Software architecture for controlling in real time aerial prototypes

A. Offermann, Jérôme de Miras, P. Castillo

Software architecture for controlling in real time aerial prototypes

A. Offermann[1], J. De Miras[1] and P. Castillo[1]

***Abstract*—Nowadays, there exists several platforms or experimental prototypes of aerial vehicles to control algorithm validation, most of them being specific for certain applications. The implementation of the control laws in commercial platforms is often restricted to certain criteria and pre-established conditions defined by the commercial system (or the builder). In this paper, a new generic software architecture operating under Linux, MATLAB®, ROS and Ardupilot is introduced for analyzing, evaluating and improving control algorithms for aerial robotics. The tedious programming code is not necessary because the code is generated by MATLAB Simulink®. The proposed architecture is composed of a ground station (GS) and a robot with an embedded system. This platform is validated with a new aerial prototype with tilting four rotors. Experimental results illustrate the good performance of the software architecture even when different maneuvers are demanded to the aerial prototype.**

I. Introduction

Multiple types of architectures for aerial vehicles have been developed in the robotics or control community, some of them are based on ROS - Robot Operating System - or Ardupilot. Several researchers working on aerial vehicles use MATLAB, especially Simulink, for simulating theirs control algorithms. Nevertheless they encounter a problem when validating theirs algorithms in real time, because an arduous programming process is necessary mainly for non-specialized users.

Fl-AIR, for example, is a framework developed at the Heudiasyc laboratory based on Linux [1]. This architecture was conceived for validating control algorithms in robots and contains a simulator that allows to safely test them in real time. It was applied with success in numerous aerial drone applications, nevertheless, the programming part to control implementation is not so easy and needs to be performed by experts in C++. Similarly, authors in [2] proposed a test-bed to implement and validate in real time high-level control strategies for aerial vehicles. The architecture is composed of a mini-helicopter equipped with various sensors and a Ground Station (GS) computer using Xtratum with Linux RT and PaRTiKle. In the architecture a radio control can be used to send signals from the PC to the aerial vehicle. Even if the platform performance is adequate for control algorithms, its good domain requires very good skills in programming.

ROS has been developed to obtain a versatile framework allowing simplified communication within a system but also between systems. It becomes an interesting solution for creating software architectures. This principle has been used in [3] where a quadcopter vehicle lands on an Unmanned Surface Vehicle (USV). The embedded system of the quadcopter is composed of an "Odroid" board and an ardupilot that controls the attitude of the vehicle. In the architecture, the high level control (navigation) is done using ROS nodes. Moreover, the authors use Mavlink to retrieve data from Ardupilot, though, the frequency is low. This fact does not allow the possibility to modify low level control laws through a ROS node. Another example of ROS application was presented in [4] where multiple UAVs (unmanned aerial vehicles) are flying in cooperation. The UAVs are Bitcraze Crazyflie 2.0 quadcopters. In this platform, a ROS package named *crazyflie_ros* is used to perform several tasks such as controlling a UAV with a joystick, perform hovering, etc. MATLAB's Robotics System Toolbox (MRST) provides solutions to easily generate ROS nodes by creating models in Simulink. These solutions have been applied, for example, on TurtleBots robots for path planning [5]. Similarly, in [6], the authors have simulated a hexacopter aerial vehicle using the V-REP software. In this architecture, the MRST was used to implement a linear controller (PID) and to communicate with the simulator, via ROS.

UAV prototypes are used as test-bed during last decades. Their versatility and dynamic are interesting for several applications in the robotic and the automatic control community, see [7]–[10]. Quadcopter aerial vehicles are the most popular UAV prototypes. Nevertheless, they are under-actuated and for some applications their use could be complicated. Other configurations of aerial vehicles with four rotors have been proposed in the literature, most of them are over or full-actuated systems, adding in the prototype actuators for increasing the control inputs, see for example [11]–[14].

In this work, we propose an innovative software architecture that has been conceived with MATLAB and ROS and is composed of an aerial robot with its embedded system and a ground station. In this software architecture, a simulator based on Simulink is designed for previously, validating in simulations the control algorithms. The platform exports the Simulink model for testing in real-time free flights avoiding all the tedious programming process (no extra code programming is necessary). The prototype is a quadcopter vehicle with tilting rotors conceived also for evaluating control algorithms.

This work was supported from PIA Robotex project

[1]A. Offermann, J. De Miras and P. Castillo are with Université de Technologie de Compiègne, Heudiasyc, Compiègne Cedex 60201, France {alexis.offermann@gmail.com, (jerome.demiras, castillo)@hds.utc.fr}

The outline of the paper is the following; the proposed software architecture and the simulator are presented in Section II. The mathematical description and control of the aerial vehicle is resumed in section III. The closed-loop system is then tested in simulations and implemented in an embedded system for flight tests. Main results illustrating the closed-loop behavior are given in section IV. At the end, some final discussions of this work are given in section V.

II. Software architecture

Our software architecture is composed of an intuitive simulator with an easy implementation and by a real-time platform for tests.

A. Simulator

The dynamic representation of a system is presented commonly in mathematical equations, nevertheless, its graphical representation is not obvious and its analysis is only done for expert users. Our idea is to place, instead the mathematical model (equations), a draw in 3D of the system where its movements are governed by its nonlinear dynamics. This 3D model (that can be done using CAD - Computer-Aided Design) is imported into MATLAB Simulink via Simscape™ toolbox compatible software[1]. Remark that the Simscape toolbox allows to add forces, torques and joints between different parts of the tested design. Therefore, it becomes possible to implement and to apply any control algorithm since a bridge between classic Simulink blocks and Simscape exists. In addition, in this simulator, it is possible to emulate tests into the system and include external perturbations and so on. The workflow for the simulator is presented in Figure 1.

B. Real-time test-bed

The previous Simulink program is converted into C++ code and exported into the embedded system of the robot. In addition, this architecture provides several possibilities to use Simulink toolboxes and eliminate the tedious programming part before uploading the code into the embedded system. This last advantage allows to make changes quickly and even to test different control algorithms much faster than with conventional programming. Moreover, the debugging part is also being reduced to zero.

[1]Autodesk®- inventor in our case

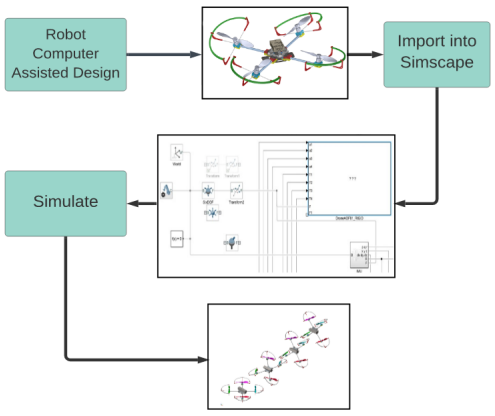

Figure 1. Workflow for the simulator

The robot's embedded system is composed in our case with a Raspberry Pi microcontroller and a Navio-2 shield from Emlid™. The Navio-2 has been chosen because it allows to have access to a several kind of sensors and actuators. A ground station (GS) containing the simulink program is used and has an interface for analyzing the states of the system and changing references values or gains. The data exchange between the GS and the robot is performed with ROS and is asynchronous, nevertheless the robot's performances is assured because it posses internal nodes running in real time.

Two different GS have been conceived for robot applications, they can be programmed either on MATLAB directly, or with a plugin named *rqt plot* from ROS. In our system two graphic interfaces (*rqt_plot* and *rqt*) are combined for data exchange and visualization allowing an useful and powerful system when experiments objectives change. *rqt_plot* is a data visualization tool for ROS that allows to analyze data evolution in real time. Some visualizing options are also available to increase visibility on data (auto scroll, scale change etc..). Using this plugin, graphs have more fluidity than when using Simulink. *rqt* is also an useful interface tool that allows us to publish data in specific communication channels in ROS called "topic". Moreover, gains parameters can be changed in real time by publishing them in specific topics.

Other advantage of this architecture is the possibility to record data from topics with *rosbag*. The sample rate of the nodes can be set in the Simulink model directly. As previously mentioned, the GS and the robot communicate via *wlan* totally asynchronously at the speed of the network. Nevertheless, this does not represent a control problem because the interface is only used to analyze the states or to tune parameters that will not command directly the vehicle. The manual control of the robot is achieved with an external radio-controller and a receiver connected on the Navio. The proposed software architecture is presented in Figure 2.

The principle of our architecture is to convert the code provided by the manufacturer of the shield to read and write the Navio2 IO[2] into ROS "nodes" (nodes being subsystems of ROS including one or several topics). The messages types will then be imported into MATLAB Simulink in order to be used directly as IO.

For this platform the following nodes have been developed

- *IMU node* for reading two IMU - Inertial Measurement Unit - sensors and include an observer for improving attitude;
- *GPS node* for connecting and accessing to GPS sensor.
- *Barometer node* to measure the altitude of the vehicle using a barometer sensor.

[2]Input Output

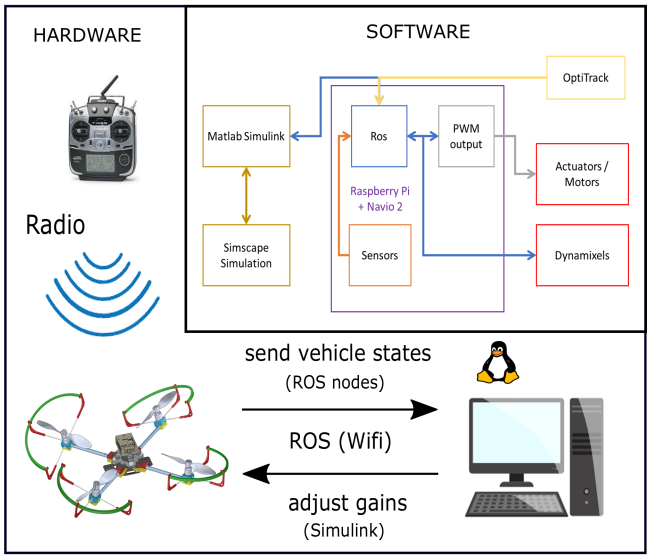

Figure 2. Architecture of the system

- *Radiocontroller node*, conceived for control manually the robot accessing completely classical radio-controllers.
- *PWM node*, this node is created for sending the computed control inputs to the motors.
- *LED node* created for information purpose. It can be used to display the status of the GPS for example.
- *A/D node*, this node converts from analog to digital voltage.

In addition, for this architecture, it is possible to add a wide panel of external components working directly with ROS. For example, our platform is equipped with Dynamixels™ servo-motors that are high efficiency actuators and allow to access several inner data as position, torque, current, temperature, etc, through a ROS node.

For this architecture, an *Optitrack node* - motion capture system - is also created. This node allows us to access to the Optitrack System for measuring the position and orientation of an object in 3D.

III. SYSTEM AND CONTROL

Our practical goal for the architecture is to control a complex aerial system composed of four motors with tilting arms in X configuration. This configuration has the property that the frame of the first arm (noted $\mathcal{M}_1$ in Figure 3) is located with an angle of $45°$ with respect to the body frame while the others arms are placed with an angle of $90°$ with respect to the previous ones.

A. Dynamic model

From Figure 3, notice that four frames are used in the dynamic analysis, they are denoted as inertial, body, arm and propeller frames. The dynamic model of this system is obtained using the Newton-Euler formalism as follows

$$\begin{aligned} \sum F_{\mathcal{F}_I} &= m\,\ddot{\xi} + \Omega \times m\,\dot{\xi} \\ \sum M &= J\,\dot{\Omega} + \Omega \times J\,\Omega \end{aligned} \tag{1}$$

where $\mathcal{F}_I$ denotes the inertial frame, m is the mass of the vehicle, ξ represents its vector position, Ω the angular velocity vector and J the moment of inertia of the vehicle.

Each motor produces a force in the propeller frame. For expressing the forces produced by the motors in the inertial frame, it is necessary to express them firstly into the arm frame using ${}^{a_i}R(\alpha_i)_{p_i}$. This rotation is due to the tilt of the propeller with an angle of α_i. Later, the force is expressed into the body frame using ${}^{B}R(\beta)_{a_i}$. β corresponds to the rotation between the body (where the Navio micro-controller is placed) and the i^{th} arm frame. And finally, the force is expressed into the inertial frame using ${}^{I}R_B$. ${}^{*^1}R_{*^2}$ denotes the rotation matrix between two frames. In our configuration, β_i is constant and related on the number of the arm. These angles are summarized in the $B_\times$ vector as follows

$$B_\times = \left[\tfrac{\pi}{4}\ ,\ \tfrac{3\pi}{4}\ ,\ \tfrac{-3\pi}{4}\ ,\ \tfrac{-\pi}{4}\right] \tag{2}$$

The real control inputs in our system are the thrust, T_i, and the tilt angle, α_i, for each motor, with $i : 1 : 4$, making it an over actuated system. After analysis of the system, a virtual input vector, noted by $\mathbf{u}_{1\times 6}$, is proposed. This vector is the multiplication result of the two rotations (${}^{a_i}R(\alpha_i)_{p_i}$ and ${}^{B}R(\beta)_{a_i}$, including the 8 actuators of the system) and is defined as follows

$$\mathbf{u} = \underbrace{\begin{bmatrix} p & 0 & p & 0 & -p & 0 & -p & 0 \\ -p & 0 & p & 0 & p & 0 & -p & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ pH & -p & -pH & p & -pH & -p & pH & p \\ -pH & p & -pH & -p & pH & p & pH & -p \\ 1 & H & 1 & -H & 1 & -H & 1 & H \end{bmatrix}}_{G} \begin{bmatrix} T_1\,sin(\alpha_1) \\ T_1\,cos(\alpha_1) \\ T_2\,sin(\alpha_2) \\ T_2\,cos(\alpha_2) \\ T_3\,sin(\alpha_3) \\ T_3\,cos(\alpha_3) \\ T_4\,sin(\alpha_4) \\ T_4\,cos(\alpha_4) \end{bmatrix} \tag{3}$$

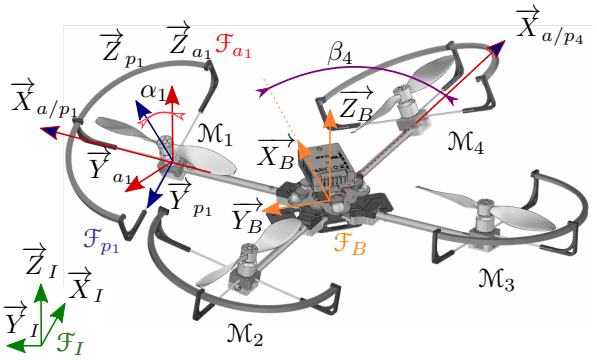

Figure 3. Frame definition on the tilting quadcopter. From figure $\mathcal{F}_B$ represents the body frame, $\mathcal{F}_{a_i}$ the frame of the i^{th} arm, $\mathcal{F}_{p_i}$ the frame of the i^{th} propeller, $\mathcal{M}_i$ the i^{th} motor, α_i the tilt angle of the i^{th} arm and β_i the angle between the x-body axis and the x axis of the i^{th} propeller.

with $p = \sqrt{2}/2$ and $H = {}^{k_\tau}/_{k_f}$, where k_τ and k_f denote the drag and force constants of the propellers.

Using (3) system (1) can be written as

$$\begin{bmatrix} m\,\ddot{\xi} \\ J\,\dot{\Omega} \end{bmatrix} = \begin{bmatrix} {}^{I}R_{B} \\ I_{[3\times 3]} \end{bmatrix} \mathbf{u} - \begin{bmatrix} m\,g\,\overrightarrow{z_I} \\ 0_{[3\times 1]} \end{bmatrix} + \begin{bmatrix} -\Omega \times m\dot{\xi} \\ C_{g-all} \end{bmatrix}_{[6\times 1]} \quad (4)$$

where g denotes the gravity, $\overrightarrow{z_I}$ defines a unitary vector in the z-axis, and C_{g-all} represents the torques due to Coriolis effects, containing the $\Omega \times J\Omega$ term. ${}^{I}R_{B}$ defines the classical rotation matrix between the body and the inertial frame.

B. Control algorithm

The control law is based on a feedback linearization, and is proposed as

$$\mathbf{u} = -\begin{bmatrix} {}^{I}R_{B}^{-1} \\ I_{[3\times 3]} \end{bmatrix} \left[\begin{bmatrix} m\,(k_1\dot{\xi} + k_2\tilde{\xi}) \\ J\,(k_3\Omega + k_4\tilde{\eta}) \end{bmatrix} - \begin{bmatrix} m\,g\,\overrightarrow{z_I} \\ 0_{[3\times 1]} \end{bmatrix} + \begin{bmatrix} -\Omega \times m\dot{\xi} \\ C_{g-all} \end{bmatrix} \right] \quad (5)$$

where $k_i > 0$, for $i = 1:4$, represents a constant gain chosen such that the dynamic of the system becomes stable. $\tilde{\xi}$ defines the position error $\tilde{\xi} = \xi - \xi_d$ and ξ_d denotes the desired position. Similarly, $\tilde{\eta} = \eta - \eta_d$ represents the orientation error. In our case, it is been computed with the error between the rotation matrix of the system and a desired rotation matrix, see [15], [16].

For computing the real control inputs (T_i and α_i) of the actuators in the prototype, the Moore-Penrose pseudo-inverse is used on matrix G. Therefore, from (3), it follows that

$$G^{\dagger} = \begin{bmatrix} \frac{1}{4p} & -\frac{1}{4p} & 0 & 0 & 0 & \frac{l^2}{4\,l\,(L^2+k^2)} \\ \frac{k}{4\,l\,p} & -\frac{k}{4\,l\,p} & \frac{1}{4} & -\frac{1}{4\,l\,p} & \frac{1}{4\,l\,p} & \frac{k^2}{4\,k\,(L^2+k^2)} \\ \frac{1}{4p} & \frac{1}{4p} & 0 & 0 & 0 & \frac{l^2}{4\,l\,(L^2+k^2)} \\ -\frac{k}{4\,l\,p} & -\frac{k}{4\,l\,p} & \frac{1}{4} & -\frac{1}{4\,l\,p} & -\frac{1}{4\,l\,p} & -\frac{k^2}{4\,k\,(L^2+k^2)} \\ -\frac{1}{4p} & \frac{1}{4p} & 0 & 0 & 0 & \frac{l^2}{4\,l\,(L^2+k^2)} \\ -\frac{k}{4\,l\,p} & \frac{k}{4\,l\,p} & \frac{1}{4} & \frac{1}{4\,l\,p} & -\frac{1}{4\,l\,p} & \frac{k^2}{4\,k\,(L^2+k^2)} \\ -\frac{1}{4p} & -\frac{1}{4p} & 0 & 0 & 0 & \frac{l^2}{4\,l\,(L^2+k^2)} \\ \frac{k}{4\,l\,p} & \frac{k}{4\,l\,p} & \frac{1}{4} & \frac{1}{4\,l\,p} & \frac{1}{4\,l\,p} & -\frac{k^2}{4\,k\,(L^2+k^2)} \end{bmatrix} \quad (6)$$

where l is the distance between motor and the gravity center of the system, $k = k_\tau/k_f$, and L is the length of an arm. Therefore

$$\begin{bmatrix} T_1\,sin(\alpha_1) \\ T_1\,cos(\alpha_1) \\ T_2\,sin(\alpha_2) \\ T_2\,cos(\alpha_2) \\ T_3\,sin(\alpha_3) \\ T_3\,cos(\alpha_3) \\ T_4\,sin(\alpha_4) \\ T_4\,cos(\alpha_4) \end{bmatrix} = G^{\dagger} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{bmatrix} \quad (7)$$

Finally, we obtain

$$T_i = \sqrt{(T_i\,\sin(\alpha_i))^2 + (T_i\,\cos(\alpha_i))^2} \quad (8)$$

$$\alpha_i = \tan^{-1}\left(\frac{T_i\,\sin(\alpha_i)}{T_i\,\cos(\alpha_i)}\right) \quad (9)$$

IV. Software architecture validation

As explained in previous section, the prototype is a quadcopter configuration with four tilting rotors as depicted in Figure 4. The tilting of the arms is produced by Dynamixels™ servomotors that are attached to carbon tubes and the structure of the vehicle. All linking pieces have been done in 3D printing. The hardware components of our prototype are summarized in Table I.

Figure 4. Drone with tilting propellers

Hardware component	Reference
Servomotors	Dynamixels ®AX-12A
Bottom, top plate and landing gear	F450 ®frame
Motors	T-Motors MT2212-13
Microcontroller	Navio + Raspberry pi®3

Table I
Drone's components.

The nonlinear system (4) and the controller (5) have been validated with the proposed software architecture, firstly using the simulator and later in flight tests. The scenario to validate is to keep at hover the aerial vehicle in a desired position while its body reaches a roll angle bigger than $10°$. Notice that this scenario is not possible for classical quadcopters because when the vehicle reaches this angle, it moves in a lateral direction.

A. Simulation results

In Figure 5 the performance of the quadcopter dynamics is illustrated. These pictures are obtained from the simulator where the control algorithm was applied to stabilize the drone at hover with roll angle bigger than $10°$ and $\dot{x}, \dot{y} = 0$m/s.

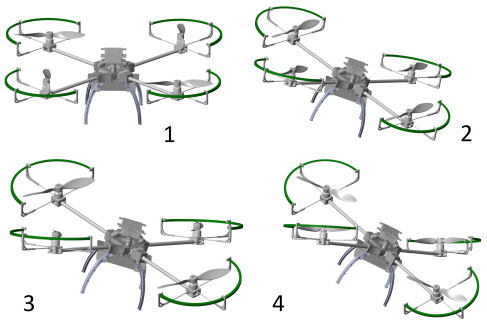

Figure 5. Drone simulation performance when applying the control algorithm into the system. The goal is to reach $\phi > 10°$.

B. Free flight experiments

For evaluating the proposed software architecture and the closed-loop system in real time, free flight tests were carried out. The experiments were carried out in three phases.

The first phase, named Q_1, is when the vehicle take-off and landing in semi-autonomous mode, i.e., the attitude

and altitude of the drone are managed with the feedback algorithm. The virtual inputs controlling the x and y axes are disabled. In this configuration, the yaw angle is controlled using the tilting arms and motor torques. The second phase, called Q_2, is related with the translational dynamics in the plane x, y. This phase is activated after the aerial robot reaches a desired altitude. For moving the vehicle in the plane, the virtual controllers in x and y are activated for controlling theirs tilt arms to keep the desired position in the plane. When the desired position in the plane x, y is reached, the last phase Q_3 starts. In this phase, the α angles reach a roll angle $> 10°$ with $\dot{x}, \dot{y} = 0$, as showing in Figure 6.

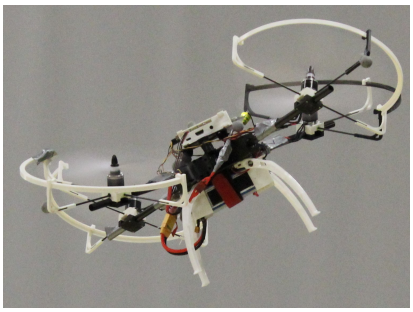

Figure 6. Picture of the drone while tilting

Control gains k_i used in the experiments are the following

$$k_1 = \begin{bmatrix} k_{1_a} & 0 & 0 \\ 0 & k_{1_b} & 0 \\ 0 & 0 & k_{1_c} \end{bmatrix}; \quad k_2 = \begin{bmatrix} k_{2_a} & 0 & 0 \\ 0 & k_{2_b} & 0 \\ 0 & 0 & k_{2_c} \end{bmatrix};$$

$$k_3 = \begin{bmatrix} k_{3_a} & 0 & 0 \\ 0 & k_{3_b} & 0 \\ 0 & 0 & k_{3_c} \end{bmatrix}; \quad k_4 = \begin{bmatrix} k_{4_a} & 0 & 0 \\ 0 & k_{4_b} & 0 \\ 0 & 0 & k_{4_c} \end{bmatrix};$$

Tables II and III summarizes the constant parameters used in real-time experiments. Same inertial moment for the three axis are considered.

Gain	Value	Gain	Value
k_{1_a}	0.561	k_{2_a}	0.867
k_{1_b}	0.561	k_{2_b}	1.02
k_{1_c}	0.85	k_{2_c}	0.7
k_{3_a}	90	k_{4_a}	26
k_{3_b}	50	k_{4_b}	22
k_{3_c}	80	k_{4_c}	55

Table II
GAINS VALUES USED IN THE CONTROLLER.

Param.	Description	Value	Unit
m	Mass of the vehicle	1960	g
l	Distance between motor and C.o.M.	0.25	m
J	Inertial moment of the system	0.006	Kg.m^2
k_f	Force constant for a given PWM	0.00305	$N.\mu s^{-1}$
k_τ	Torque constant for a given PWM	4.9715e-05	$N.m.\mu s^{-1}$

Table III
SYSTEM PARAMETERS

In Figure 7 the drone performance is depicted in 3D when applying the three phases.

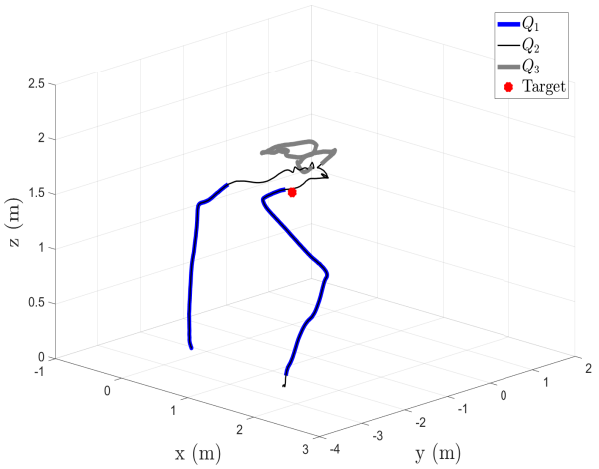

Figure 7. Aerial robot behavior during the free flight tests. The desired waypoint is in red colour. Phase Q_1 is represented in blue line, phase Q_2 is shown in black line and phase Q_3 is depicted in gray colour.

In Figure 8 the states response of the system obtained during experiments are illustrated. Notice that in phase Q_3 the states responses in x, y remain smaller and close to zero. Some errors can be observed due to the performance of the linear controller. Remember also that the idea in this paper is to evaluate the software architecture and not the control performance. Similarly, note in this figure that the roll angle reaches values bigger than $10°$. Note that the closed-loop system performance is acceptable. Some errors can be perceived due to the limitations of the linear controller. Observe also in Figure 8 a bias in the pitch and roll angles due to small asymmetry on the platform that is compensated with this bias in the angles (static error).

Remark from Figure 9 that the tilt angles performances, α_i, reach important values to achieve the control goal. Notice also here that the motor thrust, T_i, does not vary a lot, therefore, as expected for the drone configuration, the tilt arms vary to converge to the desired position.

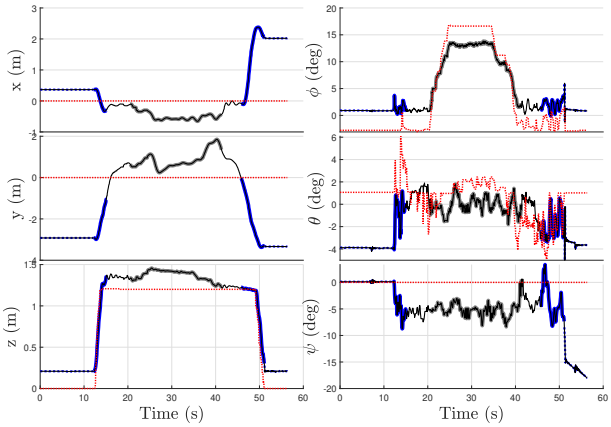

Figure 8. States performances of the drone in flight tests. Dotted red line is the desired reference for the vehicle. Blue line represents the states of the vehicle while being stopped. Hard blue line designs the states during flight mode Q_1. Q_2 flight performances are represented in black solid line. Eventually, Q_3 mode is depicted in gray.

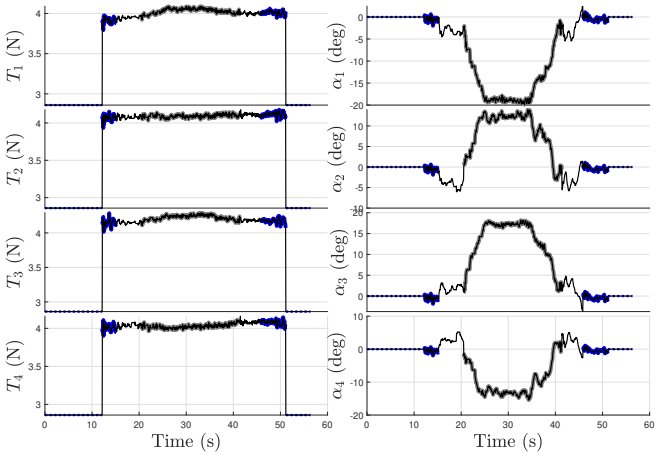

Figure 9. Actuators states of the drone during the experimental validation. Dotted blue line is used for the states while the drone is stopped. Hard blue line for the first flight mode Q_1 while black line illustrates the real-input responses in Q_2 mode flight. Q_3 flight mode is represented in gray line.

In Figure 10 the virtual control inputs performances, u_i, of the system are depicted. Notice that u_1 and u_2 are zero in mode Q_1 as stated before. Observe also in this figure, that u_3 remains quasi-constant during all the experiment signifying that the aerial vehicle is at hover. Similarly, notice that u_2 reaches higher values to compensate the tilt angles and avoid that the vehicles moves in the y axis. Eventually, spikes are due to the landing perturbation. A video of the flight tests can be seen at: https://youtu.be/kGbH6eN7poE

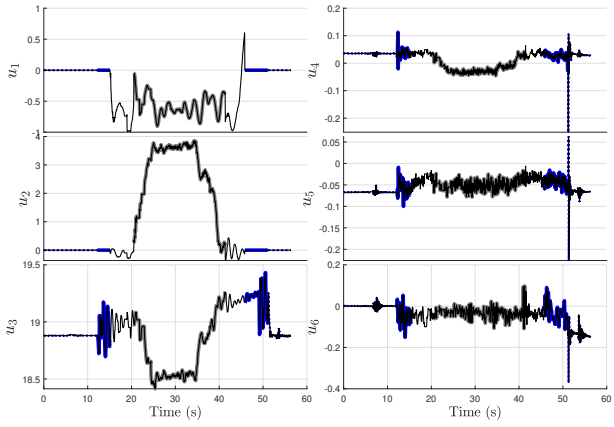

Figure 10. Virtual control inputs computed during the experimental tests. Dotted blue line represents the system stopped. Blue line denotes the input in Q_1 mode flight (take-off and landing) while black line represents the input in Q_2 mode flight for tracking the desired positions. Q_3 flight mode is depicted in gray line.

V. Conclusions

An innovative software architecture was presented in this paper. This architecture is based on MATLAB-ROS configuration allowing quick changes in the control algorithm and access to MATLAB features. In addition, with this architecture, the arduous programming tasks are not necessary. In the architecture, a simulator based on Matlab Simulink was introduced. This simulator allows to import CAD system designs that can be used for emulating the performance of different kind of controllers. The software architecture was validated in simulator and in real-time modes, where a quadcopter aerial vehicle with tilting rotors has been controlled. Other kinds of robots can be used with this architecture.

During free flight tests, a well performance of the architecture was observed allowing to change control gains or desired positions during the experiment. The behavior of the proposed scenario for validating the whole system was depicted in some graphs showing a well performance of the architecture and the closed-loop system.

References

[1] Sanahuja G. *Framework libre AIR*, July 3rd, 2013, Robotex, Toulouse, Online: https://robotextechdays.sciencesconf.org/, Online: https://devel.hds.utc.fr/software/flair/

[2] A. Berna, P. Castillo, G. Sanahuja et al., *Development of a test-bed to implement and validate real-time control strategies for aerial vehicles*, International Federation of Automatic Control (IFAC), Milano, Italy, August, 2011, pp 7660-7665.

[3] J. N. Weaver, D. Z. Frank, E. M. Schwartz et al., *UAV performing autonomous landing on usv utilizing the robot operating system*,ASME District F - Early Career Technical Conference (ECTC), Birmingham, Alabama, USA, November, 2013

[4] W. Hoenig, N. Ayanian,*Flying multiple UAVs using ROS*, Studies in Computational Intelligence, In: Koubaa A. (eds) Robot Operating System (ROS) vol 707, Springer, Cham, May, 2017.

[5] M. Galli, R. Barber, S. Garrido et al.,*Path planning using MATLAB-ROS integration applied to mobile robots*, IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC), Coimbra, April, 2017, pp. 98-103.

[6] A. Cantieri, A. Oliveira, M. Wehrmeister et al.,*Environment for the Dynamic Simulation of ROS-Based UAVs*, Environment for the Dynamic Simulation of ROS-Based UAVs, In: Koubaa A. (eds) Robot Operating System (ROS) vol 707, Springer, Cham, May, 2017.

[7] L. E. Muñoz, O. Santos, P. Castillo et al., *Energy-based nonlinear control for a quadrotor rotorcraft*, ICUAS conference, Washington, DC, USA, June, 2013, pp. 1177-1182.

[8] P. Castillo, R. Lozano, and A. E. Dzul, *Modelling and Control of Mini-Flying Machines*, Springer, 2005.

[9] H. Bouadi, M. Bouchoucha, M.Tadjine, *Sliding Mode Control based on Backstepping Approach for an UAV Type-Quadrotor*, International Journal of Applied Mathematics and Computer Sciences, January, 2007, pp. 12-17.

[10] H. Voos, *Nonlinear control of a quadrotor micro-UAV using feedback-linearization* IEEE International Conference on Mechatronics, Malaga, Spain, April, 2009, pp. 1-6.

[11] R. Kumar, A. Nemati, M. Kumar et al., *Tilting-Rotor Quadcopter for Aggressive Flight Maneuvers Using Differential Flatness Based Flight Controller*. ASME Dynamic Systems and Control Conference (DSCC), Tysons, Virginia, USA, October, 2017.

[12] M. Ryll, H. H. Bülthoff and P. R. Giordano, *Modeling and control of a quadrotor UAV with tilting propellers*. IEEE International Conference on Robotics and Automation (ICRA), Saint Paul, MN, USA, May, 2012, pp. 4606-4613.

[13] M. Ryll, H. H. Bülthoff and P. R. Giordano, *First Flight Tests for a Quadrotor UAV with Tilting Propellers* IEEE International Conference on Robotics and Automation (ICRA), Karlsruhe, Germany, May 6-10, 2013, pp. 295-302.

[14] M. Ryll, H. H. Bülthoff and P. R. Giordano, *A Novel Overactuated Quadrotor UAV: Modeling, Control and Experimental Validation* IEEE Transactions on Control Systems Technology, Institute of Electrical and Electronics Engineers, March, 2015, vol. 23, no. 2, pp. 540-556.

[15] M. Ryll, G. Muscio, F. Pierri et al., *6D Physical Interaction with a Fully Actuated Aerial Robot* IEEE International Conference on Robotics and Automation (ICRA), Singapore, May 29 - June 3, 2017, pp. 5190-5195.

[16] R. Mahony, T. Hamel, J-M. Pflimlin., *Nonlinear Complementary Filters on the SpecialOrthogonal Group* IEEE Transactions on Automatic Control, Institute of Electrical and ElectronicsEngineers, 2008, 53 (5), pp.1203-1217.