moments in the body-axis reference frame X_B–Y_B, while the FCC processes these commands and computes the corresponding control inputs in the wind-axis reference frame. The relationship between the two sets of command inputs depends exclusively on the collective orientation angle δ_{coll}. This control allocation strategy, implemented in the on board computer of the prototype, is inherently based on two main assumptions. Firstly, the longitudinal thrust component, produced by the forward throttle δ_{th}, has to be preserved during any longitudinal, lateral and directional maneuver in order to guarantee the desired motion and speed of the airship. Secondly, the longitudinal and lateral commands have to be proportional to the related moments to manage forward flight and hovering with the same control philosophy[6].

B. Command Lines

The command action is performed through ten brushless motors, driven by two dual-channel and six single-channel control boards. Each thrust-vectoring propeller, in particular, is controlled by two distinct command lines: the orientation one, which rotates the vertical arm and thus the thrust direction, and the propulsion one, which is responsible of the propeller regime variation. The orientation line has five elements: the dual-channel control board, the brushless motor, the gearbox, the encoder and the transmission system. Due to the peculiar propeller setup, this control board is shared with the corresponding command line of the adjacent front or rear propeller and is interfaced to the computer through the RS-232 standard interface.

The propulsion line is made up by four components: the slip rings, the control board, the brushless motor and the encoder. In this case the control board is dedicated and is connected to the computer through the RS-422 standard interface, which should protect the communication capability against the signal degradation caused by the slip ring presence.

C. Avionic System

The avionic system is essentially based on the following devices:

- On-board Computer (PU);
- Navigation Sensor Unit (NSU);
- Avionics Battery;
- Propulsive Battery;
- Electrical Power Generation (EPG)

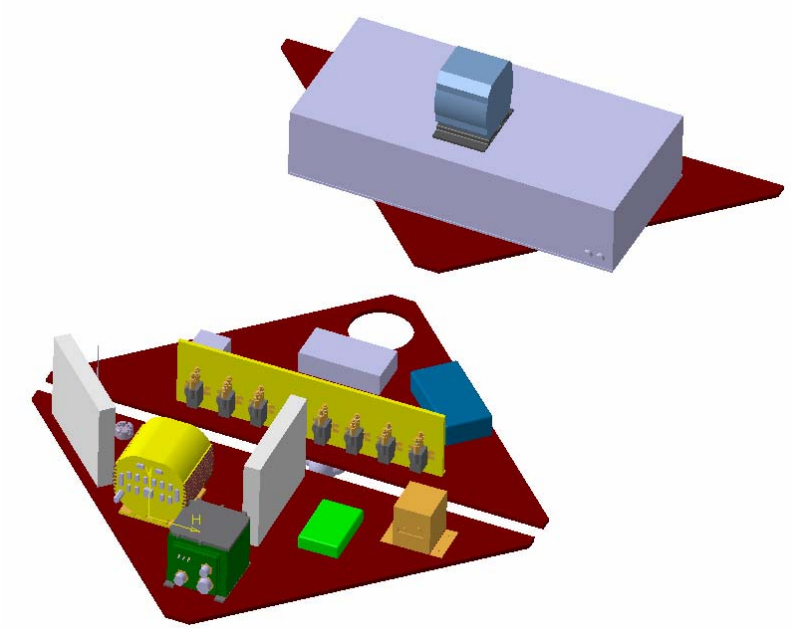

Figure 8. The avionics panels.

The on-board computer is based on a *x86* compatible processor, installed on a PC/104 mainboard. The Navigation Sensor Unit (NSU) basically gathers all the sensors used for the control and navigation task, such as the Inertial Measurement Unit (IMU), the gyroscopes, the accelerometers, the Air Data Sensors and a GPS card. Other sensors are purposely distributed in the airship to monitor crucial parameters, such as the temperature of the batteries and the weight on the landing gear. All these sensors are directly interconnected to the main computer through proper I/O cards. Other devices are purposely inserted for safety reasons, to command the shut-off valves or dissect parts of the electrical circuit in emergency situations. These devices are commanded directly by the pilot through a relay board on the on-board computer or automatically through the EPG, in case of fault of the on-board computer.

The five apparatus are disposed on two triangular panels fixed in the bottom part of the central structure of the airship, as shown in Figure 8. The location of the avionic system panel in a central position has been selected in order to make easier the connections to the other on board systems without affecting significantly the airship Center of Gravity position.

IV. Hardware/Software Architecture of the On-Board Computer (PU)

The innovative control system of the ETF airship requires a good computational capability to implement the complex control logic. Pilot commands have to be received via a radio-link and then converted into the actual commands and sent to the actuators. This conversion is performed by the CAS described in Section III.A. The software running on the on-board computer has manifold tasks, such as the CAS implementation, the connection with the ground station and the interface with the avionic system and actuators. In the following subsections, the hardware and software architecture of the on-board computer, named Processor Unit (PU), will be described in detail.

A. Hardware Architecture

The PU is based on a PC/104 mainboard, a solution combining reliability, compactness and easy implementation. The mainboard includes a low-power *x86* compatible processor and basic interfacing capabilities (RS-232, Ethernet and USB ports, mouse and keyboard input). The hard-disk is a solid-state Flash-Disk, which is typical for embedded systems that do not require the larger capacity of magnetic drives but must be highly reliable. To effectively interface the PU with the other components of the avionic system and the actuators, several additional I/O boards are necessary:

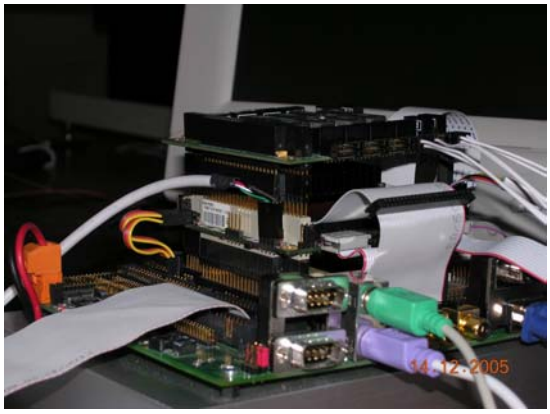

Figure 9. Prototype version of the PU.

- an Analogic/Digital acquisition board, used to convert voltage signals to a digital format accessible by the PU. This board is used to acquire signals coming from the Weight-on-Wheel analogic sensors, basically load cells measuring the load distribution on the airship tripods;
- a serial board comprising eight serial RS-232/RS-422 ports and four RS-232. These ports are used to interface the PU with the actuators (engine controllers) and with some avionic devices (NSU);
- a relay board, including twenty software-activated relays, nine of which used to command the Remote Control Switches controlling the on-board power distribution and two of which used to activate the emergency shut-off valves. The board also includes twenty optoisolated digital inputs, two of which are used to acquire the signals from the shut-off valve proximity sensors.

This configuration, as illustrated in Figure 9, includes all the interfaces needed for the ETF prototype, and leaves several spare inputs and outputs, which might become necessary in case some extra device is added.

B. Software Architecture

The PU Operating System is *QNX*[12], a real-time OS designed specifically for embedded systems. Taking advantage of a true micro-kernel architecture, it ensures deterministic execution of the running applications. This is a crucial issue in a safety-critical system, in which every task must be executed at the expected time and without delay. The real-time nature of *QNX* also allows to synchronize the running applications in a significantly better way than what can be accomplished using a traditional OS.

The software is structured to optimize functionality in the several different tasks it has to accomplish. The main element, named "*Core*", is basically the real-time version of the *Simulink* model representing the CAS. The *Core* interfaces with the other applications through the use of Shared Memories, which are reserved portions of system memory accessible to different running applications. The hardware interfaces are controlled by several low-level applications specifically designed to achieve input and output requirements, while interacting with the *Core* through the Shared Memories. Figure 10 shows the software architecture: it is evident how the *Core* component is heavily relying on the low-level applications, as these are its only means of interfacing with external hardware. There are seven low-level applications, interfacing with the *Core* through nine Shared Memories:

- *ClientRX* receives data from the Ethernet interface (radio-link) and writes it to the Shared Memory *joystick.shm*;
- *WriteRelay* reads data in *engine.shm* and consequently directs the relays;
- *WriteSerial* interfaces with the engine controllers through the serial ports, sending the commands contained in *engine.shm* and writing replies received in *query.shm*;
- *NSU* receives and decodes data sent by the NSU avionic device, then writes it to *nsu.shm*;
- *WriteSensori* reads values from the A/D acquisition board (WOW sensors) and digital values from the relay board (proximity sensors) then writes everything into *sensori.shm*;
- *EPG* is in charge of the Ethernet communication with the EPG avionic device, which monitors the batteries and is used as a partial backup of the PU for safety procedures;
- *ClientTX* reads data from all Shared Memories and sends it to the ground station.

Having more than one application writing data into a Shared Memory can cause severe conflicts, so it is very important to avoid this eventuality. There are no problems with reading operations, in fact, several applications can read data at the same time.

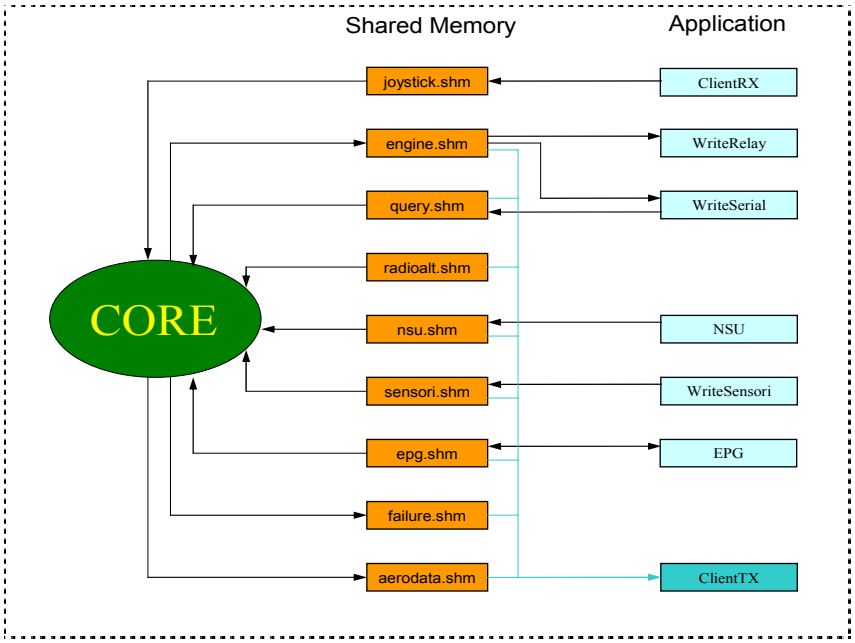

Figure 10. Software architecture of the PU.

The *Core* component is developed using automatic code generation techniques. The entire *Simulink* model of the Control Allocation System is converted into an embedded executable code using the RT-LAB software package. The generated executable code uses Shared Memories as a mean of input/output communication, as the physical interfaces are controlled by the low-level applications. These in turn are developed using the QNX Integrated Development Environment. Theoretically, it would be possible to write *Simulink* S-functions to interface directly the core with the hardware devices, using the same automatic code generation process. However, this solution is not attractive as it involves creating complex fully inlined S-functions, which may be much more demanding and critical than writing the low-level applications. In addition, using external applications to drive the hardware peripherals gives advantages in terms of system modularity and software reliability.

These advantages certainly make this solution preferable, especially when the system must be certified as an avionic apparatus. At present, it is unfeasible to obtain certification for automatically generated software. According to the DO-178 rules, in fact, all the software critical functions must be implemented through purposely structured and commented custom written code. The implementation of safety procedures is also very important, as these have to ensure a predictable behavior in case of failure.

V. System Verification & Validation

When thoroughly exploited, modeling and simulation can drastically reduced the risk of confronting significant problems during the system integration phase. To optimize the process, an effective strategy consists in identifying the criticalities and developing a set of test harnesses, made up by series of hardware layouts, verification procedures and test cases.

A. Interpretive Simulation

Model-based design has proven to be an effective framework for the development of activities related to guidance, navigation and control. In particular, programming tools such as *Matlab* and *Simulink*, which strongly rely on the model-based design, have become a must for designing control systems and reusing the models in multiple activities. They provide the designer with the ability to manage complex designs by segmenting models into hierarchies of components in an interactive graphical environment, which allows the designer to easily configure and check signals, parameters, and model properties. Once a set of equations has been written, hence, it is straightforward to implement them according the control scheme of Figure 1, and run simulations to diagnose bugs or unexpected behaviors in the control design. In this context, the interpretive simulation task is to test the system in all the operational modes and under environmental conditions which may be difficult or impossible to access for system tests (such as icy roads in the middle of summer, or the conditions of outer space, etc…). Simulation allows intricate test sequences to be performed quickly and repeatably at relatively low cost, without risking loss of life or valuable assets. Although interpretive simulation is essential to verify the consistency of the control laws, however, it does not provide full understanding of the mechanisms driving the real system. Usually, in fact, the simulation runs in non-real time, as the process is sequential, even when a certain degree of parallelism is brought about by the PC processor. Variables are usually in double-precision floating-point format and are exchanged between plant and controller through a direct communication. Information about time delays, digit truncation, precision or bit loss are neglected and this deficiency is enough to justify the need for further tests.

B. Real-Time Validation of the Automatically Generated Code

After modeling the CAS in the *Simulink* environment, the RP technique can be effectively used to test the code in operative conditions. This involves the generation of automatic code in C, which can be compiled and executed in the *QNX* real-time environment. In this phase the communication is bi-directional between the *Simulink* plant model and the CAS real-time version, so that a commercial computer, with limited I/O capabilities, can be effectively used as the target platform. Plant and CAS run on two distinct PCs, controlled by different operative systems and connected by a simple LAN cross-over cable. The translation from the *Simulink* code to the C code, might bring about some discrepancies which might drastically affect the command action. Therefore, suitable tests must be scheduled to assess these discrepancies and evaluate the correct functioning of the embedded code. This implies comparing results obtained from the real-time execution with the ones derived from the non-real-time interpretive simulation. Obviously, this comparison must be accomplished under the same inputs, which can be time-histories composed with the superposition of characteristic commands (step, doublet, chirp, etc…) or with maneuvers recorded from a standard input device (joystick).

Initially, a certain degree of inconsistencies can be expected, due to pathological problems, which can be easily solved using preventive measures, such as the insertion of dead zones, saturation thresholds and rate limiters. When these errors are fixed, satisfactory results can be achieved, as shown by the example of Figure 11. In particular, the upper diagram reports the actual comparison of the output values, while the lower diagram shows the difference between the real and the estimated outputs. In this case, there are only minor discrepancies, such as a peak, which is clearly marked and might be attributed to a time delay caused by the input dead-zone crossing.

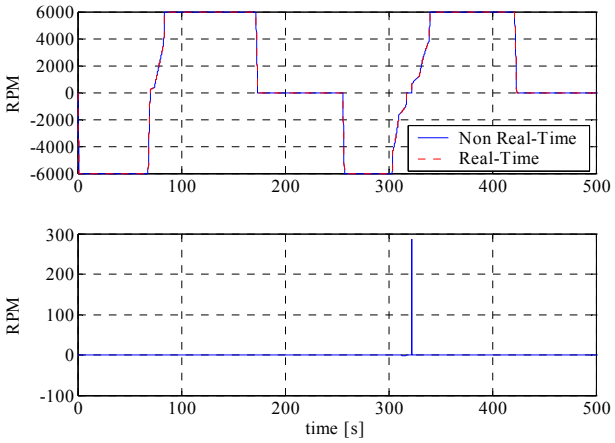

Figure 11. Example of a comparison between the *Simulink* non-real time execution and the C real-time code.

C. Software Stress Analysis

Once the real-time behavior of the automatically generated code has been evaluated, attention can be focused on a series of procedures, based on the SIL technique, which has to stress the PU software through a set of specific and rigorous tests. A valid approach to verify the functionality of the *Core* component, in fact, is to test it under all the predictable conditions and the widest range of unpredictable conditions. In particular, different sets of input commands are generated and fed to the pertinent Shared Memory, the *joystick.shm*, which gathers all the signals

coming from the pilot cockpit of the ground station. Successively, the output signals of the PU *Core*, in terms of propeller rotational speeds and orientations of the four thrust-vectoring propellers, are collected and statistically analyzed in order to avoid possible anomalous situations, which could be harmful to the on-board hardware of the airship. The embedded *Core* software is stressed not only with regular input signals but also with random or senseless signals, such as input commands out of the operating ranges. These anomalous signals simulate, for example, a communication error or an execution error of the external devices and allow to deeply and efficiently test the *Core* behavior also in critical situations requiring emergency procedures.

D. I/O Interface Setting and Testing

A complex control system, such as the one being developed for the ETF airship, usually communicates with a large number of avionic devices and actuators. This means that the various apparatus must be interfaced with the PU I/O boards, which must be properly set and tested. The RP and SIL techniques used to validate the automatically generated software, which implements the CAS, are not suitable to accomplish this task. To effectively set and test the interfaces, it is crucial to have a properly configured target system. While RP focuses on testing the real-time software using a generic platform, on-target RP is based on the execution of the previously tested software on a specific platform, equipped with the necessary I/O interfaces.

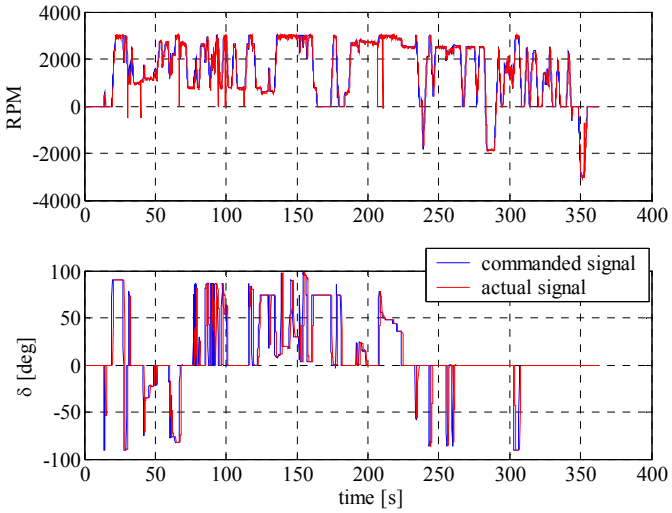

Figure 12. RPM and Orientation of the Front-Up Propeller.

On-target RP involves assembling a prototype version of the on-board computer, where the *Core* component of the CAS is executed. The communication interfaces of the avionics devices and the on-board hardware can be allocated on four different I/O boards furnished with serial ports, analogical inputs, digital inputs and relay outputs. As these boards feature different communication protocols and have a certain degree of customization, the setting phase implies that each interface is treated separately. The interface is thus developed tuning every single protocol for the devices that have to be connected. The use of the on-target RP allows not only to develop the interfaces in an efficient manner, but also to thoroughly test them. It is possible to verify whether the communication protocols are exactly implemented, whether the data exchanged with the external devices are properly formatted and if the variables are actually confined in the predicted range.

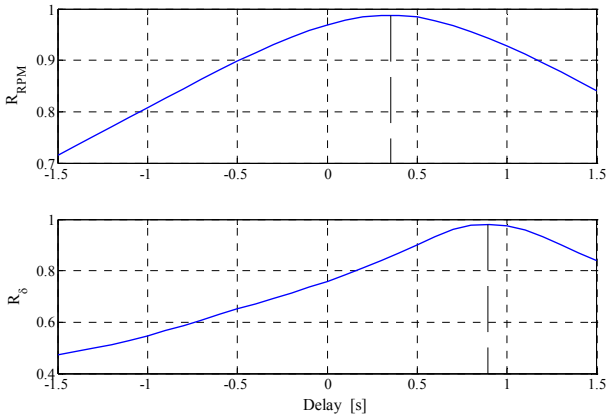

Figure 13. Batch cross-correlation for propulsive and orientation motors.

E. System Integration

After these first analyses on the embedded software and interfaces, specific test rigs can be designed for the HIL testing of all the available external hardware. In particular, a test bench has been assembled for the propulsion motor and another one has been dedicated to the orientation motor. Finally, a third test bench, resulting from the previous ones, has been conceived to evaluate the integration of the embedded control software running on the PU with all the external hardware, considering also synchronization problems and casual conflicts. Hence, it has been possible to test entirely the interaction between the PU and the maneuvering system, made up by six propulsion motors and four orientation motors. In fact, although the test bench is principally made up by just one propulsive motor and one

orientation motor with the relative components, all the serial I/O ports, RS-232 and RS-422 standard protocols are present and the missing motors are simulated through a dedicated real-time model. In this way, the embedded control code has to manage in any case different hardware components through different serial I/O ports and has to run different software to implement the communication protocols. The tests to evaluate the system integration have been carried out by using pre-recorded command inputs, which correspond to a specific maneuver of about 360 *s* accomplished by means of the ETF Simulator. For each test the commanded signal coming out from the CAS and the actual signal measured by the encoders are recorded with a sampling frequency of 10 *Hz* and processed for performance analysis and evaluation. Figure 12 shows respectively the rate number (*RPM*) and the orientation (*deg*) of the front-up propeller. In order to evaluate the degree to which the two commanded signals are correlated to the corresponding actual signals, a statistical signal processing analysis has been accomplished by using the following cross-correlation function:

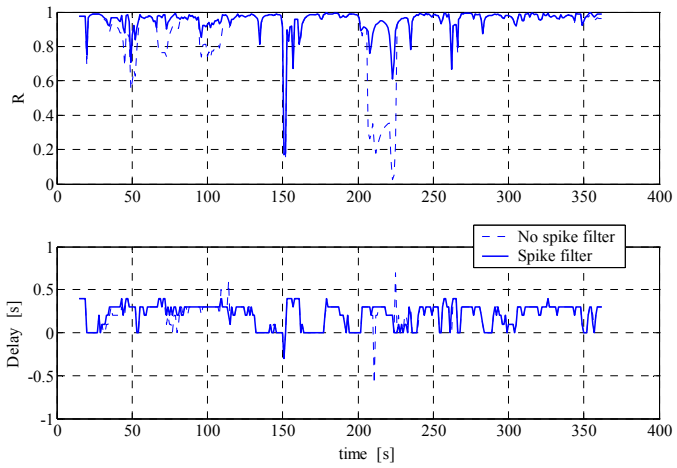

Figure 15. On-line Correlation and Time Delay for the RPM signals of the propulsive motor.

$$R(\tau)=\frac{\sum_i\left[\left(com(i)-\overline{com}\right)*\left(act(i-\tau)-\overline{act}\right)\right]}{\sqrt{\sum_i\left(com(i)-\overline{com}\right)^2}\cdot\sqrt{\sum_i\left(act(i)-\overline{act}\right)^2}} \tag{1}$$

where $com(i)$ and $act(i)$ are respectively the commanded and actual signal series, $\overline{com}$ and $\overline{act}$ are the mean values of the corresponding series and τ is the correlation delay. The response characteristics of the propulsion and orientation motors in terms of time delay can be estimated through a batch analysis, by calculating the correlation over the entire signal series $com(i)$ and $act(i)$. From Figure 13 it can be noticed that the time delay of the propulsive motor is about 0.3 *s*, whereas the time delay of the orientation motor is about 0.9 *s*. The same concept has been successively employed to implement an on-line correlation analysis in the PU embedded software, in order to monitor the motor functioning and detect possible failures or commands erroneously executed. This analysis is executed with a frequency of 1 *Hz*. At each sample time, the correlation analysis inspects the recorded signals on a dynamic window of 15 *s* and computes the relative correlation series, after a suitable filtering process that cancels all the spikes in the measured signals. The resulting on-line correlation series are then processed to obtain the maximum cross-correlation values and the corresponding time delays. Figure 14 shows an example for the commanded and actual *RPM* signals of the propulsive motor of Figure 12, with and without the filtering process of the measured

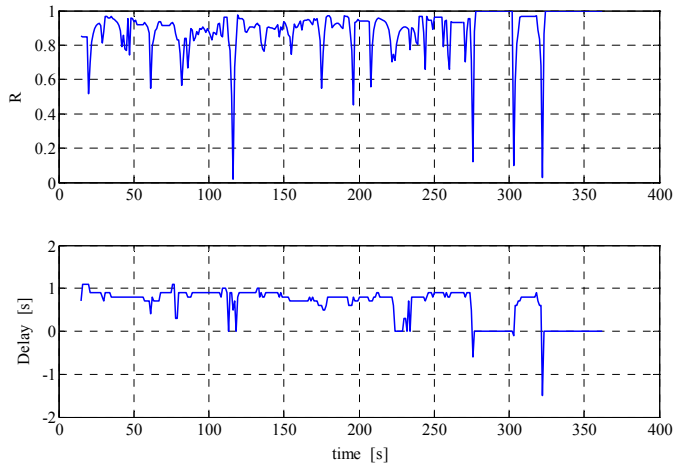

Figure 14. On-line Correlation and Time Delay for the angular signals of the orientation motor.

signals *act(i)*. Conversely, the on-line correlation and time delay values relative to the orientation motor are presented in Figure 15. These two important on-line parameters allow to check the correct working of the propulsion and orientation motors, as well as to implement emergency procedures or simply visual warnings in case their values are anomalous or drop under a fixed threshold.

Acknowledgments

Financial support for the authors was provided through a grant from the Regione Piemonte within the Regional Announcement for the Applied Scientific Research 2004, Project Code E65.

References

[1]Inventors: Gili, P.A., Battipede, M., Icardi, U., Ruotolo, R., Vercesi, P., Owner: Nautilus S.p.A. and Politecnico di Torino, "Aeromobile controllato da mezzi di spinta per la direzione di rotta, ad alta manovrabilità e bassa sensibilità alle raffiche di vento laterali," N. PCT/EP03/08950, filed 11 August 2003.

[2]Battipede, M., Lando, M., Gili, P.A., Vercesi, P., "Peculiar Performance of a New Lighter-Than-Air Platform for Monitoring", *Proceedings of the AIAA Aviation Technology , Integration and Operation Forum*, AIAA, Reston, VA, 2004.

[3]Battipede, M., Gili, P.A., Lando, M., "Prototype Assembling of the Nautilus Remotely-Piloted Lighter-Than-Air Platform", *Proceedings of the AIAA Aviation Technology , Integration and Operation Forum*, AIAA, Reston, VA, 2005.

[4]Ledin, J., "Simulation Takes Off with Hardware", *Embedded Systems Programming*, CMP Media LCC, 2002.

[5]Battipede, M., Gili, P.A., Lando, M., Massotti, L., "Flight Simulator for the Control Law Design of an Innovative Remotely-Piloted Airship", *Proceedings of the AIAA Modeling Simulation and Technologies Conference*, AIAA, Reston, VA, 2004.

[6]Battipede, M., Gili, P.A., Lando, M., "Ground Station and Flight Simulator for a Remotely-Piloted Non Conventional Airship", *Proceedings of the AIAA Guidance, Navigation and Control Conference*, AIAA, Reston, VA, 2005.

[7]Battipede, M., Gili, P.A., Lando, M., "Control Allocation System for an Innovative Remotely-Piloted Airship", *Proceedings of the AIAA Atmospheric Flight Mechanics Conference*, AIAA, Reston, VA, 2004.

[8]RT-LAB™, *Real-Time Modeling Software Package*, Ver. 7.2, Opal-RT Technologies, Inc., Montréal, Canada.

[9]Pratt, R.W., *Flight Control Systems: Practical Issues in Design and Implementation,* IEE Control Engineering Series 57, 2000.

[10]DO-178B, *Software Considerations in Airborne Systems and Equipment Certification*, Radio Technical Commission for Aeronautics, Inc. (*RTCA*), 1992.

[11]Lausetti, A., Filippi, F., *Elementi di Meccanica del Volo*, Vol. 1, ed. Levrotto & Bella, Turin, 1955, pp. 61-88.

[12]QNX Neutrino RTOS V. 6.3, QNX Software System Ltd., Ottawa, Canada, 2004.