Distributed Real-Time Hardware- and Man-in-the-loop Simulation for the ICARO II Unmanned Systems Autopilot

LORENZO POLLINI*, VALERIA PARNENZINI, MARIO INNOCENTI
Department of Electrical and Computer Engineering
University of Pisa
Via Diotisalvi, 2 56100 Pisa
ITALY
*lpollini@dsea.unipi.it http://www.dsea.unipi.it

Abstract: The autopilot market for small and research UAVs offers several products, but most of them, although widely configurable or even open-source, do not constitute a practical and safe development system for custom guidance, navigation and control systems. The ICARO project aims at providing the small UAV community with a valid autopilot alternative. The ICARO autopilot exploits rapid control system prototyping techniques and immersive manned simulation with the possibility of testing the autopilot using the Hardware-In-the-Loop (HIL) approach. This paper describes the hardware-in-the-loop and man-in-the-loop simulator for the ICARO II platform together with the synchronization protocol we developed to keep simulator and autopilot synchronized. Experimental evidence of the effectiveness of the synchronization protocol is given.

Key-Words: Distributed real-time simulation, hardware-in-the-loop, Man-in-the-Loop, Unmanned systems

1. Introduction

The autopilot market for small and research UAVs offers several products; the Micropilot MP2028, the CloudCap Piccolo II, the Procerus Kestrel Autopilot, the Mavionics MINC Autopilot System are just examples of widely known and used autopilots. Recently open-source experiments like the ArduPilot project arose. The above mentioned autopilots have similar input output interfaces, serial ports, digital and analog I/Os, pulse width modulation generation and capture. All of them were born, even if with possibilities of parameterization of their functions, as monolithic systems with little possibilities of real customization. All the open-source projects instead offer full customization but the community development process often led to code difficult to modify and maintain; in addition safety of operation is a relevant concern since no real validation and qualification of the various software releases is ever done. Recently, pushed both by market requirements, and by research needs (most customers of those autopilots are universities or research centers) autopilot producers are opening their products a little so that users can customize them with external routines, and additional functions. Some of them are starting to provide hardware in the loop functionalities. The main limitation of these autopilots is that they are not sufficiently open as platforms for implementation and development of custom guidance, navigation and control (GNC) system; furthermore testing, for instance, of advanced algorithms like fault detection and accommodation system within one of these commercial autopilots, becomes almost impossible; another relevant issue is that hardware redundancy is only little exploited as a mean for safety as, instead, is standard practice in the aerospace field. Commercial autopilots offers a proprietary configuration system which requires several time to become familiar with; on the contrary, Computer Aided Control System Design (CACSD) tools like Mathworks Matlab and Simulink are widely used and appreciated amongst the community of researchers working in GNC systems design. In addition, they often offer rapid prototyping of control systems by automatically generating software for a large variety of targets, from generic C code, to specific RT-Linux code (in several of its variants) or even directly for CPUs of the embedded world like automotive PowerPCs, DSPs etc. that do not require an Operating System to run. Furthermore, modern unmanned vehicles require advanced functionalities that are not limited anymore to stability augmentation or reference command tracking but their increasing autonomy requires implementation of very complex functionalities; examples are coordination algorithms for swarming [1] or formation flight [2-4], obstacle detection and avoidance [5-6], vision based navigation [7-9] etc. The algorithmic

ISBN: 978-1-61804-134-0

complexity of these functionalities requires extensive tests in off-line simulation; but, when real-time performance is required, implementation issues and timing interaction with the other functionalities become relevant as well. An Hardware-In-the-Loop (HIL) simulator is thus necessary in order to test the actual operational software when running in the dedicated hardware. Another relevant issue for unmanned systems operation is human-machine interface design. In order to test all the system functionalities in complete safety, from standard operations like enabling/disabling certain control loops, switching operational mode (for instance from manual control to aided or fully autonomous control), to engaging shared control strategies like recent haptic piloting support systems [10-13][5-6], it is necessary to provide the pilot with the same input/output interfaces that will be used in operations, and to feed him/her with a realistic synthetic environment[14], and to let him operate the actual remote control system (i.e. the autopilot) with, possibly, the same communications delay of real operations [15-16]. Thus a Man-In-the-Loop (MIL) simulator is needed. Upon these considerations, we started in 2007 the ICARO project [17]: the development of a new autopilot capable of overcoming all these limitations and of providing a development environment completely open and based on Matlab and Simulink, capable of real-time HIL and MIL simulation to support designers in the development phase. ICARO Autopilot is now at its second generation and the third is coming soon. This paper presents the development of the hardware-software infrastructure of a real-time distributed simulator for the complete system: simulated vehicle dynamics, 3D realistic view for man-in-the-loop simulation and the actual autopilot hardware connected via a communication bus with the rest of the system. Within this framework thus, the device under test will be the autopilot hardware. The goal of the system is twofold: to test and debug the control and high level management algorithms after their automatic implementation on the autopilot hardware, and to allow the pilot to train to operate the vehicle and test any operational mode, even in early development stages, in complete safety.

2. The ICARO II autopilot

University of Pisa started the project ICARO in 2007: an internally funded project which aims at developing a general purpose embedded computing unit, with native support for several kind of data buses, and thus for a large number of sensors and payloads, with a certain degree of redundancy, designed with rapid prototyping of new guidance, navigation and control ideas in mind, relatively low-cost but yet reliable and powerful, and not specifically dependent on any sensor suite. In fact the ICARO autopilot hardware is flexible enough to be employed in a range of unmanned systems applications: we already employed the family of ICARO autopilots onboard aircraft, quadrotor and multi-rotor helicopter, small off-road vehicles.

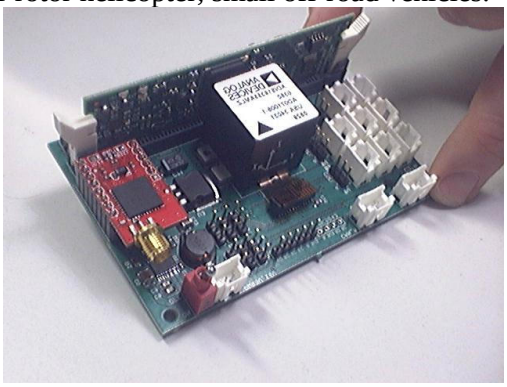

Figure 1. ICARO II autopilot with a GPS and an Analog Devices 6-DOF IMU mounted on.

The ICARO autopilot is capable both of working as a highly configurable autopilot system, similarly to other autopilots which allow in-flight configuration of gains in pre-defined control loops, and as a completely open system which allows to build a complete guidance, navigation and control system either from scratch or with the aid of predefined building. Thus ICARO must be regarded as an open platform to develop all kind of research about UAVs, and not only as an autopilot. The first generation of autopilots were based on an automotive PowerPC architecture (with floating point unit and several internal peripherals and communication devices) [17]. The second family called ICARO II, or ICARO Light was designed to be smaller, lighter and cheaper with a modern and powerful DSP (300 MFLOPS) as computing core. The system, as the previous generation, is not linked to a specific sensor suite (inertial, GPS, etc.) or communications channels but may be connected via its I/O peripherals to any device, provided that low level code C is written as interface. ICARO II interfaces are: UART, SPI, I2C, Analog/Digital I/Os, PWM generation/capture, and finally CANBUS. This latter interface will be used as a high speed communication channel for the fast exchange of the data needed to implement the HIL simulator and to keep the various elements of the distributed simulation synchronized. UART is probably the most common type of serial line (RS-232, TTL or LVTTL) and almost all sensors have

ISBN: 978-1-61804-134-0

UART as an option. I2C, known as two-wire interface, is a common bus used by many devices and integrated circuits (and also by many MEMS gyroscopes, accelerometers and magnetometers). SPI is a very common serial interface used at system level as interface between CPUs and devices. The output signals of the autopilot, namely its commands to external devices like actuators or payloads, is provided by PWM signals, commonly used to drive control surfaces servomotors, some simple digital I/O lines, or all the other bi-directional interfaces (UARTs, I2C, SPI). The ICARO II CPU can be programmed by any compiler which supports it; standard C language is the choice for low level programming and full access to hardware resources; alternatively a rapid-prototyping tool can be used. The DSP we used, for instance, is fully supported by the Mathworks Embedded Coder known in the past as RealTime-Workshop (RTW). With the aid of RTW, the GNC algorithms, developed under Simulink, can be simulated first on Matlab/Simulink, and then they can be compiled, downloaded and run in real-time on the actual CPU. This solution makes the development and prototyping of algorithms much faster and more intuitive. The power of the rapid software prototyping approach is that we were able to port the core autopilot functionalities we developed for the first ICARO autopilot, based on the PowerPC architecture to the ICARO Light family with minimal efforts; basically only the software interfaces toward the hardware peripherals (serial ports, canbus, digital and analog I/Os) had to be hand coded while all the GNC components needed no modification. The flexibility of the proposed autopilot will allow extending its usage to stabilize and control other systems [18,19].

3. Hardware In the Loop Simulation

Hardware in the loop simulation is a common approach for validation and qualification of functionalities of pieces of actual hardware (eventually with firmware) before installation into the operational environment. For the scope of this paper, the piece of hardware and associated firmware under test is the autopilot. Modern unmanned vehicles require advanced functionalities, thus the control and mission management software may be very complex and difficult to validate and qualify; an hardware in the loop simulator is a useful tool since it allows testing of the actual hardware/software combination that will be employed in operation. An autopilot has several sensor interfaces that must be emulated: inertial data (from gyroscopes and accelerometers), magnetic field data, GPS, atmospheric pressure (for measuring altitude) etc.; all these sensorial data must be simulated, for both its informative part and noise characteristics, and sent to the autopilot hardware in a timely manner so that, when the autopilot expects to have new sensor data sampled from its actual sensors, it finds instead data sent by the simulator. After elaboration of sensorial data, the autopilot produces an output for its actuators, this output may be given to real actuators and, in parallel, sent to the simulator so that it performs a simulation step (of a simulated aircraft or vehicle in general) and produce new sensorial data. In order to perform a coherent simulation, the clocks inside the autopilot and the simulator must be kept synchronized. The next section describes the synchronization protocol we developed for this particular application.

3.1. Synchronization protocol

The autopilot software is constituted mainly by a rate monotonic scheduler that, using the a system timer, schedules the various tasks that implement guidance, navigation, control, and mission management algorithms, and guarantees that each task runs synchronized with the real time. Currently the ICARO II firmware is made of 5 tasks (running at 500, 200, 100, 20 and 5 Hz), and 12 Interrupt handlers (that implements low level communications with peripherals, i.e. perform the functions of the device drivers in an Operating System) with re-entrant asynchronous code. The simulator instead is a PC program, that simulates the vehicle dynamics: reads autopilot actuator commands, simulates the system (actuators, vehicle dynamics, external disturbances, and sensors dynamics), and produces a simulated sensorial output for every sensor of the autopilot. In order to have a coherent simulation, that is with time flowing at the same rate in both systems, the autopilot clock and the simulator's clock must be kept synchronized for the entire simulation duration. There are several options for performing this task[14], the choice of the best one depends mainly on the specific problem; it is usually unadvisable to run the two systems with two different clocks and synchronize them at single instants of time (like when starting the simulation) due two unavoidable clock drifts. If one of the two systems to be kept synchronized does not have the need to run its own clock, it is advisable to synchronize both on the same

clock[14]. This is the case of typical HIL configurations were the real part (the hardware) has its own intrinsic clock and the simulated part instead may be slowed down to run at the same pace of the hardware; this should be done at the maximum possible frequency. When, as in our case, data must be exchanged with a communication channel between the two systems, these communications may serve as both data exchange and synchronization point. Figure 2 depicts the communication scheme implemented in the presented HIL simulator: Autopilot and Simulator communicate via CANbus. The Reception (RX) block inside the Simulator is where synchronization takes place. The rationale of the proposed synchronization scheme can be described as:

"Given two bi-directional communicating systems, system A that runs with its own real-time clock and cannot be slowed down, and system B without a real time clock; send data from A at a constant rate (this data will be deterministically sent at time instants equally spaced by ΔT), block execution of B until reception of data from A, advance time of B by ΔT, perform any operations, then loop. If B is fast enough, the two systems will be synchronized with an accuracy of ΔT seconds."

The macro-operations for both systems, as seen, from a communication/synchronization standpoint, are three: reception of data, elaboration of data, and transmission of the results. Thus, declination of this concept into our HIL environment leads to the following algorithms, presented as pseudo-code:

```
//Autopilot code (system A)
While (1)
    S = GetSensorData();
    C = ElaborateData(S);
    SendCommandData(C);
    WaitForRTSync();
```

```
//Simulator code (system B)
While (1)
    While    (data_not_received    ||
timeout)
        C = GetCommandData();
    S = SimVehicleAndSensors(C);
    SendSensorData(S);
```

According to the actual speed of execution of the second system (B), two different synchronization variants exists. Two cases are possible:

1. System B is fast enough to run completely in the idle time of system A;
2. System B execution time is less then *ΔT* but is not fast enough;

In the first case, it is possible to let B wait for A to send input data, and then work on freshly received data; the net result is that `SimVehicleAndSensors` always executes after `ElaborateData`. Thus the autopilot always find "fresh" data at the next `GetSensorData` call. When the simulator hardware is not fast enough to execute completely in the idle time of the autopilot, or simply the amount of autopilot idle time is too little, it is advisable to insert an artificial but deterministic delay between the data used by the autopilot and the data sent by the simulator. This can be simply achieved by double buffering the data that is received by `GetSensorData`. This means that the simulator may take up to an entire time step to elaborate the data; it is sufficient, in order to maintain synchronization, that it sends sensors data before the next call to `GetSensorData`.

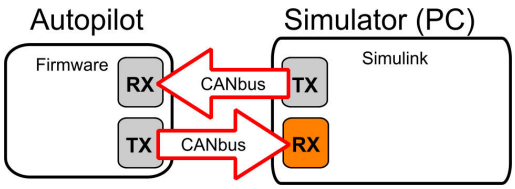

Figure 2. Sample communication scheme.

Although this solution may appear worse than the first synchronization variant, it is often not, but instead may also produce a more realistic simulations of the sensors. For instance, the ICARO II Autopilot, in its most complete configuration has two different inertial units (that guarantee a certain degree of safety to fault of one of the two); both are sampled at 500 Hz, one via SPI, the other via I2C; the I2C bus, in our case, runs at 400 Kbits and bus load is near to 70%, this means that the communication time is close to $1/500^{th}$ of seconds. Active wait for the complete read cycle for all the sensors on the I2C bus is obviously impractical, thus it is necessary to use interrupts to perform all the communications with the inertial sensors. In order to avoid incoherency of the data, when an elaboration data starts, it uses the data collected by interrupts arrived during the previous time step, thus it uses data with 1 sample time of delay, exactly as it would happen with the second synchronization variant. The semantic of the `GetSensorData` function then becomes:

- copy buffered sensor data, acquired during the previous time step, into S;

ISBN: 978-1-61804-134-0

- trigger a new cycle of interrupt-driven acquisitions that will lead to a new S for the next sample time.

Figure 4 shows the execution flow and timing of the synchronization algorithm.

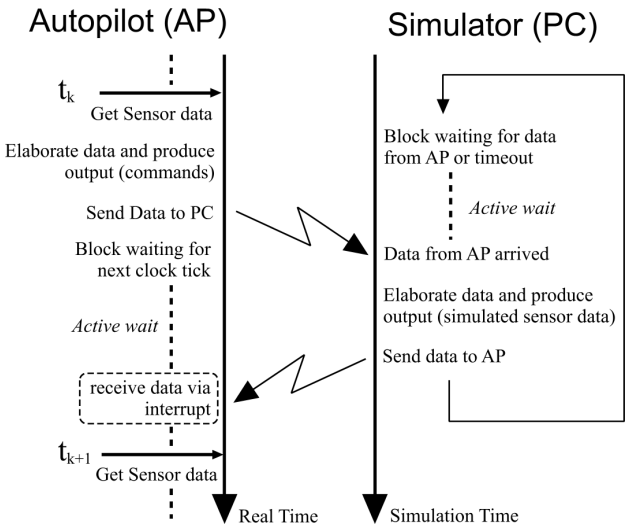

Figure 4. Execution flow and timing of the synchronization algorithm.

4. Man-In-the-Loop Simulation

Within the context of autopilot HIL simulation, a relevant issue is the man-machine interface that must allow an immersive experience for the pilot so that he can test effectively all the functionalities of the system like switching between different control modes, haptic support systems [5-6][10-13], delays due to communications etc. It is advisable then to let the pilot use the actual input device of the vehicle and to create a realistic synthetic environment of the operational area of interest. With respect to the input device, since this is usually directly connected to the autopilot, no particular actions are needed when performing HIL simulations: the operator will use the real input device, connected to the autopilot with the real communication channel, and not a virtual reproduction of it. As for the synthetic environment, we found experimental evidence of the importance of using sceneries that are representative of real places on earth as opposed too imaginary non-existent places: full immersion sensation and focus on the tasks is higher when the pilot is presented with a 3D scenery of a place he knows, and the same place is represented in 2D (map view) on the ground station screen. Thus we designed a complex communication architecture that uses Dynamic HTML tools and the UDP/IP protocol to interface GoogleEarth with our simulator in real-time. At the same time, in order to fulfill the exigency of having a clean 3D view (without clutter from textured background and building) with superimposed telemetry data, we build a second synthetic environment, to be used for engineering analysis, using DynaWORLDS[14]. Another important component of the simulation environment is the ground station; since we are performing an HIL simulation, there is no need to use a mock-up of it but the actual Ground Station hardware, software and communication channels may be used. Figure 5 shows a block diagram of the complete simulation environment.

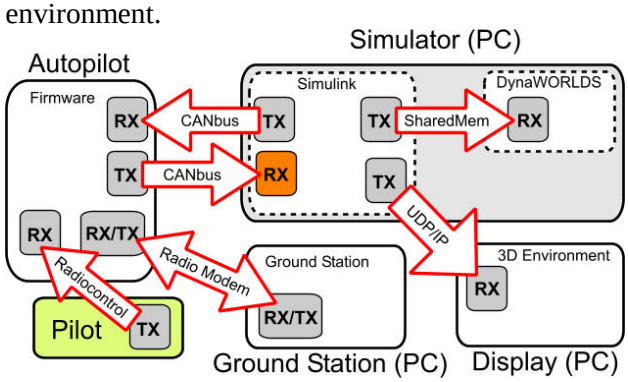

Figure 5. Block diagram of the complete HIL + MIL simulation system.

5. Application example

This section presents and application of the proposed HIL and MIL simulator; University of Pisa has a partnership with an Italian company[20] that designs, produces and operates multi-rotor vehicles (Fig. 6 shows one of their vehicles), to develop guidance, navigation and control systems for their vehicles, thus we developed a complete HIL+MIL simulator for one of their quadrotor vehicles. Figure 7 shows a sample view of the 3D synthetic environment. Figure 8 shows a snapshot of the ground station software designed at University of Pisa. The Autopilot and simulator exchanged 10 canbus packets at a rate of 100 Hz: 7 from simulator to autopilot for accelerometers, gyroscopes, magnetometers, baro-altimeter and GPS data, and 3 for autopilot commands and various debug data. Quadrotors are unstable vehicles, and rely completely on the autopilot to become “flyable”. Timing and delay in the data exchange simulations would seriously compromise vehicle stability since control loops are run by the actual autopilot. A stable simulated flight is already a good indication of synchronization; but, in order to validate the effectiveness of the synchronization system, and, at the same time, define a metric for distributed simulation fidelity and loss of

ISBN: 978-1-61804-134-0

synchronization, we identified two parameters: the actual duration of a simulation step (including the active wait) of the simulator, or task execution time (TET), and the number of packets NP already in the reception buffer of the PC when first starting the synchronization loop. It is clear that the ideal value for TET is the autopilot communication rate: 10 milliseconds; while the desired value for NP is 0, meaning that the simulator is fast enough, namely it elaborated the old data before new one arrived.

Figure 6. A Tecnodrone multi-rotor vehicle.

Figure 7. Synthetic Environment.

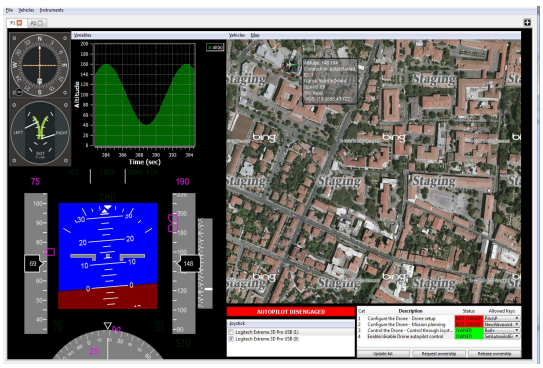

Figure 8. Ground Station screen snapshot.

Figure 9 and 10 show the result of a 200 seconds simulation run with high computational load on the simulator side: the simulator, ground station and 3D environment interface executed on the same PC. From the plots, it appears that TET values are always very close to 10 ms; a relevant jitter is also present and this is due mainly to the fact that the simulator runs on the Windows OS without any real-time extensions. The good synchronization results can be told also from the NP plot, that is 99.9% of the times equal to the desired value: 0.

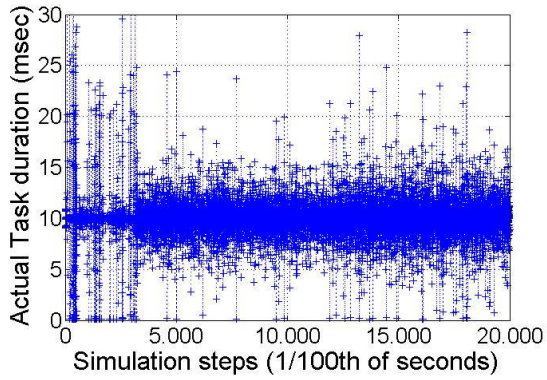

Figure 9. Sample TET values.

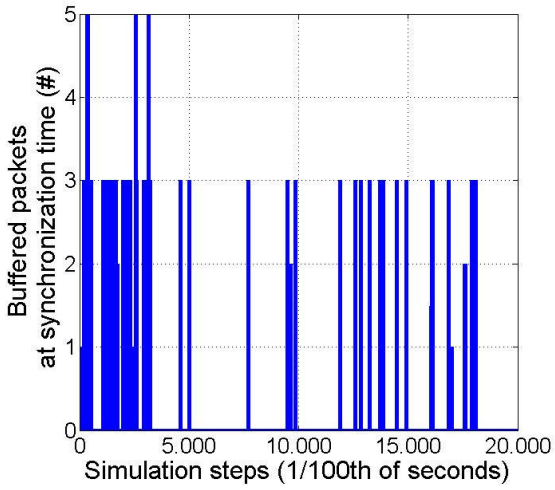

Figure 10. Sample NP values.

Those cases where NP>0 correspond, as expected, to the cases where TET>20 ms. It should be noted that NP goes back to 0 after those events (and TET drops to 1-2 ms), indicating that the system can quickly re-synchronize with the autopilot; these sporadic delays have thus only a minimal and temporary effect on the simulation fidelity.

6. Conclusions

The paper has presented a methodology for realizing a reliable HIL system that was used to implement a working simulation environment for the ICARO II

ISBN: 978-1-61804-134-0

autopilot. Particular care was given to realizing a simple and effective synchronization algorithm between the autopilot and the vehicle simulator; the method is general and may be employed in a large class of HIL problems. Finally a complete MIL and HIL system is presented with experimental results that confirm the validity of the proposed synchronization approach.

References:

[1] Passino, K. M., ”Stability Analysis of Swarms”, IEEE Transactions on Automatic Control, TR-AC 48, No. 4, April 2003.

[2] Pachter, M., D’Azzo, J. J., & Proud, A. W., Tight formation flight control. Journal of Guidance, Control, and Dynamics, 24(2), 246–254, 2001.

[3] Giulietti, F., Pollini, L., & Innocenti, M., Formation flight control: a behavioral approach. AIAA Guidance, Navigation and Control Conference, Montreal, Canada, 2001.

[4] F. Giulietti, L. Pollini, M. Innocenti, M. Napolitano, "Dynamic and control issues of formation flight," AEROSPACE SCIENCE AND TECHNOLOGY, vol. 9, no. 1, 2005.

[5] S.M.C. Alaimo, L. Pollini, J.P. Bresciani, H.H. Bülthoff, “Evaluation of Direct and Indirect Haptic Aiding in an Obstacle Avoidance Task for Tele-Operated Systems”, IFAC World Congress 2011, Milan Italy

[6] Lam, T.M., Boschloo, H.W., Mulder, M., Van Paassen, M.M., “Artificial force field for haptic feedback in UAV teleoperation”. IEEE Transactions on Systems, Man and Cybernetics, Part A. Vol. 39, Issue 6, 2009.

[7] E. Jones and S. Soatto, "Visual-Inertial Navigation, Mapping and Localization: A Scalable Real-Time Causal Approach," International Journal of Robotics Research, 2010.

[8] F. DiCorato, M.Innocenti, L. Pollini, Combined Vision–Inertial Navigation for Improved Robustness, Itzhack Y. Bar-Itzhack Memorial Symposium on Estimation, Navigation, and Spacecraft Control, Haifa, Israel, Oct. 2012

[9] F. Di Corato, M. Innocenti, G. Indiveri, and L. Pollini, "An Entropy-Like Approach to Vision Based Autonomous Navigation," IEEE International Conference on Robotics and Automation, Shanghai, China, 2011.

[10] Lam, T.M., Mulder, M., van Paassen, M.M., Mulder, J.A., van Der Helm, F.C.T., “Force-stiffness feedback in UAV tele-operation with time delay”. In AIAA Guidance, Navigation, and Control Conference, Chicago, Aug. 2009.

[11] Farkhatdinov, I., Jee-Hwan Ryu, Jinung An, “A preliminary experimental study on haptic teleoperation of mobile robot with variable force feedback gain” Haptics Symposium, Mar. 2010.

[12] Alaimo, S.M.C., Pollini, L.,Magazzù, A., Bresciani, J.P., Robuffo Giordano, P., Innocenti, M., Bülthoff, H.H., “Preliminary evaluation of a haptic aiding concept for remotely piloted vehicles”. In EuroHaptics 2010 Conference, July 2010.

[13] Alaimo, S.M.C., Pollini, L., Bresciani, J.P., Bülthoff, H.H., “A comparison of direct and indirect haptic aiding for remotely piloted vehicles”. 19th IEEE International Symposium in Robot and Human Interactive Communication, IEEE Ro-Man 2010

[14] Pollini, L. Innocenti, M., “A synthetic environment for dynamic systems control and distributed simulation”, IEEE Control Systems Magazine, Vol 20, Num. 2, pp 49-61, April 2000.

[15] Anderson, R.J., Spong, M.W., ”Bilateral control of teleoperators with time delay”, Proceedings of the 27th Conference on Decision and Control Austin, Texas, December 1988.

[16] Alaimo SMC, Pollini L, Bülthoff HH, “Admittance-based bilateral teleoperation with time delay for an Unmanned Aerial Vehicle involved in an obstacle avoidance task”, AIAA Modeling and Simulation Technologies Conference 2011 (MST-2011), American Institute of Aeronautics and Astronautics, Portland, Oregon, August, 2011.

[17] L. Pollini, M. Innocenti, F. Di Corato, M. Cellini, M. Franchi, R. Mati, V. Niccolai, "The ICARO Autopilot: A flexible Controller for small Unmanned Air Vehicles", 4th US - European Workshop and Flight Competition for Micro Aerial Vhicles (IMAV09), Pensacola, FL, June 2009.

[18] Di Puccio F., A. Musolino, R. Rizzo, and E. Tripodi, "A Self-Controlled Maglev System", Progress In Electromagnetics Research M, Vol. 26, 187-203, 2012.

[19] Di Puccio F., R. Bassani, E. Ciulli, A. Musolino, and R. Rizzo, "Permanent Magnet Bearings: Analysis of Plane and Axisymmetric V-Shaped Element Design" Progress In Electromagnetics Research M, Vol. 26, 2012.

[20] Tecnodrone, website www.tecnodrone.it

ISBN: 978-1-61804-134-0