IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received November 24, 2020, accepted December 1, 2020, date of publication December 11, 2020,
date of current version December 28, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3044098

High-Level Modular Autopilot Solution for Fast Prototyping of Unmanned Aerial Systems

CARLOS RODRÍGUEZ DE COS, (Member, IEEE), MANUEL J. FERNANDEZ, PEDRO J. SANCHEZ-CUEVAS, (Member, IEEE), JOSÉ ÁNGEL ACOSTA, AND ANIBAL OLLERO, (Fellow, IEEE)
GRVC Robotics Laboratory, Universidad de Sevilla, 41092 Sevilla, Spain

Corresponding author: Carlos Rodríguez de Cos (crdecos@us.es)

This work was supported in part by the HYFLIERS and AERIAL-CORE projects of the EU Horizon 2020 programme under Grant 779411 and Grant 871479, and in part by the ARM-EXTEND project of the Spanish Ministerio de Economía, Industria y Competitividad, under Grant DPI2017-89790-R. The work of Carlos Rodríguez de Cos and Pedro J. Sanchez-Cuevas was supported by the V Plan Propio de Investigación (VPPI) of Universidad de Sevilla and the Spanish Ministerio de Educación, Cultura y Deporte Formación de Profesorado Universitario (FPU) programme.

ABSTRACT A redundant fast prototyping autopilot solution for unmanned aerial systems has been developed and successfully tested outdoors. While its low-level backbone is executed in a Raspberry Pi® 3 + NAVIO2® with a backup autopilot, the computational power of an Intel® NUC mini-computer is employed to implement complex functionalities directly in Simulink®, thus including in-flight debugging, tuning and monitoring. Altogether, the presented tool provides a flexible and user-friendly high-level environment with enhanced computational capabilities, which drastically reduces the prototyping timespans of complex algorithms –between 50% and 75%, according to our long and proven experience in aerial robotics–, while preventing incidents thanks to its redundant design with a human-in-the-loop pilot on the reliable PX4. Three typical outdoor cases are carried out for validation in real-life scenarios, all mounted in a DJI© F550 platform. Full integration results and telemetry for more than 50 hours of outdoor flight tests are provided.

INDEX TERMS Unmanned autonomous systems (UASs), aerospace systems and applications, applications (robotics), middleware, redundancy, PX4, fast prototyping.

I. INTRODUCTION

The increasing complexity of new applications for small Unmanned Aerial Systems (UAS) is demanding autopilots with higher computational power and, more importantly, faster prototyping timespans. The novel branch of Aerial Robotics is pushing forward in this direction, mainly due to its broad range of applications, such as contact inspection, maintenance, and assembling at inaccessible and/or dangerous places. For a detailed literature review on this emerging area, we refer interested readers to cutting-edge projects [1]–[3] and its ongoing follow-up in [4] (see survey papers [5], [6]), the overview of aerial manipulation in [7], the applications with multi-joint manipulators in [8], [9], [10] and [11], the self-coordinated approach in [12] and [13], and to the *ad-hoc* autopilot in [14]. In particular, in [2] we challenged ourselves to complete complex missions outdoors, being the inter-consortium integration and code customisation difficulties found while implementing new complex algorithms the main motivation of this work.

To cope with these obstacles, most of the aerial platforms depend whether on autopilots based on open-source software compiled in target computer boards (e.g. PX4 in [15], [16]), or on *ad-hoc* solutions made by research groups to fulfil their needs, e.g. in [17]. Thus, advances in these complex functionalities are implemented either modifying the open-source code to specifically implement new challenging guidance and/or control strategies, as in [18] for Model Predictive Control (MPC) or in [19] for backstepping sliding mode control (SMC), or for general purpose applications, as in [20], [21] and [22]; or linking it with previously built software, as in the multi-layer approach in [23], the model-based design and validation in [24] or the architecture for a nonlinear controller in [25]. Although these open autopilot options provide a wide range of well-polished features that are sufficient in most outdoor[1] cases, their thorough internal integration

The associate editor coordinating the review of this manuscript and approving it for publication was Cong Pu.

[1]Applications including wired appliances to support computations off-board –as e.g. [26], [27]– are omitted due to the outdoor and autonomous nature of our design, i.e. with a completely onboard implementation system.

distorts the project schedules by expanding the time devoted to the customisation of source code. To illustrate this point, a comparison between widely used autopilots for UAVs[2] and the proposed Modular Autopilot Solution $MAS^{\boxplus}_{PX4}$is shown in Table 1. It is worth noting that this study is based on the deep knowledge acquired by our research group after more than 50 aerial projects along the last 10 years, allowing us to identify common drawbacks in the early stages of prototyping of these autopilot solutions:

- The implementation of customised functionalities generally demands offline *ad-hoc* modifications of the native source code, resulting in longer debugging times during the integration stages.
- The implementation of complex algorithms is not possible in the existing computer boards, mainly due to the lack of computational power and/or to their closed nature, which, in turn, might compromise their feasibility.

Even though the prolongation of the timespans by the former could be tolerated on paper, the solution commonly given to adhere to the project deadlines when it occurs is to limit the extension of the advances in complex functionalities. In contrast, we found the latter a hard constraint while executing dexterous manipulation tasks in the onboard computer in [2], mainly due to the coverage of localisation, vision, flight control and manipulation subtasks. Altogether it ends up forcing a trade-off that downgrades the implementation, and hence, the overall performance. To overcome these limitations, the main contributions of the $MAS^{\boxplus}_{PX4}$ autopilot solution are:

C1. Drastic reduction of programming and debugging times. The full use of the user-friendly and highly **modular** Simulink® [35] environment expedites the software development and implementation of complex algorithms.

C2. High-level in-flight code debugging, tuning and monitoring, while preserving the autopilot **redundancy** for unexpected incidents and/or learning capabilities.

C3. Enough **computational power** to implement and execute heavy and complex algorithms onboard, with no need for downgrading the implementation.

C4. Nonetheless, selected functionalities of PX4 can be kept running, thus providing a step-by-step way to implement fast-prototyping functionalities as desired. For instance, here we keep the EKF estimator and the motor mixer and modify the planning and control core algorithms.

Remark 1: We underscore that the fast prototyping potential relies on two novel specifications. First, the mainframe Simulink® environment whose modularity and fast integration capabilities are widely known, facilitating the external validation and/or cooperation between research groups. Secondly, to the best of the authors' knowledge, $MAS^{\boxplus}_{PX4}$ is the first autopilot with in-flight human-in-the-loop redundancy and debug capabilities, drastically reducing the implementation and tuning times.

Remark 2: It is also worth highlighting the flexibility of the $MAS^{\boxplus}_{PX4}$ framework in terms of computational capabilities. In here, we have proposed a particular hardware solution with an Intel® NUC computer board, but it could be replaced by others. For example, sooner than later the All Up Weight (AUW) is expected to be reduced, as it evinces the arrival of lighter computers, such as [37] or the recent release [38] with an NPU of 5 TOPS.[3]

The paper is structured as follows: Section II describes the hardware and software architectures as well as a thorough description of the autopilot low-level implementation, and Section III is devoted to the experimental validation in three separate cases of study. Finally, the paper is wrapped up with a conclusion section.

TABLE 1. Comparison between $MAS^{\boxplus}_{PX4}$ and alternative autopilots.

	Modularity & customisability[a]	In flight tuning	Need of full compilation	Redundancy
$MAS^{\boxplus}_{PX4}$	High Level[b]	Yes	No[f]	Yes
PX4	Low Level[c]	Yes[d]	Yes	No
ROSflight	Low Level[c]	Yes	Yes	No
ArduPilot	Low Level	Yes[d]	Yes	No
DJI	No[e]	Yes	Yes	No
Paparazzi	Low Level	Yes	Yes	No

[a]Understood as the possibility of actively and directly modifying *all* the control loops, from the ground operator inputs to the generalised force signals.

[b]High level means here i) implementing algorithms using an interface which natively does *not only* depend on low-level code, and ii) not demanding the user to get involved into the integration apart from the minimal interconnection variables.

[c]Simulink external mode can be used with deprecated PX4 versions for Pixhawk, not allowing complex algorithms.

[d]Tuning available via MAVlink using tools such as Mission Planner or QGroundControl (see [36] for further details).

[e]The DJI software in the board cannot be modified directly. References to some of the control levels can be demanded via SDK using another computer board.

[f]Simulink® invokes a model interpreter. Nonetheless, this interpretation is significantly faster than PX4 and more user-friendly than compiling a standalone executable or a binary file. A detailed and thorough discussion about the differences between compilation and interpretation in this graphical environment is included in Appendix A.

II. AUTOPILOT ARCHITECTURE

As aforementioned, the main goal of the proposed architecture is to obtain a safe autopilot that facilitates the fast-prototyping implementation of complex algorithms onboard, thus overcoming the previously presented limitations and allowing their full experimental demonstration.

This is achieved by making a reliable environment that integrates: i) the customised algorithms in Simulink® –more complex and optimised over a wider range of conditions–; and ii) the flight stack PX4 (see [15], [16]) –simpler, more reliable, and well-known for having been used for more than 10 years in the majority of our prototypes of different projects–. A detailed discussion of similar types of software

[2]Only solutions validated in flight are considered, thus excluding general purpose and/or educational approaches, such as [28]–[34]. In any case, the only redundant alternative is $MAS^{\boxplus}_{PX4}$.

[3]TOPS stands for Tera Operations Per Second.

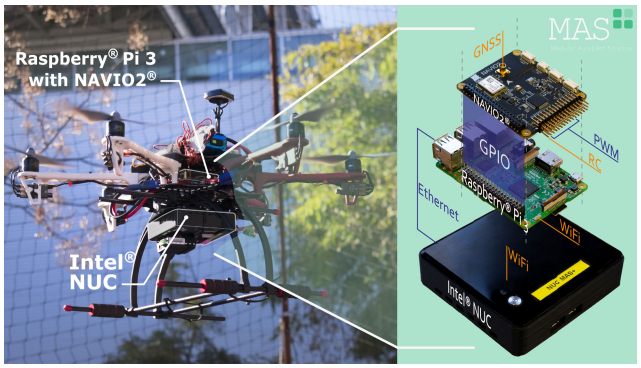


FIGURE 1. Validation platform in flight with the hardware solution used as an exploded detail view highlighting the setup.

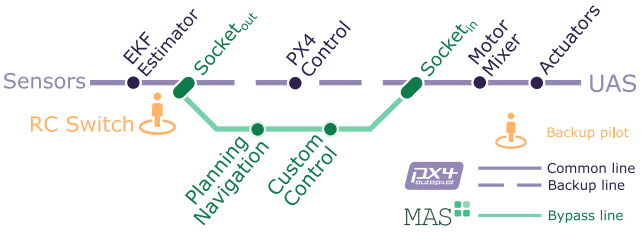


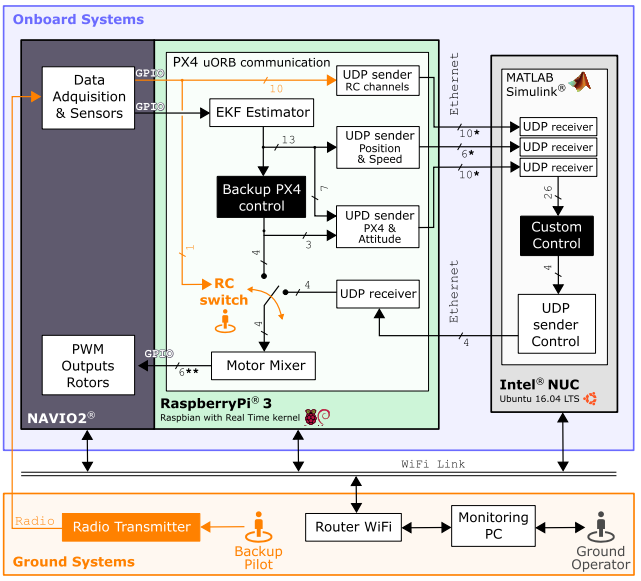
FIGURE 2. Redundant MAS$^{\oplus}_{PX4}$ solution as a schematic diagram (similar to a transit map) with the modifications in aquamarine bypassing the backup PX4 autopilot in purple via the operator switch in orange.

architectures in commercial flight control systems is provided in [39], of course despite the apparent differences. The actual hardware implementation of this solution is shown in Fig. 1 and a sketch of its global software functionality –highlighting the backup and bypass lines– is depicted in Fig. 2.

In particular, the proposed solution is implemented with custom daemon modules, located after the state estimator and before the motor mixer[4]: i) the PX4 ecl/EKF state estimation is sent to the customisation environment, making use of the current reliable estimator, which is not the focus of this work; and ii) both redundant controllers, i.e. the cascade-like of PX4 and the customised one, join at the command switch (RC Switch in Figs. 2 and 3).

In this second point, the roles of the two agents involved in the experimentation phase come into action. On the one hand, the ground operator, an expert engineer involved in the development of the algorithm being validated, confirms the correct behaviour of the new functionalities. On the other hand, the backup pilot (or pilot in command) is in charge of guaranteeing the safety conditions of aircraft during the flight. If during their task, this pilot detects any risk in the experiment, they would trigger the backup switch, bypassing the prototyped algorithm to the backup PX4 controller. It is worth noting that the latter has been previously tested and tuned to be used as a safe reliable controller.

[4]This PX4 version is available at https://github.com/grvcTeam/MASp.



*Associated to these signals, a 1 byte timestamp is added to the standard 4-byte messages used per signal.

**Dependant on the number of rotors (6 for an hexarotor).

FIGURE 3. Architecture for MAS$^{\oplus}_{PX4}$, with both human agents included in the ground systems and highlighting the RC switch.

A. HARDWARE ARCHITECTURE

The hardware implementation –schematically depicted in Fig. 3– consists of a Raspberry Pi® 3 [40] equipped with a NAVIO2® as carrier [41], used for both the backup controller and the state estimation; and an Intel® NUC [42] as the main computer. In this approach, the Raspberry Pi® 3 is adopted as the interface board and delegated the backup autopilot, and the Intel® NUC is dedicated to implementing customised algorithms directly on Simulink® modules (e.g. control, planning or navigation), maximising the computational capacity. All of these make the autopilot modular, customisable and safe, due to the redundancy master backup switch.

Remark 3: It is important to highlight that those modules do not need to be compatible with the Simulink® external mode, thus guaranteeing broader compatibility in normal mode and full functionality. Additionally, accelerator mode can be used if needed, with its implicit limitations (see Appendix A).

For the sake of completeness, a comparison with other environments is in order. On the one hand, the proposed autopilot has in common with the Dronecode ecosystem [43] –which includes PX4– the possibility of using the hardware Raspberry Pi® 3 + NAVIO2®. However, none of the projects fostered by Dronecode employs an external computer board for the implementation of complex algorithms. Additionally, the ROS capabilities could be enabled through MAVLink/MAVROS (see [36]) –as in Dronecode– or via the ROS toolbox in Simulink®, and hence, both providing full connectivity. On the other hand, the proposed autopilot significantly improves the technical specifications of another

TABLE 2. UDP connection results from both sides (25 MB packages).

Port		Intel® NUC			Raspberry Pi® 3		
		BW [Mb/s]	Jitter [ms]	Lost Data	BW [Mb/s]	Jitter [ms]	Lost Data
8081	I/O	1.04	0.49	0/1785	1.04	0.488	0/1785
8082	I/O	1.04	0.54	0/1785	1.04	0.542	0/1785
8083	I/O	0.96	1.37	0/1785	0.96	1.366	0/1785
8888	O/I	0.96	0.56	0/1785	0.96	0.557	0/1785

common solution, Pixhawk [44], which is known for being only valid in simple and slow tasks —due to its PID implementation and sample frequency—, and for not supporting Ethernet communications, unlike the proposed approach.

Additionally, it is also worth mentioning ROSflight [45], a recently appeared board with which the $MAS^{\boxplus}_{PX4}$ environment concurs on some characteristics. Both share, for instance, the implementation of new algorithms on a companion computer board, and their sensor and actuator interfaces are similar to our Raspberry Pi® 3 + NAVIO2® approach. Nonetheless, although ROSflight also focuses on easing the development stages –in particular the low-level coding–, it is based on ROS; while in $MAS^{\boxplus}_{PX4}$, this feature is optional and the prototyping interface is the user-friendly graphical interface Simulink®. Additionally, $MAS^{\boxplus}_{PX4}$ includes redundancy with the mature PX4 backup, thus offering the possibility of progressively implementing custom functionalities safely.

Moving back to $MAS^{\boxplus}_{PX4}$, it is worth noting that a trade-off between the computational power and the endurance has to be *a priori* decided, as usual in electrically-powered UAS applications. As commented in Remark 2, for lower computational missions, the external board Intel® NUC can be replaced by any other that reduces the AUW of the benchmark platform (3.2 Kg), and improves its endurance (about 8 min).

B. SOFTWARE INTEGRATION

The core of this reliable implementation is the integration of PX4 with the prototyping environment. A thorough analysis is therefore dedicated to the internal UDP communications,[5] which over a robust Ethernet communication form the *sine-qua-non* condition for the reliability of the whole solution.

On the one hand, the quality of the Ethernet UDP connection is studied in terms of bandwidth, jitter and lost datagrams in a 20 seconds experiment with overestimated[6] packages (25 MB) using iPerf [46]. This tool calculates jitter as the difference between the receiving and sending times –via a timestamp added to the package sent by the client–, correcting any discrepancy due to the lack of clock synchronisation. The results are then collected in Table 2, confirming that no datagrams were lost during the test and that a bandwidth of 1 Mb/s is more than enough for the application. In what

[5]For the sake of completeness, the pseudocode for the UDP modules is shown as an appendix.

[6]This assures the possibility of future scalability.

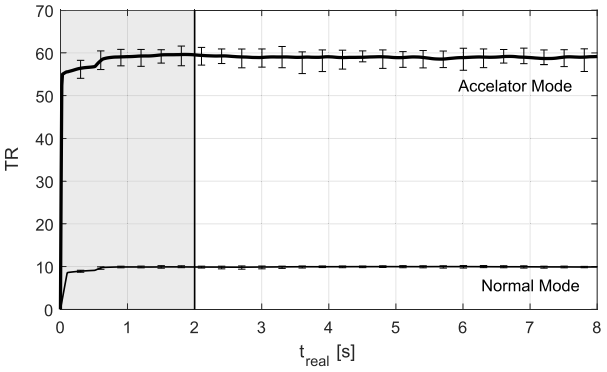


FIGURE 4. Time ratio for 20 tests of 8 seconds, on normal and accelerator modes, in an Intel® NUC 5i7RYH, where the solid lines correspond to mean values and the error bars to most distant value of these tests.

respects to the jitter, it should be highlighted that, although the package congestion in UDP protocols is lower than in TCP, this fluctuation could also have an impact on UDP applications. Nonetheless, the provided results evince that the jitter in the proposed solution is negligible.

On the other hand, a preliminary study on the Round Trip Time (RTT) –that includes the communication and the customised code execution– is also conducted. For that purpose, the time between a simple RC command is given and its impact on the controller output is measured.[7] As a result, we obtain a mean RTT estimation of 1235 µs using the controller in Subsection III-A. In comparison, the PX4 output runs at 250Hz (4000 µs), being the RTT of this application below this sample time. Nonetheless, the influence of the complexity of the customised algorithm on the execution times should also be studied to complete this analysis. This is done in part in Subsection II-C but more specifically in the load test for a computational-demanding implementation in Fig. 8, showing that the computational capabilities of the solution are well above the current demands. Accordingly, we can conclude that an increase in the complexity of the algorithms implemented can be compensated by the computational power margins of the proposed environment. Considering this, the RTT is not being significantly affected.

C. SIMULATION VERSUS REAL-TIME OVERSAMPLING

The proposed solution is designed to use the full potential of the Simulink® environment (i.e. avoiding incompatibilities), and to reduce the simulation-to-experimentation times at the expense of real-time. This results in two phenomena: time-variant random disturbances and high-frequency spurious signals. To study both of them, let us define the time ratio *TR* as the rate between the clock time according to the simulation records and its real value according to the PX4 board clock.[8] A thorough batch of experimental tests was carried out and,

[7]Without passing by the EKF estimator, being considered the contributions of this step to the RTT negligible for this analysis.

[8]Notice that the implementation of explicit time-variant terms, such as integrals, will need the corresponding correction with this time ratio.

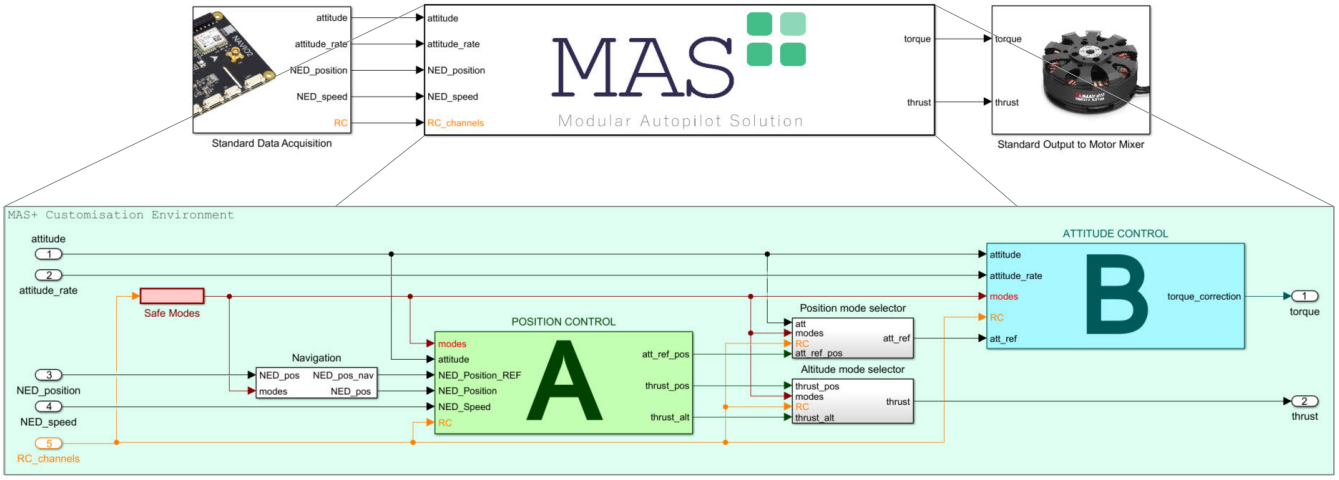


FIGURE 5. MAS$^{\boxplus}_{PX4}$ customisation environment including the standard Simulink® communication modules (upper) with a detailed view of the main module, highlighting the position (A) and attitude (B) controllers, together with the navigation module and the flight modes (lower).

as shown in Fig. 4, the *TR* is only altered during short initial transients of about 2 seconds, thus being assimilable to the time convergence of the ecl/EKF estimator in PX4. Consequently, this time ratio can be considered constant in the design of custom algorithms (but being recommended to re-calculate its value after significant changes in the complexity of the algorithms to re-adjust time-varying terms).

Moreover, the potential problems that may arise for high-frequency residual signals were also analysed in depth. The experimental tests evince that in all cases those spurious signals were completely filtered by the propulsion system dynamics and other common UAS filters, as well as for the 200-250Hz sampling rate limitation of the PX4 motor mixer uORB. In summary, no incidents were detected in more than 50 hours of flight, and therefore the influence of signal oversampling is considered negligible in UAS applications.

III. EXPERIMENTAL VALIDATION (EV)

The validation of the autopilot is demonstrated through three typical implementation cases, thus providing a general overview of its possibilities in real-life scenarios, and showing the friendly fast-prototyping implementation of simple and complex algorithms. These experiments, in which a standard DJI© F550 frame [47] (see Fig. 1) is selected as the platform to mount the autopilot, are:

EV1. The progressive development of a cascade PID controller mimetic to the controller in PX4, to illustrate the redundancy and reliability of MAS$^{\boxplus}_{PX4}$ and to serve as an example of the implementation of a complete strategy.

EV2. A complex nonlinear control algorithm *without any simplification*, to reveal the computational capabilities of the autopilot as well as to demonstrate its feasibility for more complex implementations. Additionally, since the algorithm runs in faster-than-real-time, its implementation stability is analysed.

EV3. A navigation module, to illustrate the wide range of possibilities MAS$^{\boxplus}_{PX4}$ offers for fast prototyping apart from control (not be confounded with a final evaluation). Accordingly, the controller chosen to run underneath this module is the cascade PID in EV1, thus assuring that those researchers not mainly interested in control can implement their modules without getting discouraged by the complexity and lack of intuitiveness of the controller.

The procedure followed to implement these cases –thought to avoid incidents while changing the core algorithms– is also worth mentioning. Firstly, the basic functionalities of an initial solution in Simulink® progressively replace their homologous in PX4, testing these changes in a confined scenario with the platform suspending from a structure. When all these basic changes are checked, the limitations of the scenario are relaxed by removing the suspension of the platform, thus allowing a realistic evaluation of the novel functionalities. Finally, the obtained control strategy is demonstrated in a non-confined scenario equivalent to a fully autonomous flight outdoors, testing the higher-level capabilities. As a result, a reliable complete solution directly evolving from a simulation-based development is obtained.

To ease understanding of the implementation of the algorithms for EV1-EV3, in Fig. 5 we show the general modular structure of the Simulink® environment. This includes the inputs and outputs and the position (A) and attitude (B) subsystems, whose contents will be detailed in each experiment.

A. EV1: SIMPLE PID CONTROLLER MODULE

A standard PID cascade controller is implemented by progressively replacing parts of its PX4 counterpart and tuning them from the inner to the outer loop, as follows: i) Attitude rates; ii) Attitude; iii) Altitude; iv) Cartesian speed; and, finally, v) Cartesian position, resulting in a semi-autonomous controller. As previously mentioned, this cascade controller

is intended to be mimetic to the native controller in PX4, but implemented in Simulink® using the Intel® NUC. This is used both to validate the customisation environment and to illustrate the redundancy capabilities of the solution.

In Fig. 6 we depict the Simulink® implementation for subsystems (top), together with the results for both attitude and position controllers (bottom), respectively. Moreover, all references, including position, attitude and their associated rates, are tracked with minimal steady-state error, similarly to the error in PX4. Altogether, this demonstrates that both controllers can be considered mimetic and corroborates the whole reliability of the autopilot.

B. EV2: COMPLEX NONLINEAR CONTROLLER MODULE

To exemplify the computational capabilities, we implement a cascade nonlinear controller based on feedback linearisation and command-filtered backstepping. This type of controllers –together with the cascade PIDs– are the most commonly used in this kind of multi-rotor platform. Although standard in aerial robotics, the one implemented here is an adapted version of [48] –and references therein–, being briefly summarised next. Thus, let $U = \text{col}(U_1, U_2, U_3) \in \mathbb{R}^3$ be the virtual command in the Cartesian position outer loop of the cascade, hence becoming the feedback-linearisation controller

$$U = \ddot{p}_{ref} + K_D\,\dot{p}_e + K_P\,p_e + K_I\left(p_e^{I,0} + \int_{t_0}^{t} p_e\,d\tau\right),$$

with $p_{ref} \in \mathbb{R}^3$ the cartesian position reference and $p_e \in \mathbb{R}^3$ its associated error; $K_P, K_I, K_D \in \mathbb{R}^{3\times3}$ the matrix control gains; and $p_e^{I,0} = \text{col}(0, 0, z_e^{I,0})$ a feedforward integral action counteracting the initial vertical forces, $z_e^{I,0}$. This outer loop provides the attitude reference commands for the inner and nonlinear control loop reading

$$\begin{aligned} T &= m\,\|U - g\|, \\ \tan\theta_{ref} &= \frac{U_1\cos\psi + U_2\sin\psi}{U_3 - \|g\|}, \\ \sin\phi_{ref} &= -\frac{U_1\cos\psi - U_2\sin\psi}{(T/m)}, \end{aligned}$$

where $m \in \mathbb{R}$ is the UAV mass, $g \in \mathbb{R}^3$ the gravity acceleration vector, $T \in \mathbb{R}$ the total thrust and $\psi \in \mathbb{R}$ the yaw angle.

On the other hand, the attitude controller of the inner loop is formulated using the command-filtered backstepping technique (see references therein [48]), yielding

$$\begin{aligned} \tau_a &= \Omega \times I\Omega + \Gamma_2\,Z_2 + I(W^\top \bar{Z}_1 + \dot{\Omega}_{ref}), \\ \dot{\Omega}_{ref} &= -\bar{\Gamma}(\Omega_{ref} - \Omega_d), \\ \dot{\epsilon} &= -\Gamma_1\epsilon - W(\Omega_{ref} - \Omega_d), \end{aligned}$$

with $\tau_a \in \mathbb{R}^3$ the control torque produced by the differential thrust between the rotors (see [49] and references therein); $\Theta = \text{col}(\phi, \theta, \psi) \in \mathbb{R}^3$ and $\Omega \in \mathbb{R}^3$ the attitude and attitude rate, with $W(\Theta) \in \mathbb{R}^{3\times3}$ their associated matrix; $I \in \mathbb{R}^{3\times3}$ the inertia matrix of the UAV in the body frame;

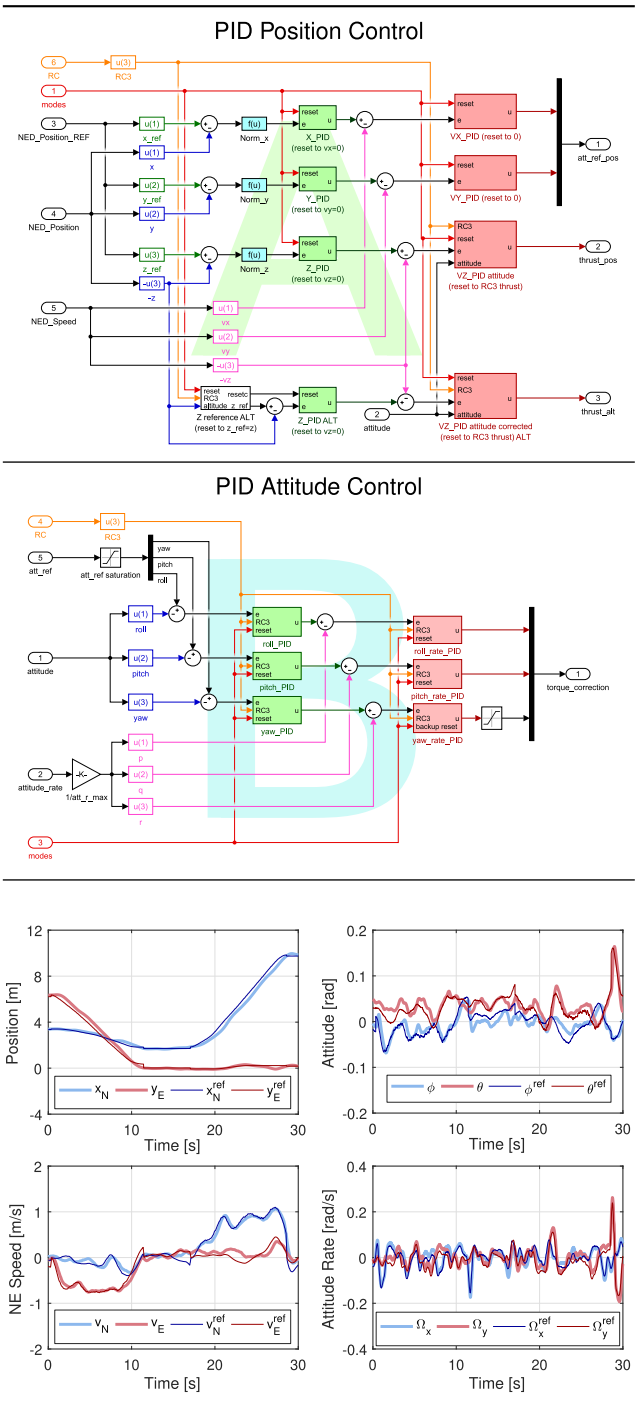


FIGURE 6. Cascade PID controller implemented in MAS$^{\boxplus}_{PX4}$ with results from the position and attitude control loops at constant altitude.

$\bar{Z}_1 = \Theta_{ref} - \Theta - \epsilon$ the filtered attitude error; $\epsilon \in \mathbb{R}^3$ a filter to counteract the mismatch of the tracking error; $Z_2 = \Omega_{ref} - \Omega$ the attitude rate error, and Γ_1, Γ_2 and $\bar{\Gamma} \in \mathbb{R}^{3\times3}$ are control and filter matrix gains. Notice that the gyroscopic terms have been neglected because of their irrelevance for this platform, and that the high-order implementation of this solution includes 9 nonlinear integrators.

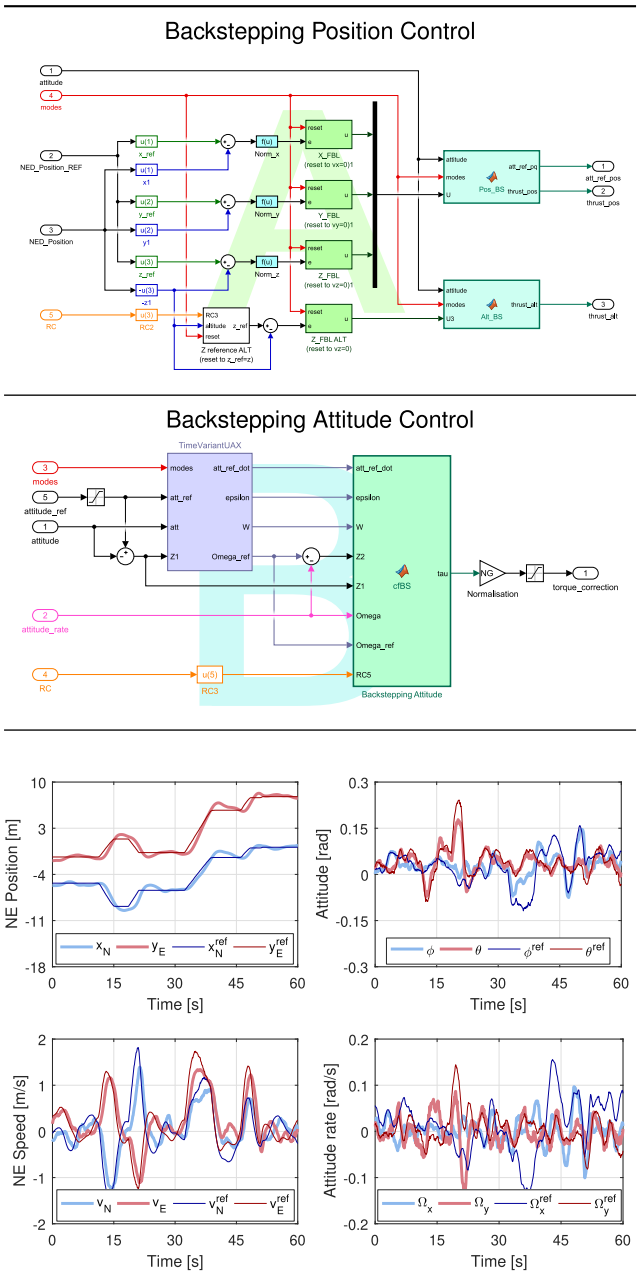


FIGURE 7. Command-filtered backstepping controller implemented in MAS$^{\boxplus}_{\text{PX4}}$ with results for position and attitude at constant altitude.

In Fig. 7 we depict again the Simulink® implementation of this much more involved nonlinear controller for the subsystems (top), together with the results at the bottom when commanded to track a variable reference. It is worth noting in this case that the attitude controller steady-state error does not affect the position controller.

Additionally, a CPU load test for this algorithm –in which aggressive RC commands and *ad-hoc* parameter tuning were induced in its first half to estimate their maximum impact– is shown in Fig. 8, becoming clear that the Intel® NUC is capable of running significantly more demanding algorithms even without turning to the accelerator mode of Simulink®,

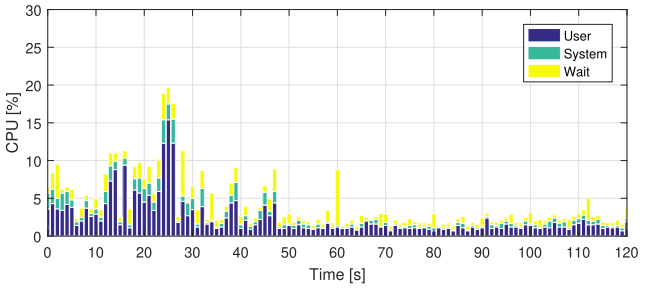


FIGURE 8. Stacked CPU load executing command-filtered backstepping controller in MAS$^{\boxplus}_{\text{PX4}}$, being the RAM use consistently steady at 25%.

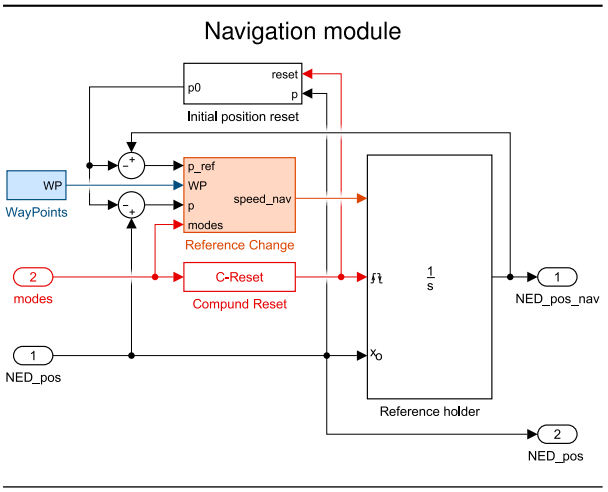


FIGURE 9. Navigation module implemented in MAS$^{\boxplus}_{\text{PX4}}$.

as expected. It is also worth noting that the variations due to the aforementioned additional elements in the first half of the experiment are perceptible. Nonetheless, considering the time ratio analysis in Fig. 4, in which significant computational power margins were detected, we can conclude that their impact on the reliability of the solution is negligible.

C. EV3: AUTONOMOUS NAVIGATION MODULE

Finally, a separate autonomous flight has been carried out in an outdoor scenario with wind disturbances and GPS inaccuracies including an additional navigation module depicted (see Fig. 9). The flight path, as depicted on the left side of Fig. 10, consists of an XY square followed by a vertical rectangle sharing one of its sides, being this manoeuvre chosen to show the response of the controller to changes in all directions. To highlight the changing speed during the flight, a set of time-equispaced points have been added to the position plot. It can be concluded that the solution is perfectly capable of tracking the proposed trajectory with a good overall performance. As shown by both path tracking and position subplots, the most important error peaks correspond to the regions around the waypoints, as expected, being the displayed overshoots at their typical range in aerial robotics.

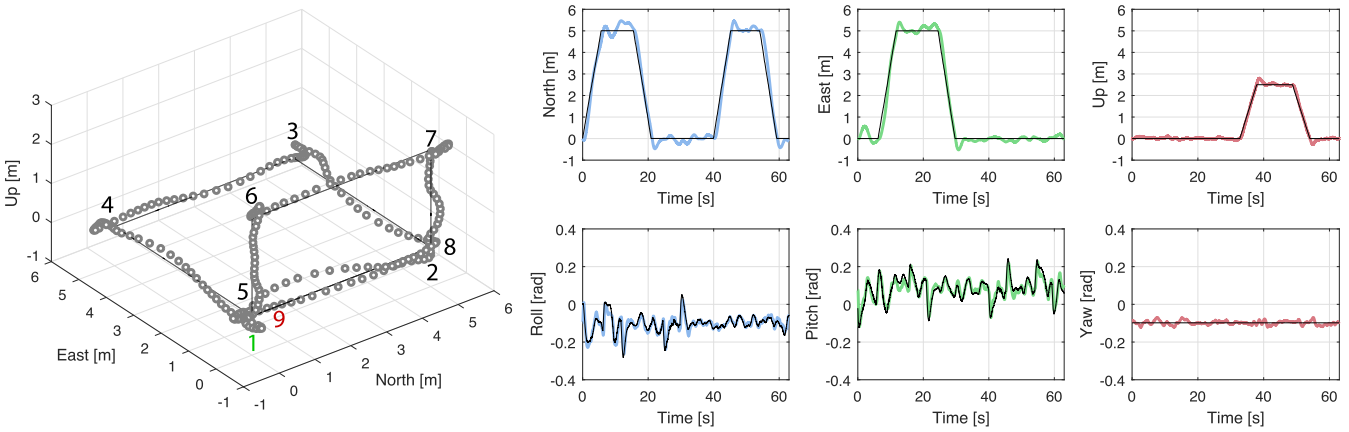


FIGURE 10. Monitoring of the UAV using a PID cascade controller: path tracking (on the left, with position markers separated by 0.25 s and navigation waypoint order –where the initial and final positions are highlighted in green and red, respectively–), global position (top right) and attitude (bottom right), where black lines represent the references (with speeds of up to 1 m/s) and the coloured lines/markers the UAV state.

IV. CONCLUSION

In the present work, a high-level modular autopilot solution for fast prototyping of custom algorithms in UAVs is presented and demonstrated in an experimental setting. The approach is proven to be safe and reliable in terms of communication, integration and execution of new algorithms due to its inherent redundancy. Moreover, the proposed solution provides a user-friendly modular environment in the well-known Simulink® software, making the theory-to-implementation step easier and faster, and facilitating the interchange and external validation of custom algorithms. Finally, its feasibility for computation-intensive methods is demonstrated by the implementation of a high-order nonlinear controller, thus overcoming the prototyping issues generally arisen when applying complex algorithms on aerial robotics.

Future work is underway to include additional modules like estimator and mixer for other multi-rotor setups. All these implementations and modules will be available in a shared open repository,[9] thus facilitating the aforementioned interchange and validation of custom functionalities.

APPENDIX A
Simulink® RUNNING MODES

Since the need for compilation in Simulink® is not evident, in here we include a disambiguation of the difference between *compilation* and *interpretation* in this graphical programming environment. According to the available technical documentation, the model is interpreted in Normal mode during each simulation run (see 10-11 in [50] and [51]), thus being this mode preferable when frequent changes of the model are needed. Moreover, including only Interpreted Function blocks –without using Math-Function ones– forces the execution engine to be called at each simulation time step, mimicking interpreters. Therefore, some kind of *compilation* –in the sense of Simulink®, i.e. the translation of the block diagram to an internal representation that interacts with the Simulink® engine (see 11-26 in [50])– is made in most applications, but it can never be comparable to a full standalone solution with an external compiler.

In what respects to the Accelerator mode [52], Simulink® uses by default the Just-In-Time (JIT) *compilation*. This generates an execution engine in memory for the top-level model only, and not for referenced models. As a result, a C compiler is not required during simulations. This is related to another helpful prototyping tool for the interpreter/compiler, the Model Reference. Wherever this Model-Reference block is used, the simulation mode of a sub-system can be controlled –i.e. by setting it to Accelerator mode, it is simulated through code generation; and by setting it to Normal mode this is done in interpreted mode–. Additionally, Accelerator mode does not support most runtime diagnostics of Normal mode, which is also very important during prototyping.

Finally, in Rapid Accelerator mode, Simulink® creates a standalone executable including both the solver and the models interacting with Simulink® via the external mode, with the subsequent lack of debugging options (see [53]).

In summary, the interpretation made in Normal mode (and in Accelerator mode via JIT) is not comparable to a full compilation obtaining a standalone executable or a binary file, as in Rapid Accelerator mode. The graphical programming, the tools for diagnostics and the possibility of a modular compilation (if desired with Model Reference) make this approach superior in prototyping stages. To provide a quantitative measure, we made a comparison in terms of time consumption. On average, our solution takes about 1-2 s to interpret a modified model (Normal mode), while the complete compilation of minimal changes in PX4 –our previous implementation approach– last about 20-25 s, 10 times slower in the least favourable case.

[9]https://github.com/grvcTeam/MASp

Algorithm 1 UDP Module – SEND Data

```
                                   ▷ code simplified for brevity
[· · · ]                           ▷ init daemon module in px4
thread_task(){
create_udp_socket()
subscribe_topic(topic_name)                          ▷ data source
while thread_is_running do
    if topic_update() then
        struct_data ← topic_data                     ▷ read topic content
        destination ← udp_send(struct_data)
    end if
end while
close_udp_socket()}
```

Algorithm 2 UDP Module – RECEIVE Data

```
                                   ▷ code simplified for brevity
[· · · ]                           ▷ init daemon module in px4
thread_task(){
create_udp_socket()
advert_topic(topic_name)
while thread_is_running do
    if recv_data() then
        struct_data ← data_received
        topic_name ← uorb_publish(struct_data)
    end if
end while
close_udp_socket()}
```

APPENDIX B
PX4-Simulink® UDP COMMUNICATIONS

As mentioned in the *Software Integration* subsection, the proposed solution is based on the UDP communications between PX4 and a Simulink® environment. This is implemented via two sockets, one after the ecl/EKF estimation and another one just before the input to the motor mixer. The core of their respective pseudocodes is shown in Algorithms 1 and 2.

Additionally, for those interested in the integration between the basic PX4 functionalities and the customisable $\text{MAS}^{\boxplus}_{\text{PX4}}$ environment, we refer readers to our GitHub repository at https://github.com/grvcTeam/MASp, which is currently under construction. Apart from the code support, a wiki of the Simulink® environment and the space to share custom modules with the $\text{MAS}^{\boxplus}_{\text{PX4}}$standard I/O connections will be included.

REFERENCES

[1] *Aerial Robotics Cooperative Assembly System*, ARCAS Project, Eur. Commission Under the Seventh Framework Programme. [Online]. Available: http://www.arcas-project.eu/

[2] A. Ollero, J. Cortes, A. Santamaria-Navarro, M. A. T. Soto, R. Balachandran, J. Andrade-Cetto, A. Rodriguez, G. Heredia, A. Franchi, G. Antonelli, K. Kondak, A. Sanfeliu, A. Viguria, J. R. Martinez-de-Dios, and F. Pierri, "The AEROARMS project: Aerial robots with advanced manipulation capabilities for inspection and maintenance," *IEEE Robot. Autom. Mag.*, vol. 25, no. 4, pp. 12–23, Dec. 2018.

[3] *Collaborative Aerial Robotic Workers*, AEROWORKS Project, Eur. Commission Under the H2020 Framework Programme. [Online]. Available: http://www.aeroworks2020.eu/

[4] *AERIAL Cognitive Integrated Multi-Task Robotic System With Extended Operation Range and Safety*, AERIAL-CORE Project, Eur. Commission Under the H2020 Framework Programme. [Online]. Available: https://aerial-core.eu/

[5] F. Ruggiero, V. Lippiello, and A. Ollero, "Aerial manipulation: A literature review," *IEEE Robot. Autom. Lett.*, vol. 3, no. 3, pp. 1957–1964, Jul. 2018.

[6] X. Meng, Y. He, and J. Han, "Survey on aerial manipulator: System, modeling, and control," *Robotica*, vol. 38, no. 7, pp. 1288–1317, Jul. 2020.

[7] A. Ollero and B. Siciliano, *Aerial Robotic Manipulation* (Springer Tracts in Advanced Robotics). Cham, Switzerland: Springer, Jul. 2019.

[8] N. Michael, J. Fink, and V. Kumar, "Cooperative manipulation and transportation with aerial robots," *Auton. Robots. Special Issue, Robot., Sci. Syst.*, vol. 30, no. 1, pp. 73–86, 2011.

[9] C. Korpela, M. Orsag, T. Danko, B. Kobe, C. McNeil, R. Pisch, and P. Oh, "Flight stability in aerial redundant manipulators," in *Proc. IEEE Int. Conf. Robot. Automat.*, May 2012, pp. 3529–3530.

[10] A. E. Jimenez-Cano, J. Martin, G. Heredia, A. Ollero, and R. Cano, "Control of an aerial robot with multi-link arm for assembly tasks," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, May 2013, pp. 4916–4921.

[11] A. Suarez, F. Real, V. M. Vega, G. Heredia, A. Rodriguez-Castano, and A. Ollero, "Compliant bimanual aerial manipulation: Standard and long reach configurations," *IEEE Access*, vol. 8, pp. 88844–88865, 2020.

[12] J. Á. Acosta, C. R. de Cos, and A. Ollero, "Accurate control of aerial manipulators outdoors. A reliable and self-coordinated nonlinear approach," *Aerosp. Sci. Technol.*, vol. 99, Apr. 2020, Art. no. 105731.

[13] J. A. Acosta, C. R. de Cos, and A. Ollero, "A robust decentralised strategy for multi-task control of unmanned aerial systems. Application on underactuated aerial manipulator," in *Proc. Int. Conf. Unmanned Aircr. Syst. (ICUAS)*, Jun. 2016, pp. 1075–1084.

[14] Y. E. Tlatelpa-Osorio, J. J. Corona-Sanchez, and H. Rodriguez-Cortes, "Quadrotor control based on an estimator of external forces and moments," in *Proc. Int. Conf. Unmanned Aircr. Syst. (ICUAS)*, Jun. 2016, pp. 957–963.

[15] L. Meier, D. Honegger, and M. Pollefeys, "PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, May 2015, pp. 6235–6240.

[16] (May 2019). *Dronecode Project PX4 Open Source Flight Control Software*. [Online]. Available: http://px4.io

[17] J. R. Guadarrama-Olvera, J. J. Corona-Sánchez, and H. Rodríguez-Cortés, "Hard real-time implementation of a nonlinear controller for the quadrotor helicopter," *J. Intell. Robotic Syst.*, vol. 73, nos. 1–4, pp. 81–97, Jan. 2014.

[18] M. Bangura and R. Mahony, "Real-time model predictive control for quadrotors," *IFAC Proc. Volumes*, vol. 47, no. 3, pp. 11773–11780, 2014.

[19] L. Xu, H. Ma, D. Guo, A. Xie, and D. Song, "Backstepping sliding-mode and cascade active disturbance rejection control for a quadrotor UAV," *IEEE/ASME Trans. Mechatronics*, early access, Apr. 27, 2020, doi: 10.1109/TMECH.2020.2990582.

[20] B. Fuller, J. Kok, N. A. Kelson, and L. F. Gonzalez, "Hardware design and implementation of a MAVLink interface for an FPGA-based autonomous UAV flight control system," in *Proc. Australas. Conf. Robot. Automat. (ACRA)*, Melbourne, VIC, Australia, Dec. 2014, pp. 1–6.

[21] Q. Lin, Z. Cai, and Y. Wang, "Design, model and attitude control of a model-scaled gyroplane," in *Proc. IEEE Chin. Guid., Navigat. Control Conf.*, Aug. 2014, pp. 1282–1287.

[22] J. R. Kutia, K. A. Stol, and W. Xu, "Aerial manipulator interactions with trees for canopy sampling," *IEEE/ASME Trans. Mechatronics*, vol. 23, no. 4, pp. 1740–1749, Aug. 2018.

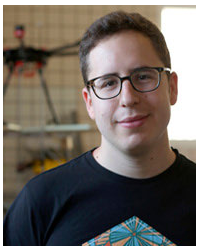
[23] F. Ruggiero, M. A. Trujillo, R. Cano, H. Ascorbe, A. Viguria, C. Perez, V. Lippiello, A. Ollero, and B. Siciliano, "A multilayer control for multirotor UAVs equipped with a servo robot arm," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, May 2015, pp. 4014–4020.

[24] D. Santamaría, F. Alarcón, A. Jiménez, A. Viguria, M. Béjar, and A. Ollero, "Model-based design, development and validation for UAS critical software," *J. Intell. Robotic Syst.*, vol. 65, nos. 1–4, pp. 103–114, Jan. 2012.

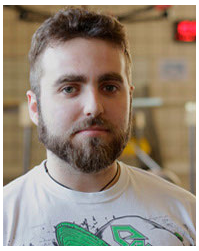
[25] S. Omari, M.-D. Hua, G. Ducard, and T. Hamel, "Hardware and software architecture for nonlinear control of multirotor helicopters," *IEEE/ASME Trans. Mechatronics*, vol. 18, no. 6, pp. 1724–1736, Dec. 2013.

[26] M. Tognon and A. Franchi, "Dynamics, control, and estimation for aerial robots tethered by cables or bars," *IEEE Trans. Robot.*, vol. 33, no. 4, pp. 834–845, Aug. 2017.

[27] M. Jafarinasab, S. Sirouspour, and E. Dyer, "Model-based motion control of a robotic manipulator with a flying multirotor base," *IEEE/ASME Trans. Mechatronics*, vol. 24, no. 5, pp. 2328–2340, Oct. 2019.
[28] Y. S. Lee, B. Jo, and S. Han, "A light-weight rapid control prototyping system based on open source hardware," *IEEE Access*, vol. 5, pp. 11118–11130, 2017.
[29] D. Hercog and K. Jezernik, "Rapid control prototyping using MATLAB/simulink and a DSP-based motor controller," *Int. J. Eng. Educ.*, vol. 21, no. 4, p. 596, 2005.
[30] R. Grepl, "Real-time control prototyping in MATLAB/simulink: Review of tools for research and education in mechatronics," in *Proc. IEEE Int. Conf. Mechatronics*, Apr. 2011, pp. 881–886.
[31] S. Rebeschieß, "MIRCOS-microcontroller-based real time control system toolbox for use with MATLAB/simulink," in *Proc. IEEE Int. Symp. Comput. Aided Control Syst. Design*, Aug. 1999, pp. 267–272.
[32] R. Bucher and S. Balemi, "Rapid controller prototyping with MATLAB/simulink and linux," *Control Eng. Pract.*, vol. 14, no. 2, pp. 185–192, Feb. 2006.
[33] S. Grzegorz, Z. Tomasz, and B. Andrzej, "Rapid control prototyping with scilab/scicos/RTAI for PC-based ARM-based platforms," in *Proc. Int. Multiconference Comput. Sci. Inf. Technol.*, Oct. 2008, pp. 739–744.
[34] Y.-S. Lee, J.-H. Yang, S.-Y. Kim, W.-S. Kim, and O.-K. Kwon, "Development of a rapid control prototyping system based on MATLAB and USB DAQ boards," *J. Inst. Control, Robot. Syst.*, vol. 18, no. 10, pp. 912–920, Oct. 2012.
[35] (May 2019). *MathWorks Simulink*. [Online]. Available: https://www.mathworks.com/products/simulink.html
[36] A. Koubaa, A. Allouch, M. Alajlan, Y. Javed, A. Belghith, and M. Khalgui, "Micro air vehicle link (MAVlink) in a nutshell: A survey," *IEEE Access*, vol. 7, pp. 87658–87680, 2019.
[37] (May 2019). *Intel Compute Card*. [Online]. Available: https://www.intel.com/content/www/us/en/products/boards-kits/compute-card
[38] (May 2019). *KHADAS Vim Boards*. [Online]. Available: https://www.khadas.com/vim
[39] L. Sha, "Using simplicity to control complexity," *IEEE Softw.*, vol. 18, no. 4, pp. 20–28, Jul. 2001.
[40] (May 2019). *Raspberry Pi 3 Model B*. [Online]. Available: https://www.raspberrypi.org/
[41] (May 2019). *Emlid NAVIO2*. [Online]. Available: https://emlid.com/navio/
[42] (May 2019). *Intel NUC Boards and Kits*. [Online]. Available: https://www.intel.com/content/www/us/en/products/boards-kits/nuc.html
[43] (May 2020). *2019 Dronecode Project*. [Online]. Available: https://www.dronecode.org/
[44] (May 2019). *Pixhawk—The Hardware Standard for Open Source Autopilots*. [Online]. Available: https://pixhawk.org/
[45] J. Jackson, G. Ellingson, and T. McLain, "ROSflight: A lightweight, inexpensive MAV research and development tool," in *Proc. Int. Conf. Unmanned Aircr. Syst. (ICUAS)*, Jun. 2016, pp. 758–762.
[46] (Oct. 2019). *iPerf—iPerf3 and iPerf2 User Documentation*. [Online]. Available: https://iperf.fr/iperf-doc.php#tuningudp
[47] (May 2019). *DJI Flame Wheel ARKF Kit F550*. [Online]. Available: https://www.dji.com/en/flame-wheel-arf
[48] C. R. de Cos, J. A. Acosta, and A. Ollero, "Command-filtered backstepping redesign for aerial manipulators under aerodynamic and operational disturbances," in *Proc. 3rd Iberians Conf. ROBOT*, Nov. 2017, pp. 817–828.
[49] Z. Zuo, "Trajectory tracking control design with command-filtered compensation for a quadrotor trajectory tracking control design with command-filtered compensation for a quadrotoEr," *IET Control Theory Appl.*, vol. 4, no. 11, pp. 2343–2355, 2010.
[50] (Nov. 2020). *Simulink Reference*. [Online]. Available: https://www.mathworks.com/help/pdf_doc/simulink/simulink_ref.pdf
[51] (Nov. 2020). *Improving Simulation Performance in Simulink*. [Online]. Available: https://es.mathworks.com/company/newsletters/articles/improving-simulation-performance-in-simulink.html
[52] (Nov. 2020). *How Acceleration Modes Work*. [Online]. Available: https://www.mathworks.com/help/simulink/ug/how-the-acceleration-modes-work.html
[53] (Nov. 2020). *Choosing a Simulation Mode*. [Online]. Available: https://www.mathworks.com/help/simulink/ug/choosing-a-simulation-mode.html



CARLOS RODRÍGUEZ DE COS (Member, IEEE) received the B.Sc. and M.Sc. degrees in aerospace engineering from the University of Seville, Seville, Spain, in 2015, where he is currently pursuing the Ph.D. degree with the GRVC Robotics Laboratory. He has been a Visitor with the I3S UNSA-CNRS, University of Nice Sophia-Antipolis, France. His research interests include aerial robotics, nonlinear control, adaptive control, compliant robotics, and state estimation.



MANUEL J. FERNANDEZ received the B.Sc. degree in telecommunications engineering from the University of Seville, Seville, Spain, in 2020, where he is currently pursuing the M.Sc. degree. His research interests include aerial robotics, communication security, swarming, blockchain technologies, and robot manipulation.

PEDRO J. SANCHEZ-CUEVAS (Member, IEEE) received the B.Sc. and M.Sc. degrees in aerospace engineering from the University of Seville, Seville, Spain, in 2014 and 2018, respectively, where he is currently pursuing the Ph.D. degree with the GRVC Robotics Laboratory. He was part of the Iberian Robotics Team that won the third challenge of the MBZIRC 2020. His research interests include aerial robotics, mechatronics, and inspection robotics.

JOSÉ ÁNGEL ACOSTA received the degree in servo-electrical and mechanical engineering from the University of Huelva, the degree in electrical engineering degree from the University of Seville, and the Ph.D. degree from the University of Seville, in 2004. Since 1999, he has been part of the Department of Systems Engineering and Control, where he is currently a Professor. He has been a Marie Curie Control Training Site Fellow in Supélec, CNRS, France, in 2003 and 2005, and a Visitor, since 2005. From 2008 to 2011, he was an Academic Visitor Researching with the Imperial College, London. His research interests include nonlinear control of dynamical systems with emphasis on electro-mechanical, aerospace and robotic systems. He received the European Award, in 2005, and was nominated to George S. Axelby Outstanding Paper Award in IEEE Transactions on Automatic Control, in 2006.

ANIBAL OLLERO (Fellow, IEEE) received the Engineer and Ph.D. Engineer degrees from the University of Seville, Seville, Spain. He has been a Full Professor with the Universities of Santiago de Compostela, Malaga and Sevilla and stagiere with the LAAS-CNRS, since 1979. From 1990 to 1991, he was a Visiting Scientist with Carnegie Mellon University. He is currently leading the GRVC Robotics Laboratory with more than 80 members with the University of Seville. He has lead or participated in more than 157 research and development projects, including 29 projects funded by the European Commission and other projects funded by NASA. Since December 2019, he has been the Coordinator of the H2020 Project AERIAL-CORE. He has been the Principal Investigator in more than 50 contracts with industries, including AIRBUS Group and Boeing R&T Europe. He received an ERC Advanced Grant from the European Research for the GRIFFIN project, in 2018.