Software-in-the-loop simulation for improving flight control system design: a quadrotor case study

Giuseppe Silano, Pasquale Oppido and Luigi Iannelli

Abstract— **Simulation is a standard approach used for designing complex systems like the flight controller in multi-rotor vehicles. In this paper we illustrate how the software-in-the-loop (SIL) methodology allows to detect and manage instabilities of a quadrotor control system that otherwise might not arise when considering only Matlab/Simulink simulations.**

The use of the SIL technique allows to understand the behavior of the flight control system by comparing and evaluating different scenarios, with a details level quite close to reality. At the same time, it is possible to discover issues that a model-in-the-loop (MIL) simulation does not necessarily detect, even if carried out through a multi-physics co-simulation approach.

The paper aims to give the reader a practical and concrete evidence of such considerations through the case study of a micro quadrotor.

I. Introduction

Aerial robotics is a fast-growing field of robotics and in particular multi-rotor aircraft, like quadrotors, are rapidly increasing in popularity also out of the scientific community. Thanks to their hovering and vertical take-off and landing (VTOL) capabilities and the capacity to perform tasks with complete autonomy, they are now a standard platform for numerous military and civilian applications, e.g., inspections of power lines, bridges and pipelines [1], soil and field analysis [2], crop monitoring [3].

However, designing autopilots for UAVs (Unmanned Aerial Vehicles) is a challenging task, which involves multiple interconnected aspects. Numerous researchers are currently addressing the problem of designing autonomous guidance [4] and navigation systems [5] as well as control systems for multi-rotor vehicles [6]. Therefore, having tools able to show what it happens when some new applications are going to be developed in unknown or critical situations is more and more important.

Simulation is one of such helpful tools, widely used in robotics, enabling not only to verify the components integration and to evaluate their behavior under different circumstances but also to simplify the development and validation processes. Furthermore, simulation is cheaper than experiments with real robots, in terms of time and human resources: it makes possible simulating multiple robots when the hardware may not be available and getting a better understanding of implemented methods under various conditions.

Different solutions, typically based on dedicated robotic simulators such as Gazebo [7], V-REP [8], AirSim [9] are available to this purpose. They employ recent advances in computation and computer graphics in order to simulate physical phenomena (gravity, magnetism, atmospheric conditions) and perception (e.g., providing sensor models) in such a way that the environment realistically reflects the actual world. Definitely, it comes out that software platforms able to test algorithms for UAVs moving in a simulated 3D environment are becoming an indispensable part of the design phase.

The aim of this paper is to show the role and the effectiveness of robotics simulators in flight control system design for multi-rotor aircraft (a quadrotor, in our case). In particular it will be explained, by using a rather complex example, how the software-in-the-loop (SIL) simulation allows to detect and to manage instabilities that otherwise might not arise when considering only Matlab/Simulink simulations. On the other hand such instabilities may not be just related to the complexity, accuracy or detailed modeling of the simulated plant, but rather they may appear due to peculiar features of the final realization and, in particular, the software that will implement the control strategy. Indeed, aspects like synchronization, overflow, tasks communication, are all managed by libraries or tools available during the control design phase and yet they are specific of the final code implementation [10]. From such perspective, SIL simulation has to be considered a valuable tool for discovering, in an earlier phase of the usual V-model process, those issues that a model-in-the-loop (MIL) simulation does not necessarily detect. At the same time, a SIL simulation, obtained by using realistic and detailed simulators, gives the opportunity of validating in an easy way the effects of modifying the control strategy for complex missions. That represents quite often the easiest way to tune the flight control system and to check its validity.

Although advantages of such methodology are reasonable for the scientific community from a very general viewpoint, in the authors' opinion an illustrative case study can be of interest in particular if declined to the specific application, and when the code is provided as open-source [11] for scientific and educational activities.

II. Motivating case study

The case study here considered is the stabilizing controller discussed in [12] that in our case has been designed by considering the Parrot Bebop 2 quadrotor (see Fig. 1).

A detailed aircraft model was used in a twofold way: firstly, for tuning the controller gains (obtained as the solution of an optimization problem), and then for validating the

Giuseppe Silano, Pasquale Oppido and Luigi Iannelli are with the Department of Engineering, University of Sannio in Benevento, Piazza Roma, 21 - 82100 Benevento, Italy, emails: {giuseppe.silano, pasquale.oppido, luigi.iannelli}@unisannio.it

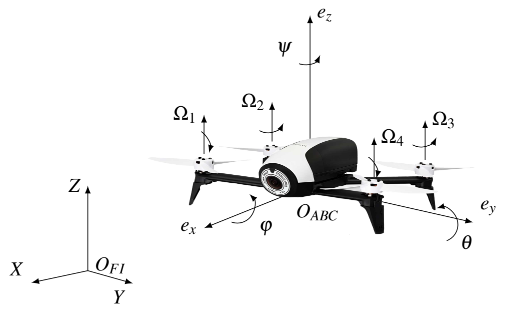

Fig. 1. Bebop in the body-frame (O_{ABC}) and the fixed-frame (O_{FI}) reference systems.

flight control system by comparing MIL simulation results (the flight control system implemented as a Simulink model) with those obtained through SIL simulation (the flight control system implemented as an executable object code) [10].

A. Dynamical model

The design of a high performance attitude and position controller requires often an accurate model of the system. We here recall the commonly used dynamical model of a quadrotor [13] and, by following usual approaches, we introduce two orthonormal frames: the fixed-frame O_{FI} (where FI stands for Fixed Inertial), also called inertial (or reference) frame, and the body-frame O_{ABC} (where ABC stands for Aircraft Body Center) that is fixed in the aircraft center of mass and is oriented according to the vehicle orientation (attitude), see Fig. 1. The translational dynamic equations of the vehicle can be expressed in the inertial frame as follows:

$$m\ddot{\xi} = -mgE_z + u_T R(\varphi,\theta,\psi)E_z, \tag{1}$$

where g denotes the gravity acceleration, m the mass, u_T the total thrust produced by the rotors, $\xi = \begin{pmatrix} x & y & z \end{pmatrix}^\top$ the drone position expressed in the inertial frame, $E_z = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}^\top$ is the unit vector along the Z-axis, while $R(\varphi,\theta,\psi)$ is the rotation matrix from the body to the inertial frame and it depends on the attitude $\eta = \begin{pmatrix} \varphi & \theta & \psi \end{pmatrix}^\top$ (i.e., Euler angles roll, pitch and yaw, respectively) that describes the body-frame orientation according to the ZYX convention [13]. Conversely, the rotational dynamics can be expressed as

$$I\dot{\omega}_B = -\omega_B \times I\omega_B + \tau, \tag{2}$$

where '$\times$' denotes the vector product, $\omega_B = \begin{pmatrix} \omega_x & \omega_y & \omega_z \end{pmatrix}^\top$ is the vector of the angular velocity expressed in the body-frame, $I = \text{diag}(I_x, I_y, I_z)$ is the inertia matrix of the vehicle w.r.t. its principal axis, and $\tau = \begin{pmatrix} u_\varphi & u_\theta & u_\psi \end{pmatrix}^\top$ is the control torque vector obtained by actuating the rotors speeds according to the rotors configuration and the vehicle shape.

At low speeds and around the hovering state the simplified dynamic model consists of six second order differential equations obtained from balancing forces and momenta acting on the drone, where $c_\bullet$ and $s_\bullet$ denote $\cos(\bullet)$ and $\sin(\bullet)$ functions, respectively:

TABLE I
PARROT BEBOP 2 PARAMETERS' VALUES.

	Sym.	Value	Unit
Total quadrotor mass	m	0.5	kg
Body inertia along x-axis	I_x	0.00389	$\text{kg}\,\text{m}^2$
Body inertia along y-axis	I_y	0.00389	$\text{kg}\,\text{m}^2$
Body inertia along z-axis	I_z	0.0078	$\text{kg}\,\text{m}^2$
Time motor constant	T_m	0.0125	s
Thrust motor constant	b_f	$8.548\,58 \times 10^{-6}$	kg m
Motor moment constant	b_m	0.016	m
Distance of propellers	l	0.12905	m
Maximum propellers speed	$\Omega_{\max}$	1475	$\text{rad}\,\text{s}^{-1}$

$$I_x\ddot{\varphi} = \dot{\theta}\dot{\psi}(I_y - I_z) + u_\varphi \tag{3a}$$
$$I_y\ddot{\theta} = \dot{\varphi}\dot{\psi}(I_z - I_x) + u_\theta \tag{3b}$$
$$I_z\ddot{\psi} = \dot{\theta}\dot{\varphi}(I_x - I_y) + u_\psi, \tag{3c}$$
$$m\ddot{x} = u_T\left(c_\varphi s_\theta c_\psi + s_\varphi s_\psi\right) \tag{4a}$$
$$m\ddot{y} = u_T\left(c_\varphi s_\theta s_\psi - s_\varphi c_\psi\right) \tag{4b}$$
$$m\ddot{z} = u_T c_\theta c_\varphi - mg. \tag{4c}$$

Equations (3)–(4) represent the nominal model used for designing the control law in [12] and here described in Sect. II-B. However a more detailed model should be considered when simulation has to be employed as part of the control design process. Thus we introduced further details for catching more realistic behaviors writing the model inputs as

$$u_T = b_f\left(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2\right), \tag{5}$$

and

$$\tau = \begin{pmatrix} u_\varphi \\ u_\theta \\ u_\psi \end{pmatrix} = \frac{b_f}{\sqrt{2}} \begin{pmatrix} l\left(-\Omega_1^2 - \Omega_2^2 + \Omega_3^2 + \Omega_4^2\right) \\ l\left(-\Omega_1^2 + \Omega_2^2 + \Omega_3^2 - \Omega_4^2\right) \\ \sqrt{2}b_m\left(-\Omega_1^2 + \Omega_2^2 - \Omega_3^2 + \Omega_4^2\right) \end{pmatrix}, \tag{6}$$

where Ω_i, $i \in \{1,2,3,4\}$, are the actual rotors angular velocities expressed in $\text{rad}\,\text{s}^{-1}$, l is the distance from the propellers to the center of mass, while b_f and b_m are the thrust and drag factors, respectively. Further details can be found in [4] or [13].

We also modeled the actuators dynamics

$$T_m\dot{\Omega}_i + \Omega_i = \Omega_i^{\text{ref}}, \tag{7}$$

where T_m is the motor time constant (assumed the same for all motors) and Ω_i^{ref} is the commanded motor velocity, considering physical constraints acting on the propellers velocities modeled in the simulator. Table I reports the Parrot Bebop 2 drone parameters' values.

B. Flight control system

With the aim of illustrating a control design methodology exploiting the SIL simulation, we started by considering a common cascaded control architecture. Figure 2 describes the overall system. The position controller (the outer loop controller) uses the measured drone position ξ_d to compute the thrust (u_T) and attitude (φ_r and θ_r) that should have the drone in order to reach the desired position ξ_r with the desired heading orientation (yaw angle) ψ_r. The attitude

TABLE II

PHYSICAL CONSTRAINTS OF THE PARROT BEBOP 2 QUADROTOR.

	Sym.	Unit	Output limit
Roll angle	φ_r	deg	$[-30,\ 30]$
Pitch angle	θ_r	deg	$[-30,\ 30]$
Thrust command	u_T	N	$[0,\ 18.6]$

controller (the inner loop controller) uses the measured drone attitude η_d to compute the model inputs u_φ, u_θ and u_ψ that should be actuated in order to achieve the desired attitude (φ_r, θ_r, ψ_r). The control mixer inverts eqs. (5) and (6) thus obtaining the commanded motor velocities Ω_i^{ref} in eq. (7). We here consider the control strategy proposed in [12] (authors refer to it as nonlinear internal model control) that we recall below for the sake of completeness. The drone position along the X-, Y- and Z-axis is controlled through the virtual inputs u_x, u_y and u_z

$$u_x \triangleq \left(c_{\psi_d} s_{\theta_d} c_{\varphi_d} + s_{\psi_d} s_{\varphi_d}\right) u_T \tag{8a}$$

$$u_y \triangleq \left(s_{\psi_d} s_{\theta_d} c_{\varphi_d} - c_{\psi_d} s_{\varphi_d}\right) u_T \tag{8b}$$

$$u_z \triangleq u_T c_{\theta_d} c_{\varphi_d} - mg, \tag{8c}$$

(see eqs. (4)) that are given by the control laws

$$u_q = \left(\frac{\alpha_q}{\mu_q}\dot{e}_q - \frac{\beta_q}{\mu_q^2} e_q\right) m, \quad q \in \{x, y, z\}, \tag{9}$$

where $\mu_q > 0$, $\beta_q < 0$ and $\alpha_q = 1 - \beta_q > 0$ are controller parameters to be appropriately chosen and e_q are the position and orientation components of the tracking errors.

The outer controller uses u_x and u_y to compute the desired roll and pitch angles

$$\varphi_r = \sin^{-1}\left(\frac{u_x s_{\psi_r} - u_y c_{\psi_r}}{u_T}\right), \quad \theta_r = \sin^{-1}\left(\frac{u_x c_{\psi_r} + u_y s_{\psi_r}}{u_T c_{\varphi_r}}\right),$$

and the total thrust u_T given by $u_T = \sqrt{u_x^2 + u_y^2 + (u_z + mg)^2}$. Conversely, the inner controller determines the inputs u_φ, u_θ and u_ψ to regulate the drone attitude according to the following control laws:

$$u_\varphi = I_x \left(\frac{\alpha_\varphi}{\mu_\varphi}\dot{e}_\varphi - \frac{\beta_\varphi}{\mu_\varphi^2} e_\varphi - \frac{e_\theta e_\psi}{\mu_\theta \mu_\psi}\left(\frac{I_y - I_z}{I_x}\right)\right) \tag{10a}$$

$$u_\theta = I_y \left(\frac{\alpha_\theta}{\mu_\theta}\dot{e}_\theta - \frac{\beta_\theta}{\mu_\theta^2} e_\theta - \frac{e_\varphi e_\psi}{\mu_\varphi \mu_\psi}\left(\frac{I_z - I_x}{I_y}\right)\right) \tag{10b}$$

$$u_\psi = I_z \left(\frac{\alpha_\psi}{\mu_\psi}\dot{e}_\psi - \frac{\beta_\psi}{\mu_\psi^2} e_\psi - \frac{e_\varphi e_\theta}{\mu_\varphi \mu_\theta}\left(\frac{I_x - I_y}{I_z}\right)\right). \tag{10c}$$

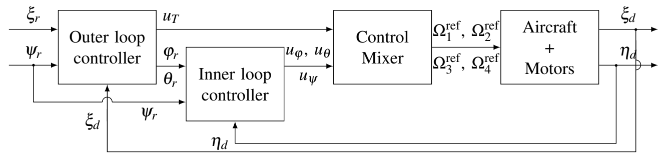

Fig. 2. The control scheme. Subscript d indicates the drone variables and r indicates references to controllers.

TABLE III

CONTROLLER PARAMETER VALUES.

Sym.	Value	Sym.	Value
μ_x	0.08	β_x	-26.4259
μ_y	1	β_y	-26.3627
μ_z	1	β_z	-27.2277
μ_φ	0.09	β_φ	-14.8134
μ_θ	0.26	β_θ	-16.1561
μ_ψ	0.05	β_ψ	-14.3431

Outputs of the outer loop controller are subject to constraints (physical or determined by the specific control strategy) that for the considered application are reported in Table II.

Note that the control strategy defined by (9)–(10) guarantees some local stability properties when applied to the system (3)–(4) (see [12, Theorem 1]). Thus it is of interest to understand if such stability properties are preserved when applying the described control architecture to the actual hardware platform that, on the other hand, exhibits more complex behaviors due to some neglected aspects like physical constraints, the digital controller implementation, the actuators and sensors dynamics. To this aim, rather than implementing the control laws on the hardware platform, we can proceed through a SIL simulation approach.

However, first it is required to choose controller parameters' values and that can be achieved through a MIL simulation approach. The Matlab/Simulink platform was used to minimize in a numerical way (through the `fmincon` function of the MathWorks Optimization Toolbox$^{\text{TM}}$) the integral of the squared error (ISE)

$$\text{ISE}(\mu_k, \beta_k) \triangleq \frac{1}{t_f - t_i}\int_{t_i}^{t_f} \left(\|e_\eta(t)\|^2 + \|e_\xi(t)\|^2\right) \mathrm{d}t, \tag{11}$$

w.r.t. the control parameters μ_k, β_k, $k \in \{\eta, \xi\}$ with $\eta = \{\varphi,\ \theta,\ \psi\}$ and $\xi = \{x,\ y,\ z\}$. Differently from [12], here a more accurate model takes into account also motor dynamics, saturation constraints and controller discretization, by considering also β_k gains as further decision variables. The optimal values of the gains are reported in Table III.

III. NUMERICAL EXPERIMENTS

A. Controller implementation

When moving from the control design based on the nominal model (Sects. II-A and II-B) to the actual implementation, several issues should be addressed. First of all we should consider that the proposed control architecture is based on control loops that, at least for the position controller, are nothing but standard PD (proportional-derivative) controllers. That are standard solution in the literature in quadorotor controllers design [14]. For such class of controllers, a classical way of dealing with the time derivative of the control error is to differentiate only the output signal that, in our case, is assumed to be directly available as a state variable of the system (of course, in a possible hardware deployment, a state estimator has to be introduced). A second aspect to consider towards the real implementation is the controller discretization. As a common rule in cascade structures, the inner loop needs to be regulated at a rate faster than the outer

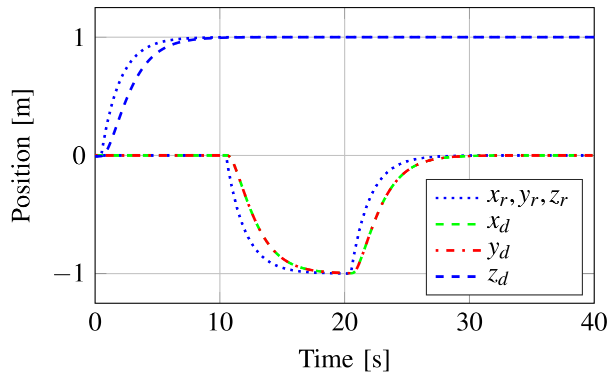

Fig. 3. Drone position: numerical experiments in Matlab/Simulink.

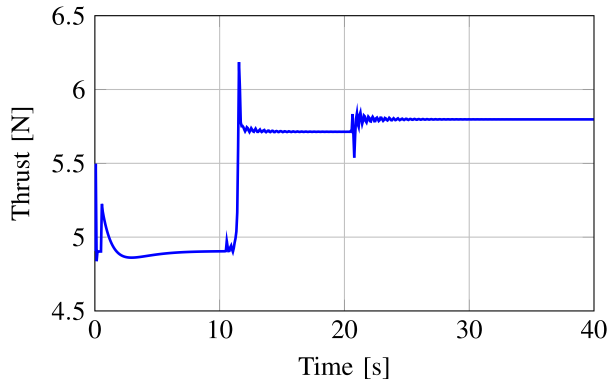

Fig. 4. The total thrust u_T requested by the controller: numerical experiments in Matlab/Simulink.

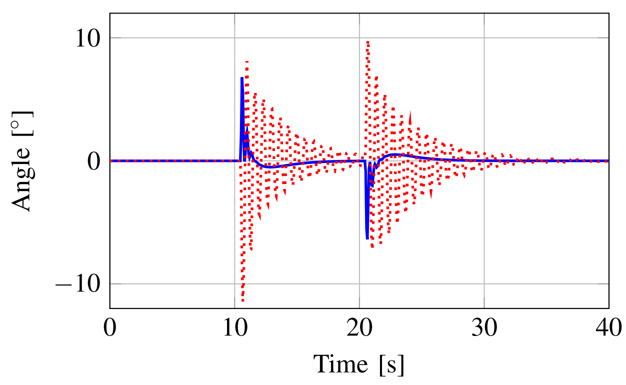

Fig. 5. Reference angles φ_r (blue, solid) and θ_r (red, dashed) computed by the outer loop controller: numerical experiments in Matlab/Simulink.

loop. In our case, the attitude controller runs at 200 Hz while the position controller runs at 100 Hz. Finally, the controller has to clip desired rotor velocities so that $0 \leq \Omega_i^{\text{ref}} \leq \Omega_{\max}$.

B. SIL simulation

The SIL simulation has been carried out by the ROS (Robot Operating System) middleware that gives the possibility to write the controller implementation in `C/C++` language facilitating rapid prototyping and the reuse of software. Regarding the robotic simulator, we used Gazebo and the ROS packages RotorS [15], [16] that both interface to ROS allowing to simulate controlled multi-rotor vehicles by taking into account quite detailed physical models that consider secondary effects like rolling moment and/or disturbances due to the drag of the rotor blades, etc.. Shortly Gazebo simulates behaviors that are more realistic than Simulink that, instead, has been used to simulate only the model described in Sect. II-A.

The SIL simulation provides the flight control system of Sect. II-B, together with the further modifications described in Sect. III-A, written in `C++` code and implemented as a ROS node running on the same ROS network. Gazebo and the controller node operate at the same frequency of 1 kHz (integration step time of 1 ms).

C. Results

MIL simulations obtained in Matlab/Simulink show an acceptable behavior of the controlled quadrotor.Figure 3 illustrates how the system performs and how the vehicle is capable of tracking the reference trajectory along the Z-axis as like as in the XY plane. In Fig. 4 the total thrust u_T computed by the controller is reported and it is evident that it satisfies constraints of Table II. In addition, also desired roll φ_r and pitch θ_r satisfy those constraints, so as depicted in Fig. 5. It appears that the flight control system (specifically, the outer loop) requests angles profiles that show a strong oscillatory behavior (in particular the pitch angle). That is probably due to the high value gains obtained from the tuning process. On the other hand those gains guarantee good tracking performances and, at the same time, pitch oscillations are damped and still within the admissible ranges. Moving to the SIL simulations, they exhibit strong instabilities and the quadrotor crashes to the soil (the video is available at `http://tiny.cc/aags9y`), thus showing how the software implementation can affect the system performances.

It is not easy to understand what is going wrong with SIL simulations even because the quadrotor model is different (and more realistic) w.r.t. the Simulink model. By looking at first tenths of second of simulations it comes out that the u_z control signal assumes negative values much below $-mg = -4.9$ N, see Fig. 6. That is an important issue since $u_z < -mg \Rightarrow u_z + mg < 0$ but, from (8c), that means $u_z + mg = u_T c_{\theta_d} c_{\varphi_d} < 0$. Of course, due to physical constraints on u_T and roll and pitch angles, it is not possible to apply a thrust that gives the desired u_z and such specific situation is well known in literature to bring the system to instability [17]. It happened that, differently from the Matlab/Simulink platform, the SIL approach allowed to understand that critical conditions arise due to physical constraints that have not been considered in the flight control design. It is important to highlight that the implementation in Matlab/Simulink is not oversimplified but it happens that the closed loop system is such that neglected dynamics and phenomena (both of the model and the controller) determine a critical behavior.

To ensure the operation of the flight control system under constraints of Table II, the approach proposed in [17] has been applied by imposing the following inequalities

$$|u_x| \leq mU_x, \qquad |u_y| \leq mU_y, \qquad |u_z| \leq mU_z, \tag{12}$$

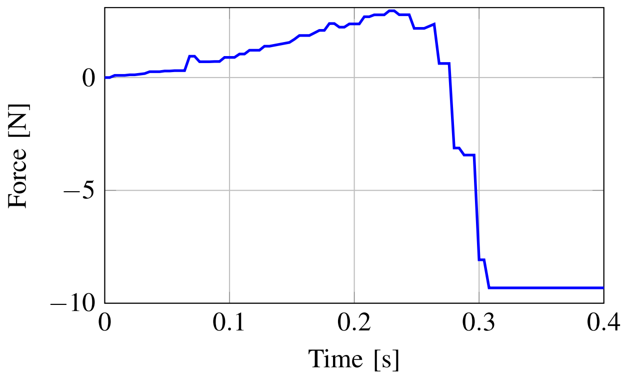

Fig. 6. Signal u_z: numerical experiments in Gazebo.

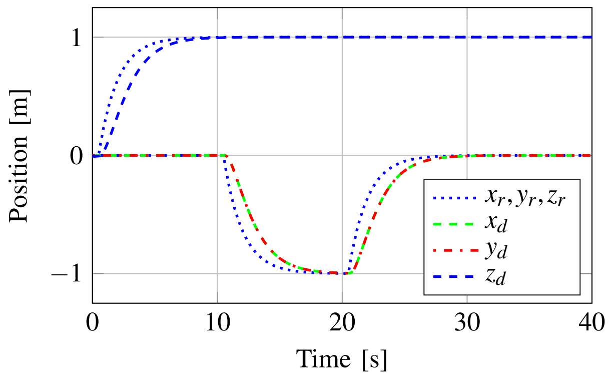

Fig. 7. Drone position obtained by applying the nested control laws (14): numerical experiments in Matlab/Simulink.

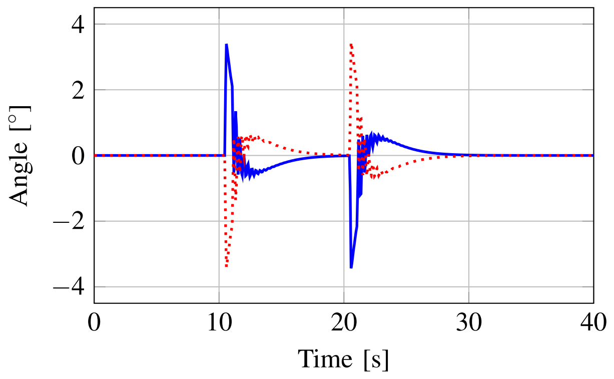

Fig. 8. Reference angles φ_r (blue, solid) and θ_r (red, dashed) obtained by applying the nested control laws (14): numerical experiments in Matlab/Simulink.

where U_x, U_y and U_z are suitable constants (see Table IV) so that

$$U_z < g \tag{13a}$$

$$U_x^2 + U_y^2 \leq (-U_z + g)^2 \tan^2 \varepsilon_c \tag{13b}$$

$$\sqrt{U_x^2 + U_y^2 + (U_z + g)^2} \leq u_{T_{\max}}, \tag{13c}$$

where ε_c is the maximum absolute value of the reference angles that has been chosen equal to 18° for getting an adequate "safety margin". According to [17, eq. (40)], now the position controller is implemented by replacing control laws in eqs. (9) with the following nested control law

$$u_q = U_q \sigma \left(\frac{K_{q_1}}{U_q} \dot{e}_q + \frac{1}{2} \sigma \left(\frac{K_{q_2}}{U_q} \dot{e}_q + \frac{K_{q_1} K_{q_2}}{U_q} e_q \right) \right), \tag{14}$$

where $K_{q_1}, K_{q_2} \geq 0$ are controller parameters and σ is the saturation function $\sigma(s) = \mathrm{sgn}(s) \min(|s|, 1)$. It is not difficulty to show that the following choice of gains

$$K_{q_1} = \frac{1}{\mu_q}, \qquad K_{q_2} = -\frac{2\beta_q}{\mu_q}, \tag{15}$$

corresponds to the same PD gains of eqs. (9) when σ functions in (14) do not saturate.

Figures 7 and 8 show Simulink results that consider the new nested control laws in the outer loop. Oscillations amplitude on the reference angles decreased but the quadrotor still tracks the reference trajectory (although a higher position error is now present on the altitude). When running the SIL simulation with the new nested control outer loop, instabilities do not appear anymore, but the control performances in tracking the same trajectory is very poor: in particular the quadrotor shows an unpredictable behavior regarding the altitude (the video is available at `http://tiny.cc/4cgs9y`) that might be due to the coupling effects with xy dynamics. Indeed, also attitude angles change even though XY references have not been modified yet (see Fig. 9). Such coupling effects are probably caused by the asymmetric positioning of the rotors w.r.t. the principal axis (Gazebo, differently from Simulink, considers such further non ideal feature, as well). Therefore, in order to reduce the attitude controller action that might be too high under such perturbations, the corresponding gains have been retuned by modifying only the inner loop gains that now minimize the cost function

$$J(\mu_\eta, \beta_\eta) = \frac{1}{t_f - t_i} \int_{t_i}^{t_f} \|e_\xi(t)\|^2 + \alpha \|u_\eta(t)\|^2 \mathrm{d}t \tag{16}$$

where α is a weighting coefficient. Although Simulink simulations show good performances, SIL simulations still are not close to what MIL simulations predict. In particular, the presence of spikes on the drone altitude (see Fig. 10) are not acceptable for a real application (as clearly shown by the video at `http://tiny.cc/eggs9y`).

Further investigations led us to understand that the naive clipping of rotor velocities might give rise to such strange behavior so as explained in [18], [19]. Once again, the use of SIL methodologies allowed to observe instabilities and issues that did not appear in classical Matlab/Simulink simulations. A smarter computation of the commanded rotor velocities has been applied for compensating such subtle effects. However, rather than using elaborated and complex algorithms,

TABLE IV
THE VIRTUAL INPUTS BOUND LIMITS.

Sym.	Value	Unit
U_x	1.181	$\mathrm{m\,s^{-2}}$
U_y	1.181	$\mathrm{m\,s^{-2}}$
U_z	4.667	$\mathrm{m\,s^{-2}}$

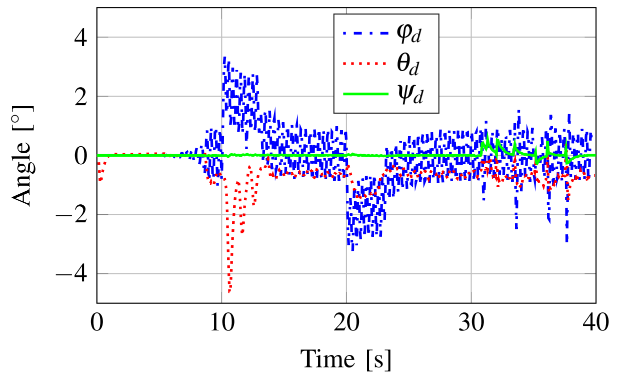

Fig. 9. Drone attitude when considering the nested control loops (14): numerical experiments in Gazebo.

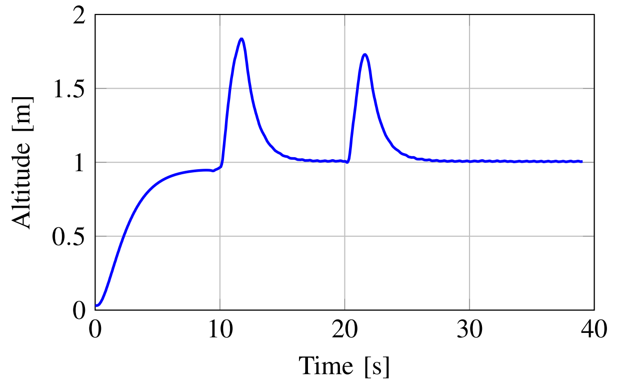

Fig. 10. Spikes on the drone altitude: numerical experiments in Gazebo.

we decided to apply the simple Algorithm 1 (see `http://tiny.cc/dnhy9y`) that gives acceptable results. The video available at `http://tiny.cc/sugs9y` illustrates the final behavior obtained from a SIL simulation.

IV. Conclusions

In this paper it has been shown how the design and the development of a control algorithm is affected by implementation issues. In particular, how the SIL simulation, here based on a ROS/Gazebo architecture, can be employed to exhibit behaviors of the overall system that in the Matlab/Simulink platform were hidden. Most of the implementation aspects related to the specific application have been presented and discussed thus showing the effectiveness and the relevance of the SIL methodology by considering a concrete complex application.

We published the software as open-source [11] with the aim to share our result with other researchers that might use the platform for testing their algorithms and understanding how SIL methodologies can improve the controller design.

Algorithm 1 Compensation of Rotors Speed Clipping

```
1: omMin $\leftarrow \Omega_1$, omMax $\leftarrow \Omega_1$, omFix $\leftarrow 0$
2: for $i := 2..4$ do
3:     if $\Omega_i <$ omMin then omMin $= \Omega_i$
4:     if $\Omega_i >$ omMax then omMax $= \Omega_i$
5: if omMin $< \Omega_{\min}$ then omFix $= \Omega_{\min} -$ omMin
6: else
7:     if omMax $> \Omega_{\max}$ then omFix $= \Omega_{\max} -$ omMax
8: for $i := 1..4$ do $\Omega_i = \Omega_i +$ omFix
9: Clip computed $\Omega_i$
```

References

[1] T. Özaslan, G. Loianno, J. Keller, C. J. Taylor, V. Kumar, J. M. Wozencraft, and T. Hood, "Autonomous Navigation and Mapping for Inspection of Penstocks and Tunnels With MAVs," *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1740–1747, 2017.

[2] C. Potena, R. Khanna, J. Nieto, R. Siegwart, D. Nardi, and A. Pretto, "AgriColMap: Aerial-Ground Collaborative 3D Mapping for Precision Farming," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1085–1092, 2019.

[3] D. Anthony, S. Elbaum, A. Lorenz, and C. Detweiler, "On crop height estimation with UAVs," in *IEEE International conference on Intelligent Robots and Systems*, 2014, pp. 4805–4812.

[4] R. C. Leishman, J. C. Macdonald, R. W. Beard, and T. W. McLain, "Quadrotors and Accelerometers: State Estimation with an Improved Dynamic Model," *IEEE Control Systems Magazine*, vol. 34, no. 1, pp. 28–41, 2014.

[5] E. Páll, L. Tamás, and L. Buşoniu, *Vision-Based Quadcopter Navigation in Structured Environments*. Springer International Publishing, 2015, pp. 265–290.

[6] M. Ryll, H. H. Bülthoff, and P. R. Giordano, "A Novel Overactuated Quadrotor Unmanned Aerial Vehicle: Modeling, Control, and Experimental Validation," *IEEE Transactions on Control Systems Technology*, vol. 23, no. 2, pp. 540–556, 2015.

[7] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," in *2004 IEEE International Conference on Intelligent Robots and Systems*, vol. 3, 2004, pp. 2149–2154.

[8] E. Rohmer, S. P. N. Singh, and M. Freese, "V-REP: a Versatile and Scalable Robot Simulation Framework," in *IEEE International Conference on Intelligent Robots and Systems*, 2013, pp. 1321–1326.

[9] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles," in *Field and Service Robotics*, 2017.

[10] H. Shokry and M. Hinchey, "Model-Based Verification of Embedded Software," *Computer*, vol. 42, no. 4, pp. 53–59, 2009.

[11] G. Silano, "BebopS GitHub repository," 2019. [Online]. Available: https://github.com/gsilano/BebopS

[12] Y. Bouzid, H. Siguerdidjane, and Y. Bestaoui, "Nonlinear internal model control applied to VTOL multi-rotors UAV," *Mechatronics*, vol. 47, pp. 49–66, 2017.

[13] B. L. Stevens, F. L. Lewis, and E. N. Johnson, *Aircraft control and simulation: dynamics, controls design, and autonomous systems*. John Wiley & Sons, 2015.

[14] T. N. Dief and S. Yoshida, "Review: Modeling and Classical Controller Of Quad-rotor," *International Journal of Computer Science and Information Technology & Security*, vol. 5, no. 4, pp. 314–319, 2015.

[15] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, "Rotors – A Modular Gazebo MAV Simulator Framework," in *Robot Operating System (ROS): The Complete Reference (Volume 1)*, K. Anis, Ed. Springer International Publishing, 2016, pp. 595–625.

[16] B. Arbanas, A. Ivanovic, M. Car, M. Orsag, T. Petrovic, and S. Bogdan, "Decentralized planning and control for UAV–UGV cooperative teams," *Autonomous Robots*, vol. 42, no. 8, pp. 1601–1618, 2018.

[17] N. T. Nguyen, I. Prodan, and L. Lefèvre, "Flat trajectory design and tracking with saturation guarantees: a nano-drone application," *International Journal of Control*, pp. 1–14, 2018.

[18] M. Ramp and E. Papadopoulos, "On Negotiating Aggressive Quadrotor Attitude Tracking Maneuvers Under Actuator Constraints," in *2018 26th Mediterranean Conference on Control and Automation*. IEEE, 2018, pp. 759–764.

[19] M. Faessler, A. Franchi, and D. Scaramuzza, "Differential Flatness of Quadrotor Dynamics Subject to Rotor Drag for Accurate Tracking of High-Speed Trajectories," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 620–626, 2018.