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.

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.
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 and position by two types of sensor errors affecting both the angular rate and position.
measurement $\omega$ and the linear acceleration $\bar{a}$, computed as:

$$\omega(t) = \omega(t) + b_\omega(t) + n_\omega(t)$$

$$\bar{a}(t) = a(t) + b_a(t) + n_a(t),$$

where $n_\omega(t)$ is an additive noise term that fluctuates very rapidly (the white noise) and $b_\omega(t)$ 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

$$b_* = \sigma \sqrt{T},$$

if they are available $\sigma$, 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.

The dynamical model is derived in the usual way by introducing two reference systems: the fixed-frame $O_F$ (where $F$ 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 (9)) 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 $c_\omega$, $s_\omega$, and $t_\omega$ denote cos($\cdot$), sin($\cdot$) and tan($\cdot$) functions, respectively.

![Fig. 4. Crazyflie in the body-frame (OABC) and the fixed-frame (OF) reference system. The forces from each rotors, the spin direction and the propellers velocity $\omega_0$ are also reported.](image)

Matlab/Simulink results and, furthermore, their effects can be investigated by considering the CrazyS simulations.

The equations (3) and (9) describe the linear and angular velocities of the aircraft along $x$, $y$ and $z$-axis in the $O_F$ frame, respectively. $R^T$ is the rotation matrix from the body to the inertial frame. The Euler angles ($\psi_d$, $\theta_d$ and $\phi_d$) are defined according to the ZYX convention [30]. The
remaining six equations (eqs. 4 and 5) describe the UAV linear and angular accelerations in the \( O_{ABC} \) frame. The diagonal matrix \( J \) 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 \( \omega_1, \omega_2, \omega_3 \) and \( \omega_4 \) represent the rotors angular velocities expressed in \( \text{rad} \cdot \text{s}^{-1} \):

\[
F_z = C_T \left( \omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2 \right),
\]

\[
M = \begin{bmatrix} M_x & 0 & 0 \\ 0 & M_y & 0 \\ 0 & 0 & M_z \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} C_T d \left( -\omega_1^2 - \omega_2^2 + \omega_3^2 + \omega_4^2 \right) \\ C_T d \left( -\omega_1^2 + \omega_2^2 + \omega_3^2 - \omega_4^2 \right) \\ \sqrt{2} C_D \left( -\omega_1^2 - \omega_2^2 - \omega_3^2 - \omega_4^2 \right) \end{bmatrix}. \tag{8}
\]

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 (\( \psi_r, \Omega_c, \theta_r \) and \( \phi_r \)) 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 (\( \psi_d \)) to compute the command signals (\( \theta_r, \phi_r, \Omega_c \) and \( \psi_r \)). 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 \( (x, y, z) \) and \( \psi_r \) by tuning the desired attitude (\( \theta_r \) and \( \phi_r \)), the heading velocity (\( \psi_r \)) and the thrust (\( \Omega_c \)) 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 \( \omega_c \) and the
firmware release. Starting from the accelerometer and gyroscope data, the filter allows to estimate the attitude ($\psi_k$, $\phi_k$ and $\theta_k$) 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],

$$\omega_k = \frac{\pi}{30} (\alpha \cdot PW M_i + q),$$  \hspace{1cm} (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 $\Omega_{mc}$:

$$\begin{align*}
PWM_1 &= \Omega_{mc} - \Delta \phi_{mc}/2 - \Delta \theta_{mc}/2 - \Delta \psi_{mc} \\
PWM_2 &= \Omega_{mc} + \Delta \phi_{mc}/2 - \Delta \theta_{mc}/2 + \Delta \psi_{mc} \\
PWM_3 &= \Omega_{mc} + \Delta \phi_{mc}/2 + \Delta \theta_{mc}/2 - \Delta \psi_{mc} \\
PWM_4 &= \Omega_{mc} - \Delta \phi_{mc}/2 + \Delta \theta_{mc}/2 - \Delta \psi_{mc}.
\end{align*}$$

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 ($\psi_d$, $\phi_d$ and $\theta_d$) and the orientation is replaced by the estimated values ($\psi_k$, $\phi_k$ and $\theta_k$).

In Figure 8 numerical results obtained in Matlab/Simulink and interfaced to the Gazebo environment. 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.

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.

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


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.