CrazyS: a software-in-the-loop platform for the Crazyflie 2.0 nano-quadcopter

Giuseppe Silano, Emanuele Aucone and Luigi Iannelli

***Abstract*— In this paper we propose CrazyS, an extension of the ROS (Robot Operating System) package RotorS, aimed to modeling, developing and integrating the Crazyflie 2.0 nano-quadcopter in the physics based simulation environment Gazebo.**

Such simulation platform allows to understand quickly the behavior of the flight control system by comparing and evaluating different indoor and outdoor scenarios, with a details level quite close to reality. The proposed extension expands RotorS capabilities by considering the Crazyflie 2.0 physical model and its flight control system, as well.

A simple case study has been considered in order to show how the package works. The use of open-source software makes the platform available for scientific and educational activities.

Fig. 1. The Crazyflie 2.0 nano-quadcopter.

I. Introduction

Unmanned Aerial Vehicles (UAVs), although originally designed and developed for defense and military purposes (e.g., aerial attacks or military air covering), during the last years gained increasing interest and attention related to civilian use. Nowadays, UAVs are employed for several tasks and services like surveying and mapping [1], for rescue operations in disasters [2], for buildings inspection [3], geophysics exploration [4], traffic monitoring [5], animal protection [6], agricultural crops [7].

Many existing algorithms for the autonomous control [8], [9] and navigation [10] are provided in the literature, but it is particularly difficult to make the UAVs able to work autonomously in constrained and unknown environments or also indoor. Thus, it follows the need for tools that allow to understand what it happens when some new applications are going to be developed in unknown or critical situations. Simulation is one of such helpful tools, widely used in robotics [11], [12], and whose main benefits are costs and time savings, enabling to carry out and to study complex missions that might be time consuming and risky in real world. Furthermore, bugs and mistakes in simulation cost virtually nothing: it is possible to crash a vehicle several times and thereby getting a better understanding of implemented methods under various conditions. To this aim, simulation environments are able to manage the complexity and heterogeneity of the hardware and applications, to promote the integration of new technologies, to simplify software design, to hide the complexity of low-level communication [13].

Different solutions, typically based on external robotic simulators such as Gazebo [14], V-REP [15], AirSim [16], are available to this purpose. They employ recent advances in computation and graphics (e.g., the AirSim photorealistic environment [17]) 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 complete software platforms able to test different algorithms for UAVs moving in a simulated 3D environment are becoming more and more important.

In this paper it is described CrazyS, a software package for modeling, developing and integrating the dynamics and the control architecture of the nano-quadcopter Crazyflie 2.0 (Fig. 1) in the Gazebo simulator. CrazyS is based on the Micro Aerial Vehicles (MAVs) simulation framework RotorS [18] and such contribution illustrates how it is possible to expand the functionalities of the simulation platform in the UAV field, by facilitating the development of different control strategies before testing them on a real platform, possibly with few changes, thanks to software-in-the-loop methodologies [9].

The chosen aircraft, the Crazyflie 2.0, is available on the market for less than $200 and it is ideal for many research areas (e.g., large swarm [19], path planning [20], mixed reality [21], etc.). The source code and the hardware are open, making it possible to go through any part of the system for complete control and full flexibility. New hardware and sensors can be linked through the versatile expansion ports, enabling the addition of the latest sensors. The small size and light weight reduce the need for safety equipment and increase the productivity. For all such reasons it appears valuable to have a realistic and detailed simulator of the Crazyflie dynamic behavior, with the possibility of validating in an easy way the effects of modifying the control architecture for complex missions.

Giuseppe Silano, Emanuele Aucone and Luigi Iannelli are with the Department of Engineering, University of Sannio in Benevento, Piazza Roma, 21 - 82100 Benevento, Italy, emails: emanuele.aucone@alice.it and {giuseppe.silano, luigi.iannelli}@unisannio.it

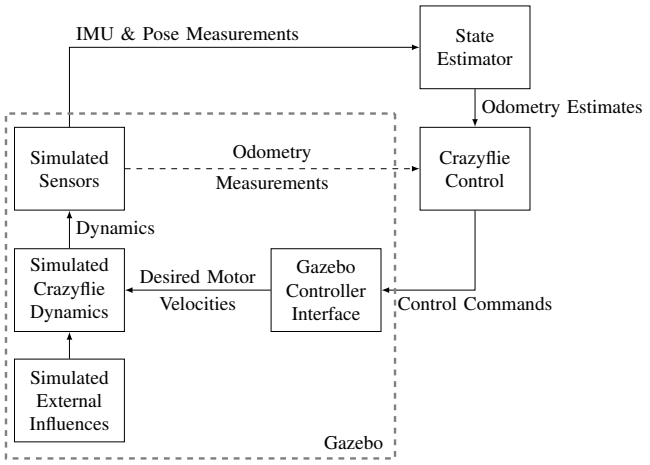

Fig. 2. Crazyflie 2.0 components in CrazyS. In the hovering example external influences are neglected.

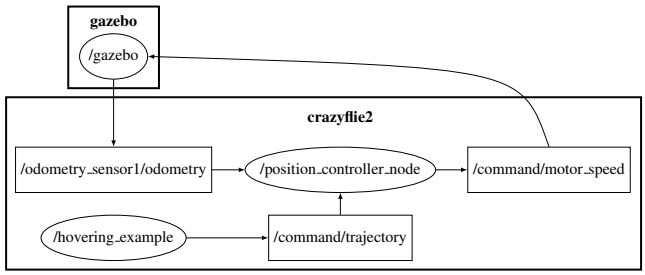

Fig. 3. Graph of ROS nodes (ellipses) and topics (squares) of the minimal hovering example with the Crazyflie 2.0. The continuous line arrows are topic subscription, with direction going from the subscriber node the the publisher one.

II. System Description

The focus of this section is to describe how the CrazyS simulator works, including ROS and Gazebo, by considering an illustrative application, i.e., the *hovering example*. An overview of the main components is represented in Fig. 2 while further details can be found in [18].

All the components of the nano-quadcopter are simulated by Gazebo plugins and the Gazebo physics engine. The body of the aircraft consists of four rotors, which can be placed in any location allowing configuration changes (e.g., from "+" to "×", see Sec. III), and some sensors attached to the body (e.g., gyroscope, accelerometer, camera, barometer, etc.). Each rotor has properly dynamics and accounts for the most dominant aerodynamic affects. Also external influences can be taken into account, such as a wind gust, but they are neglected in our case study.

To facilitate the development of different control strategies, a simple interface is provided. We developed a position control, but various are the solutions that can be used. Indeed, the simulator has to be meant as a starting point to implement more advanced control strategies for the Crazyflie. A further building block is the state estimator, used to obtain information about the state of the drone. While it is crucial on a real quadcopter, in simulation it can be replaced by a generic (ideal) odometry sensor (with or without noise and bias). In other words, the position, orientation, linear and angular velocities of the Crazyflie are directly provided by a Gazebo plugin. Section V illustrates results obtained when the aircraft sensors, i.e., accelerometer and gyroscope, are integrated in the loop.

All such features make the tool potentialities endless. Once the Crazyflie is flying, higher level tasks can be carried out and tested in the simulation environment, such as simultaneous localization and mapping (SLAM) [22], planning [23], learning [24].

In order to simulate a scenario close to the real world, we started from one of the available examples in RotorS (specifically the *mav_hovering_example.launch*) describing a quite detailed model of drone dynamics relative to an aircraft taking off and keeping indefinitely an hovering position.

Thus, we cast the model and control parts to that of the nano-quadcopter by taking into account the components of the simulation framework (see Fig. 2), the Crazyflie physical dynamics and parameters, and the perception sensors: the aim was to create a simulation environment that correctly describes the drone behavior. The overall scheme is depicted in Fig. 3 where the employed topics and nodes, parts of the structure of a ROS network, are represented. The whole process is the following: the desired position coordinates (x_r, y_r, z_r, ψ_r) are published by the *hovering_example* node on the topic *command/trajectory*, to whom the *position_controller* node (i.e., the Crazyflie controller) is subscribed. The drone state (*odometry_sensor1/odometry* topic) and the references are used to run the control strategy designed for the position tracking. The outputs of the control algorithm consists into the actuation commands (ω_1, ω_2, ω_3 and ω_4) sent to Gazebo (*command/motor_speed*) for the virtual rendering, so to update the aircraft position and orientation.

III. Model description and simulation

In RotorS (and, thus, in CrazyS), the drone is described by an Xacro file [25], i.e., an eXtensible Markup Language (XML) which is used to generate a more readable and often shorter XML code. The XML tag structure allows to set properties that are related to the physical features of the drone. Such structure has been used to describe the Crazyflie components and properties, i.e., the motors, the propellers, the mass of the vehicle, and so on.

Conversely, the robot geometry has been modeled by using the open-source software Blender. Starting from the mesh file available from [26], the propellers geometry has been changed from a "+" configuration (Crazyflie 1.0) to a "×" configuration (Crazyflie 2.0).

In order to develop a simulation environment as close as possible to the real platform, we also modeled the on-board Inertial Measurement Unit (IMU $MPU-9250$, [27]) as explained in [28]. In CrazyS, the measurements are modeled by two types of sensor errors affecting both the angular rate

	Sym.	Unit	Value
Gyroscope			
White noise density	n_ω	$\mathrm{rad/s}/\sqrt{\mathrm{Hz}}$	0.000175
Random walk	b_ω	$\mathrm{rad/s^2}/\sqrt{\mathrm{Hz}}$	0.0105
Bias correlation time	b_{t_ω}	s	1000
Turn on bias sigma	b_{ω_0}	$\mathrm{rad\,s^{-1}}$	0.09
Accelerometers			
White noise density	n_a	$\mathrm{m/s^2}/\sqrt{\mathrm{Hz}}$	0.003
Random walk	b_a	$\mathrm{m/s^3}/\sqrt{\mathrm{Hz}}$	0.18
Bias correlation time	b_{t_a}	s	300
Turn on bias sigma	b_{a_0}	$\mathrm{m\,s^{-2}}$	0.588

TABLE I
SUMMARY OF THE IMU MODEL PARAMETERS.

measurement $\tilde{\omega}$ and the linear acceleration $\tilde{a}$, computed as:

$$\tilde{\omega}(t) = \omega(t) + b_\omega(t) + n_\omega(t) \tag{1a}$$

$$\tilde{a}(t) = a(t) + b_a(t) + n_a(t), \tag{1b}$$

where $n_\bullet(t)$ is an additive noise term that fluctuates very rapidly (the white noise) and $b_\bullet(t)$ is a slowly varying sensor bias. All gyroscope and accelerometer axis measurements are modeled, independently. Table I summarizes all the model parameters and links them to the entries in the Xacro file. The accelerometer and gyroscope noise densities are easily recovered from the $MPU-9250$ datasheet. Instead, the bias part of the model (the "random walk") is rarely specified into datasheets and it can be characterized as

$$b_\bullet = \sigma\sqrt{T}, \tag{2}$$

if they are available σ, i.e., the rate noise density (aka spectral noise density), and T, i.e., the time period over which the idealized white noise process is integrated.

Finally, the *turn on bias* and the *bias correlation time* refer to the bias value originated when the inertial sensor turns on, and its time constant, respectively.

A. Dynamical model

The design of a flight control system for the quadcopter can be carried out through different approaches, from basic heuristic techniques to more advanced model based methods. Certainly, the latter ones exploit an accurate dynamical model of the plant, however also classical PID controllers might benefit from a detailed model. Indeed, after having defined the control structure possibly implemented through PID regulators, the tuning procedure could be based on a mathematical model of the process rather than working directly on the real plant, thus avoiding all problems (in terms of time, safety, costs) related to experimentally based calibration. Such model has been used in our work in the twofold way: firstly it has been employed for tuning heuristically (by looking at numerical simulations) the controller gains, and then for verifying the developed platform, by comparing simulation results with the outcome of the model implemented in Matlab/Simulink. In this way implementation details like C++ programming, controller discretization, concurrency, communication issues, can be isolated by looking at the Matlab/Simulink results and, furthermore, their effects can be investigated by considering the CrazyS simulations.

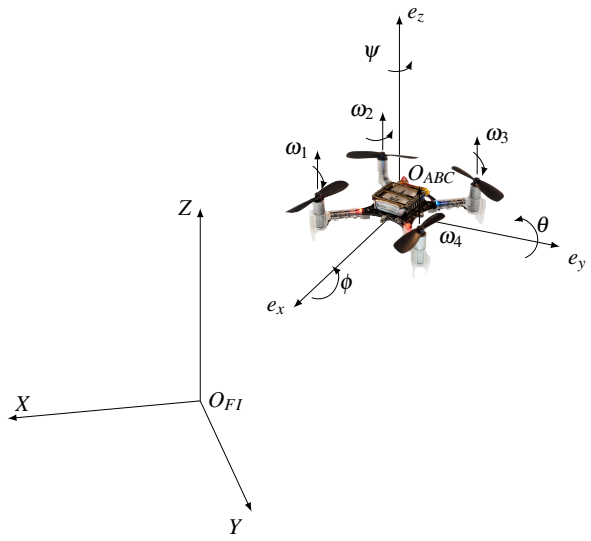

Fig. 4. Crazyflie in the body-frame (O_{ABC}) and the fixed-frame (O_{FI}) reference system. The forces from each rotors, the spin direction and the propellers velocity ω_i are also reported.

The dynamical model is derived in the usual way by introducing two reference systems: the fixed-frame O_{FI} (where FI stands for Fixed Inertial), also called inertial frame, and the body-frame O_{ABC} (where ABC stands for Aircraft Body Center) that is fixed in the aircraft center of gravity and oriented according to the aircraft attitude, see Fig. 4.

According to [29], the forces (eqs. (3) and (4)) and momentum (eqs. (5) and (6)) equations can be derived. Such model consists of nine equations for the system dynamics and four describing the inputs to the system (eqs. (7) and (8)) where $c_\bullet$, $s_\bullet$ and $t_\bullet$ denote $\cos(\cdot)$, $\sin(\cdot)$ and $\tan(\cdot)$ functions, respectively.

$$\begin{bmatrix}\dot{x}_d\\ \dot{y}_d\\ \dot{z}_d\end{bmatrix} = \mathbf{R}^T \begin{bmatrix}u_d\\ v_d\\ w_d\end{bmatrix} \tag{3}$$

$$\begin{bmatrix}\dot{u}_d\\ \dot{v}_d\\ \dot{w}_d\end{bmatrix} = \begin{bmatrix}0\\ 0\\ F_z/m\end{bmatrix} - \mathbf{R}\begin{bmatrix}0\\ 0\\ g\end{bmatrix} - \begin{bmatrix}p_d\\ q_d\\ r_d\end{bmatrix} \times \begin{bmatrix}u_d\\ v_d\\ w_d\end{bmatrix} \tag{4}$$

$$\begin{bmatrix}\dot{p}_d\\ \dot{q}_d\\ \dot{r}_d\end{bmatrix} = \mathbf{J}^{-1}\left(\begin{bmatrix}M_x\\ M_y\\ M_z\end{bmatrix} - \begin{bmatrix}p_d\\ q_d\\ r_d\end{bmatrix} \times \mathbf{J}\begin{bmatrix}p_d\\ q_d\\ r_d\end{bmatrix}\right) \tag{5}$$

$$\begin{bmatrix}\dot{\phi}_d\\ \dot{\theta}_d\\ \dot{\psi}_d\end{bmatrix} = \begin{bmatrix}1 & s_{\phi_d}t_{\theta_d} & c_{\phi_d}t_{\theta_d}\\ 0 & c_{\phi_d} & -s_{\phi_d}\\ 0 & s_{\phi_d}/c_{\theta_d} & c_{\phi_d}/c_{\theta_d}\end{bmatrix}\begin{bmatrix}p_d\\ q_d\\ r_d\end{bmatrix}, \theta_d \neq \pi/2. \tag{6}$$

The equations (3) and (6) describe the linear and angular velocities of the aircraft along x, y and z-axis in the O_{FI} frame, respectively. $\mathbf{R}^T$ is the rotation matrix from the body to the inertial frame. The Euler angles (ψ_d, θ_d and ϕ_d) are defined according to the ZYX convention [30]. The

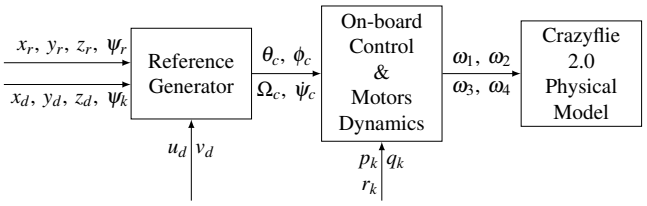

Fig. 5. The control scheme. Subscript c indicates the commands, r indicates the references, d indicates the drone variables and k indicates the sensors and data fusion outputs.

remaining six equations (eqs. (4) and (5)) describe the UAV linear and angular accelerations in the O_{ABC} frame. The diagonal matrix $\mathbf{J}$ has the inertia of the body about the x, y and z-axis, respectively, while m is the total mass of the quadcopter and g the gravitational constant.

The system inputs are reported in eqs. (7) and (8), where ω_1, ω_2, ω_3 and ω_4 represent the rotors angular velocities expressed in $\text{rad}\,\text{s}^{-1}$:

$$F_z = C_T\left(\omega_1^2+\omega_2^2+\omega_3^2+\omega_4^2\right), \tag{7}$$

$$M = \begin{bmatrix} M_x \\ M_y \\ M_z \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} C_T\, d\left(-\omega_1^2-\omega_2^2+\omega_3^2+\omega_4^2\right) \\ C_T\, d\left(-\omega_1^2+\omega_2^2+\omega_3^2-\omega_4^2\right) \\ \sqrt{2}C_D\left(-\omega_1^2+\omega_2^2-\omega_3^2+\omega_4^2\right) \end{bmatrix}. \tag{8}$$

Finally, d is the distance from the propellers to the center of gravity while C_T and C_D are the thrust and drag factors in hovering position, respectively. Further details are reported in [29] together with parameters values of the Crazyflie.

IV. Flight Control System

The flight control system follows the same idea discussed in [29]: a reference generator takes into account the position to reach and generates the command signals ($\dot{\psi}_c$, Ω_c, θ_c and ϕ_c) that are inputs for the on-board control architecture of the Crazyflie. The choice is in agreement with the aim of the work: to simulate in a virtual scenario the real aircraft behavior. Figure 5 describes the overall system while Figs. 6 and 7 describe the reference generator and the on-board control, respectively.

A. Reference generator

The reference generator uses the drone position (x_d, y_d and z_d) and the orientation along z-axis (ψ_k) to compute the command signals (θ_c, ϕ_c, Ω_c and $\dot{\psi}_c$). We assumed that the drone position and velocity come from a motion capture system (MoCap) that is modeled as an ideal virtual sensor in the simulation environment. The scheme in Fig. 6 summaries as it works.

The aim of the reference generator is to reach the position coordinates (x_r, y_r, z_r and ψ_r) by tuning the desired attitude (θ_c and ϕ_c), the heading velocity ($\dot{\psi}_c$) and the thrust (Ω_c) of the Crazyflie, later used as references for the on-board control system.

The thrust is expressed in terms of propellers speed, obtained by the sum of two terms: the feedforward ω_e and the

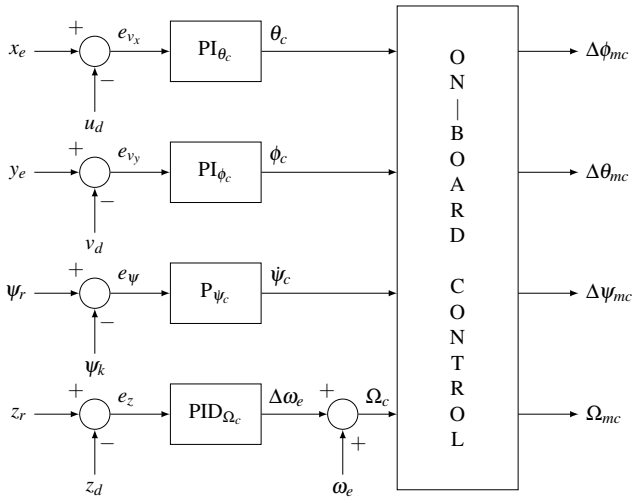

Fig. 6. The reference generator scheme. The obtained heuristic PID gains are: $K_{P_{\dot{\psi}_c}} = 3$, $K_{P_{\Omega_c}} = 14000$, $K_{I_{\Omega_c}} = 15000$, $K_{D_{\Omega_c}} = 20000$, $K_{P_{\theta_c}} = 15$, $K_{I_{\theta_c}} = 1$, $K_{P_{\phi_c}} = -15$ and $K_{I_{\phi_c}} = -1$.

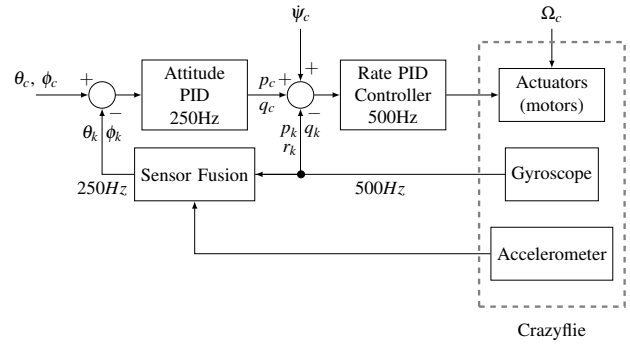

Fig. 7. On-board control architecture of Crazyflie 2.0, release 2018.01.01. The obtained heuristic gains are: $K_{P_{p_c}} = 3.5$, $K_{I_{p_c}} = 2$, $K_{P_{q_c}} = 3.5$, $K_{I_{q_c}} = 2$, $K_{P_{\Delta\phi_{mc}}} = 110$, $K_{P_{\Delta\theta_{mc}}} = 110$, $K_{P_{\Delta\dot{\psi}_{mc}}} = 110$ and $K_{I_{\Delta\dot{\psi}_{mc}}} = 16.7$.

closed loop $\Delta\omega_e$ components. As analyzed in [29], the angular velocity at equilibrium $\omega_e = 65\,673\,\text{rad}\,\text{s}^{-1}$ corresponds to the thrust force balancing the gravity. Instead, the reference signals x_e and y_e (see Fig. 6) are computed as:

$$x_e = (x_r - x_d)\cos(\psi_k) + (y_r - y_d)\sin(\psi_k) \tag{9a}$$

$$y_e = (y_r - y_d)\cos(\psi_k) - (x_r - x_d)\sin(\psi_k). \tag{9b}$$

Such signals are employed as setpoint for the velocities u_d and v_d [29], respectively.

B. On-board control system

The on-board control is decomposed into two parts: the attitude and the rate controller, both illustrated in Fig. 7. As a common rule in cascade structures, the inner loop needs to regulate at a rate faster than the outer.

In order to simulate the real quadcopter, we decided to keep the same on-board control architecture existing in the latest firmware release, the 2018.01.01.

Also the complementary filter, the default Crayzflie state estimator, has been developed according to the same

firmware release. Starting from the accelerometer and gyroscope data, the filter allows to estimate the attitude (ψ_k, ϕ_k and θ_k) and the angular velocities (p_k, q_k and r_k) used by the on-board control loop.

Finally, we modeled the actuators dynamics (see Fig. 7) considering also the relationship between the pulse with modulation (PWM) signals sent to the motors and the generated propellers speed (saturated to take into account the physical limitations of the system), as explained in [31],

$$\omega_i = \frac{\pi}{30}\left(\alpha \cdot PWM_i + q\right), \tag{10}$$

where $\alpha = 0.2685$ and $q = 4070.3$. Such actuators receive as inputs the rate controller signals $\Delta\phi_{mc}$, $\Delta\theta_{mc}$ and $\Delta\psi_{mc}$, i.e., the total variations from the equilibrium, and the thrust command Ω_{mc}:

$$\begin{cases} PWM_1 = \Omega_{mc} - \Delta\theta_{mc}/2 - \Delta\phi_{mc}/2 - \Delta\psi_{mc} \\ PWM_2 = \Omega_{mc} + \Delta\theta_{mc}/2 - \Delta\phi_{mc}/2 + \Delta\psi_{mc} \\ PWM_3 = \Omega_{mc} + \Delta\theta_{mc}/2 + \Delta\phi_{mc}/2 - \Delta\psi_{mc} \\ PWM_4 = \Omega_{mc} - \Delta\theta_{mc}/2 + \Delta\phi_{mc}/2 - \Delta\psi_{mc} \end{cases}. \tag{11}$$

V. Numerical results

Different aspects of the simulation environment have been investigated through numerical experiments and showed in this section.

A. Model validation

In order to validate the flight control system before writing its implementation code, we decided to implement first all control loops in a Simulink scheme and interface it with Gazebo for sending commands to and receiving data from the detailed aircraft physical model. To this aim, we used the MathWorks Robotics System Toolbox. It provides an interface between MATLAB and Simulink and ROS, thus allowing to move from the simulation scheme to the ROS network in an easy way. As shown in [32], the communication needs to be synchronized via the Gazebo services (*unpause* and *pause_physics*) that run and stop the simulation to avoid data losing and system instabilities.

Although the Robotics System Toolbox supports the `C++` code generation and it is able to generate a ROS node from a Simulink scheme, it is not immediate to integrate the generated code within an already developed platform, such as RotorS. Thus, the idea was to develop the code and to extend the platform paying attention to reuse software modules (e.g., launch files, messages types, nodes, etc.) already developed and functioning.

B. ROS integration

With the aim of a complete integration, the flight control system has been implemented as a ROS node (see Fig. 3) and interfaced to the Gazebo environment.

The overall system has been simulated through Gazebo/ROS and the results illustrate in a direct way how the system performs (the video is available, [33]). In particular, it is possible to see how the Crazyflie 2.0 keeps the hovering position until the simulation stops when ideal sensors are used. Moreover from the video [34] it appears evident how the control system is able to compensate attitude and position disturbances coming back to the hovering position.

A further scenario (video [35]) considers the "real" sensors (see Fig. 7) by taking into account the IMU and the complementary filter. The aircraft drift reproduces the real drone behavior when no MoCap systems are employed to obtain the aircraft attitude (ψ_d, ϕ_d and θ_d) and the orientation is replaced by the estimated values (ψ_k, ϕ_k and θ_k).

In Figure 8 numerical results obtained in Matlab/Simulink by considering the perfect state information are reported ("n" subscript signals, solid lines). Simulation results obtained in Gazebo/ROS ("s" subscript signals) are depicted, as well. In particular, dashed lines with ideal odometry while dotted lines with the real sensors are used. It is quite clear that real measurements make difficult to achieve a good position control without an effective estimation algorithm, while the small differences between Matlab/Simulink and Gazebo scenarios can be justified by the more accurate modeling in the 3D environment. On the other hand, the altitude controller works quite well in all considered scenarios.

VI. Conclusion

In this paper we presented CrazyS, a framework for simulating and integrating Crazyflie 2.0 with ROS and the Gazebo 3D environment. The tight integration with existing functionalities of the ROS package RotorS allows the comprehensive simulation of the quadcopter including the flight dynamics and any sensor available in the virtual reality environment. In this way, it has been proven the effectiveness and easiness of use of the platform not only for research but also for educational purposes, so that interested student might work in a known environment developing their own algorithms. Nevertheless, in our opinion the work could constitute the first step toward the development of a more structured platform aimed to the software-in-the-loop approach for such kind of applications.

We published the software as open-source [36] and at the same time we opened a pull request on RotorS repository [37] with the aim to share our result with other researchers that already use such platform.

VII. Acknowledgment

The authors are grateful to Benjamin Rodriguez for the help he gave during the development and implementation phases described in Sec. V-B during his MIT Independent Activities Period (IAP) spent at the University of Sannio.

References

[1] A. Loquercio, A. I. Maqueda, C. R. del Blanco, and D. Scaramuzza, "Dronet: Learning to fly by driving," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 1088–1095, April 2018.

[2] M. A. Stuart, L. L. Marc, and J. C. Friedland, "High Resolution Imagery Collection for Post-Disaster Studies Utilizing Unmanned Aircraft Systems," *Photogrammetric Engineering and Remote Sensing*, vol. 80, no. 12, pp. 1161–1168, 2014.

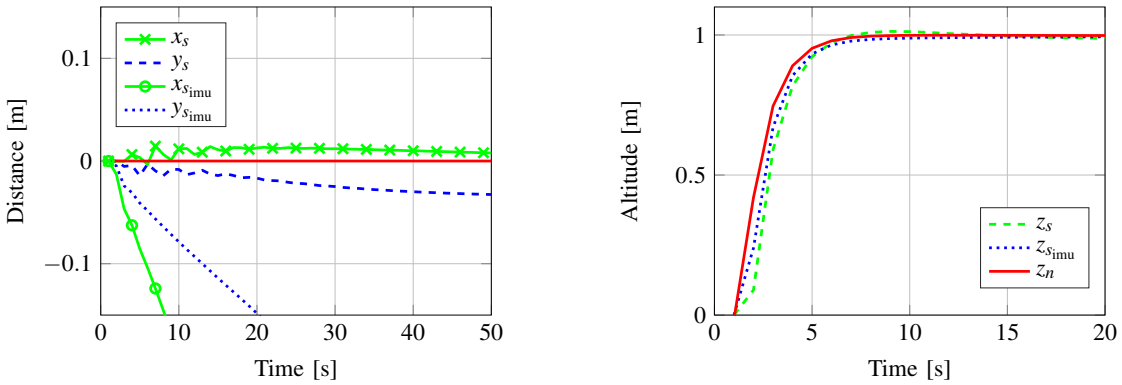

Fig. 8. Drone position during the hovering example. In red the numerical results and in blue and green the simulation results (Gazebo) with and without the real sensors.

[3] S. Choi and E. Kim, "Image acquisition system for construction inspection based on small unmanned aerial vehicle," *Lecture Notes in Electrical Engineering*, vol. 352, pp. 273–280, 2015.

[4] F. Fraundorfer, L. Heng, D. Honegger, G. H. Lee, L. Meier, P. Tanskanen, and P. M., "Vision-based autonomous mapping and exploration using a quadrotor MAV," in *IEEE International Conference on Intelligent Robots and Systems*, 2012, pp. 4557–4564.

[5] K. Kanistras, G. Martins, M. J. Rutherford, and K. P. Valavanis, "A survey of unmanned aerial vehicles (UAVs) for traffic monitoring," in *International Conference on Unmanned Aircraft Systems*, 2013, pp. 221–234.

[6] J. Xu, G. Solmaz, R. Rahmatizadeh, D. Turgut, and L. Boloni, "Animal monitoring with unmanned aerial vehicle-aided wireless sensor networks," in *IEEE 40th conference on local computer networks*, 2015, pp. 125–132.

[7] 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.

[8] B. Landry, "Planning and control for quadrotor flight through cluttered environments," Master's thesis, MIT, 2015.

[9] D. Ferreira de Castro and D. A. dos Santos, "A Software-in-the-Loop Simulation Scheme for Position Formation Flight of Multicopters," *Journal of Aerospace Technology and Management*, vol. 8, no. 4, pp. 431–440, 2016.

[10] T. Hinzmann, J. L. Schönberger, M. Pollefeys, and R. Siegwart, "Mapping on the Fly: Real-Time 3D Dense Reconstruction, Digital Surface Map and Incremental Orthomosaic Generation for Unmanned Aerial Vehicles," in *Field and Service Robotics*. Springer International Publishing, 2018, pp. 383–396.

[11] A. Tallavajhula and A. Kelly, "Construction and validation of a high fidelity simulator for a planar range sensor," in *IEEE Conference on Robotics and Automation*, 2015, pp. 1050–4729.

[12] R. Diankov and J. Kuffner, "Openrave: A planning architecture for autonomous robotics," Robotics Institute, Pittsburgh, PA, Tech. Rep. 79, 2008.

[13] A. Elkady and T. Sobh, "Robotics Middleware: A Comprehensive Literature Survey and Attribute-Based Bibliography," *Journal of Robotics*, 2012, article ID 959013.

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

[15] 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.

[16] 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.

[17] M. Mancini, G. Costante, P. Valigi, T. A. Ciarfuglia, J. Delmerico, and D. Scaramuzza, "Toward Domain Independence for Learning-Based Monocular Depth Estimation," *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1778–1785, 2017.

[18] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, *Robot Operating System (ROS): The Complete Reference (Volume 1)*. Springer International Publishing, 2016, ch. RotorS – A Modular Gazebo MAV Simulator Framework, pp. 595–625.

[19] J. A. Preiss, W. Honig, G. S. Sukhatme, and N. Ayanian, "Crazyswarm: A large nano-quadcopter swarm," in *IEEE International Conference on Robotics and Automation*, 2017, pp. 3299–3304.

[20] B. Araki, J. Strang, S. Pohorecky, C. Qiu, T. Naegeli, and D. Rus, "Multi-robot path planning for a swarm of robots that can both fly and drive," in *IEEE International Conference on Robotics and Automation*, 2017, pp. 5575–5582.

[21] W. Hönig, C. Milanes, L. Scaria, T. Phan, M. Bolas, and N. Ayanian, "Mixed reality for robotics," in *IEEE International Conference on Intelligent Robots and Systems*, 2015, pp. 5382–5387.

[22] O. Dunkley, J. Engel, J. Sturm, and D. Cremers, "Visual-inertial navigation for a camera-equipped 25g nano-quadrotor," in *IEEE International Conference on Intelligent Robots and Systems*, 2014, p. 2.

[23] L. Campos-Macías, D. Gómez-Gutiérrez, R. Aldana-López, R. de la Guardia, and J. I. Parra-Vilchis, "A hybrid method for online trajectory planning of mobile robots in cluttered environments," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 935–942, 2017.

[24] S. Bansal, A. K. Akametalu, F. J. Jiang, F. Laine, and C. J. Tomlin, "Learning quadrotor dynamics using neural network for flight control," in *IEEE Conference on Decision and Control*, 2016, pp. 4653–4660.

[25] S. Glaser and W. Woodall, "Xacro (2015)." [Online]. Available: https://wiki.ros.org/xacro

[26] T. A. C. of Teams (ACT) Lab, "GitHub Repository, RotorS fork, crazyflie-dev branch." [Online]. Available: https://goo.gl/tBbS9G

[27] B. AB, "Crazyflie 2.0 hardware specification," Bitcraze Wiki, 2018. [Online]. Available: https://goo.gl/1kLDqc

[28] J. Rehder, J. Nikolic, T. Schneider, T. Hinzmann, and R. Siegwart, "Extending kalibr: Calibrating the extrinsics of multiple IMUs and of individual axes," in *IEEE International Conference on Robotics and Automation*, 2016, pp. 4304–4311.

[29] C. Luis and J. Le Ny, "Design of a Trajectory Tracking Controller for a Nanoquadcopter," École Polytechnique de Montréal, Tech. Rep., 2016. [Online]. Available: https://arxiv.org/pdf/1608.05786.pdf

[30] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics - Modelling, Planning and Control*, 2nd ed., ser. Advanced Textbooks in Control and Signal Processing. Springer, 2008.

[31] G. Subramanian, "Nonlinear control strategies for quadrotors and cubesats," Master's thesis, University of Illinois, 2015.

[32] G. Silano, "Synchronization by using the Robotics System Toolbox," YouTube, 2018. [Online]. Available: https://youtu.be/ZPyMnu7A11s

[33] ——, "Crazyflie 2.0 hovering example," YouTube, 2018. [Online]. Available: https://youtu.be/pda-tuULewM

[34] ——, "Crazyflie 2.0 hovering example with disturbances," YouTube, 2018. [Online]. Available: https://youtu.be/sobBFbgkiEA

[35] ——, "Crazyflie 2.0 hovering example with real sensors," YouTube, 2018. [Online]. Available: https://youtu.be/ri7Xbo2iwFk

[36] ——, "CrazyS GitHub Repository," 2018. [Online]. Available: https://github.com/gsilano/CrazyS

[37] ——, "CrazyS GitHub pull request," 2018. [Online]. Available: https://github.com/ethz-asl/rotors_simulator/pull/465