Efficient Development of Model-Based Controllers in PX4 Firmware: A Template-Based Customization Approach

Simone D'Angelo*, Francesca Pagano*, Francesco Longobardi, Fabio Ruggiero, Vincenzo Lippiello

***Abstract*— This paper introduces a refined iteration of the PX4 autopilot firmware tailored to support developers in integrating bespoke control algorithms alongside the existing control framework. The proposed methodology employs a template-driven approach and introduces two novel control modules, thereby enabling users to harness all firmware functionalities within their custom modalities, including the QGroundControl interface, while retaining all the standard modules and compatibility with the QGroundControl interface. With its transparent and adaptable structure, the software framework presented herein lays a robust groundwork for implementing tailored and specialized solutions across diverse aerospace domains. As a practical demonstration, we apply the developed firmware to the domain of inspection and maintenance, wherein it incorporates an admittance controller and a model-based control algorithm for a tiltable drone equipped with a sensorized tool. The efficacy and versatility of the proposed approach are validated through simulations and empirical trials conducted across multiple aerial platforms. The produced code is released to the community.**

I. INTRODUCTION

Over the past decade, Unmanned Aerial Vehicles (UAVs), commonly known as drones, have witnessed escalating adoption in both academic research and industrial applications. These aerial platforms have demonstrated versatility across diverse domains, encompassing agriculture [1], aerial photography, and infrastructure inspection and maintenance [2]–[4]. They offer cost-effective and inherently safe solutions for tasks such as remote sensing, surveillance, payload transportation, non-destructive testing [5], and device installation [6]. However, the term UAV encompasses a broad spectrum of aerial platforms characterized by varying capabilities, levels of autonomy, and aerodynamic configurations. Understanding the intricacies of airframe structures becomes imperative, particularly when implementing model-based control laws, as control design is strictly linked to physical configuration.

Due to the heterogeneity of UAVs, open-source autopilot software provide versatile control solutions that can be readily deployed on various platforms, albeit with a trade-off in flight performances. Prominent examples include the PX4 and Ardupilot control frameworks, which support several unmanned aerial vehicles, ground rovers, and underwater robots [7], [8]. Other flight controllers, such as iNAV [9] and Betaflight [10], offer similar versatility, while dRehmFlight [11] caters primarily to hobbyist applications and is mainly suited for rapid prototyping. Despite their adaptability, existing open-source solutions do not currently accommodate advanced configurations such as omnidirectional tilting drones [12] or aerial manipulators [13], [14], which are prevalent in aerial robotics research and increasingly deployed in industrial settings. Private companies often resort to proprietary solutions [15]. At the same time, research laboratories frequently rely on custom control architectures that may lack features necessary for field deployment and real-world testing or see limited adoption beyond research settings [16]. Developing robust frameworks is inherently complex, as they must meet real-time requirements, ensure safety, hardware compatibility, provide API interfaces, and offer code support. Consequently, many efforts focus on customizing open-source autopilots, leveraging their modular structure and functionalities. However, such developments often target specific platforms and control laws, with limited code reuse. As autonomous systems advance, the management of software complexity becomes paramount.

This paper aims to address this gap by introducing a tool for easily implementing custom embedded control laws within the open-source PX4 firmware[1] while fully utilizing its functionalities and retaining standard control modules. The PX4 firmware was chosen to be customized for some key advantages. This leading autopilot software has a modular structure compatible with various hardware

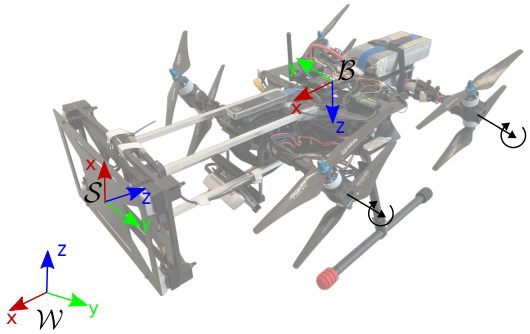

Fig. 1: H-shaped one-axis tilting drone equipped with a sensorized tool. Four load cells on the plate measure the forces and torques exchanged during the interaction. The main frames are depicted.

The research leading to these results has been supported by the COWBOT project, in the frame of the PRIN 2020 research program, grant n. 2020NH7EAZ_002, the AI-DROW project, in the frame of the PRIN 2022 research program, grant n. 2022BYSBYX, funded by the European Union Next-Generation EU, and the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie (grant agreement n. 953454). The authors are solely responsible for its content.

* Simone D'Angelo and Francesca Pagano are both first authors.

The authors are with CREATE Consortium and PRISMA Lab, Department of Engineering and Information Technology, University of Naples Federico II, Via Claudio 21, Naples, 80125, Italy. Corresponding authors' emails: simone.dangelo@unina.it, francesca.pagano@unina.it.

[1]When writing, the latest stable release is v1.14.

boards, such as the Pixhawk [7], and has a large support community. Furthermore, PX4 operates under a BSD license, enabling code modifications without mandatory submission to the main branch, thus safeguarding companies' intellectual property; in contrast, Ardupilot functions under the GPL license [8]. Therefore the proposed customization can help users and researchers implement embedded and model-based controllers to improve performance and ease the deployment of non-standard aerial systems in real-world applications. The produced code is freely available to the community [2].

The paper is organized as follows: Section II discusses the related works; the firmware customization is described in Section III while Section IV introduces some usage examples. The experimental part is divided between simulation studies in *ROS*/*Gazebo* and real-world experiments, reported in Sections V and VI respectively. Finally, Section VII concludes and analyzes future directions and functionalities to be developed.

II. RELATED WORKS

The PX4 autopilot was initially introduced in [17] as a node-based multi-threaded framework tailored for embedded platforms. Over nine years, this software has been extensively utilized in thousands of UAV-related studies. However, directly interfacing with the PX4 firmware can present challenges, leading many researchers to seek alternative approaches. Frequently, researchers opt to develop firmware from scratch, as seen in [18], [19], or execute the drone's controller on a companion computer. This latter approach is common especially when controlling hybrid or non-standard systems, such as an omnidirectional aerial manipulator [20]. While such strategies may streamline development efforts, they entail losing the comprehensive and thoroughly tested functionalities already implemented within the PX4 ecosystem. Choosing this route deprives researchers and developers of the capabilities provided by the PX4 disconnecting them from the potential growth of a dynamic and engaged community.

An exemplary illustration of PX4 utilization in research is demonstrated in [21], where an advanced PX4-based framework is introduced to execute complex missions, showcasing its efficacy through real-world outdoor experiments. However, this work predominantly deals with standard platforms, leveraging the standard embedded controller and relying on the companion computer for additional control layers.

While the PX4 modular structure allows the creation of new controller modules within the firmware, coding these modules can be time-consuming. Often, the standard proportional-integral-derivative (PID) controller is the preferred choice. Consequently, several studies focus on offline PID optimal gain tuning within PX4, both with [22] and without system identification [23], to enhance performance and robustness for specific user cases. Conversely, the concept of harnessing the PX4 flight stack and implementing a custom embedded controller has been explored in various ways in the literature. For instance, in [24], a disturbance-observer-based nonlinear sliding mode surface controller is validated using PX4-based simulation, indicating improved performance, although real-world experiments are not conducted. In [25], the authors devise a model predictive controller to regulate the angular rates of a quadcopter and integrate it into the PX4 Autopilot. Initially simulated in MATLAB, the controller is then converted to C++. The authors integrate the new controller by modifying the Attitude Rate Controller of PX4, thereby excluding the existing one. Similarly, in [26], a similar result is achieved by implementing a model reference adaptive control. The controller is designed for seamless integration into the PX4 firmware, allowing users to continue using the standard flight controller as before. However, specific implementation details are not provided. The obtained results demonstrate improved tracking performance compared to the PX4 controller. In [27], a controller implementing a geometric controller and an admittance control law in the PX4 flight stack is developed entirely in MATLAB using the UAV Toolbox Support Package for PX4 Autopilots [28], which enables modification of the PX4 flight controller. However, this approach overwrites the standard firmware, implying the loss of some software functionalities during task execution. Both [29] and [30] focus on fully actuated UAVs. In [29], the authors outline a method to extend existing controllers for under-actuated UAVs to support fully actuated ones. In [30], the PX4 firmware is extended to address the control allocation problem for UAVs with tiltable rotors. This is achieved by modifying specific firmware modules and leveraging the PX4 dynamic allocation while maintaining and exploiting the standard PX4 control modules.

In this work, the modifications made to the Control Allocation module and the simulation tools presented in [30] served as a foundational framework for integrating the capability to control tilting platforms into the proposed firmware.

A. Contributions

The primary objective of this paper is to augment the capabilities of the PX4 firmware, thereby enabling researchers and developers to design and deploy their own UAV control algorithms while leveraging the functionalities delineated in Section I.

Flexibility is lacking in most of the analyzed works: researchers predominantly embark on controller implementations from scratch, often disregarding pre-existing features. Typically, these endeavors entail replacing the reliable PX4 controller with a newly proposed one. However, excluding the existing modules means losing the reliability of the standard control stack. Moreover, the prevailing state-of-the-art solutions frequently overlook compatibility with new PX4 firmware versions. Besides, in the event of issues with the newly introduced controller, the standard modules can act as a reliable fallback option.

[2] https://github.com/prisma-lab/Px4_Tilting_Custom_Control
https://github.com/prisma-lab/qgroundcontrol

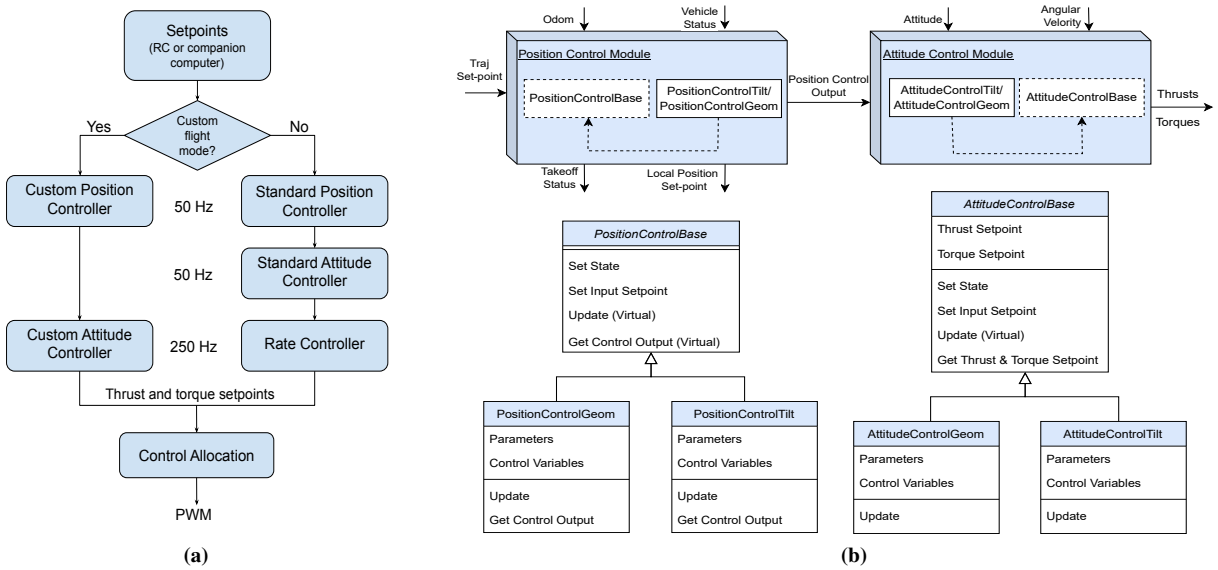

Fig. 2: (a) Custom PX4 firmware flow chart: on the left branch, the new modules implementing the model-based control law; on the right one, the original PX4 flight control stack. (b) Class inheritance scheme for modular development of model-based controllers.

The proposed modular implementation of control laws aims to facilitate the seamless integration of modifications with future firmware releases.

The project aims to establish a framework for implementing new controllers, guided by the following design principles:

- minimal impact on the original code for seamless integration with future releases of PX4;
- isolation of all custom code within newly created modules to prevent interference with other firmware components;
- enabling the ability to switch seamlessly between custom and standard controllers during flight operations;
- testing with different usage examples provided in the released firmware.

To evaluate the feasibility of these principles, new controllers will be implemented in custom flight modes for standard drones and tilting platforms. Tests conducted with simulated and real platforms will demonstrate the seamless transition between the original and custom control modalities.

The primary contribution of this work is the release of a custom firmware for PX4, available on GitHub for utilization and customization by interested researchers seeking to deploy it on their drones or evaluate controller performance in simulated environments. The custom firmware incorporates two distinct model-based controllers—i.e. a geometric tracking controller [31] and an omnidirectional control approach [5]—already tested in simulation and on real hardware. The proposed framework comprises two key components: the first houses all PX4-specific code for integrating the new controller firmware, and the second serves as a template. The template, constructed using C++ inheritance, will facilitate users in developing new control laws based on those provided in this work. Furthermore, an admittance filter is introduced alongside the implemented controllers to exemplify an initial practical application within the inspection and maintenance domain. This filter utilizes measurements from external force sensors to dynamically adjust desired trajectories enabling compliant or stiff interaction with the environment.

III. SOFTWARE ARCHITECTURE

The developed PX4 software architecture incorporates both the custom flight controller and the standard one. Illustrated in Fig. 2a, the firmware features two distinct control stacks: the original stack, comprising a cascade of P/PID controllers divided into three modules, and a custom controller divided into two modules. The structure of the custom control stack mirrors the classic inner and outer loop configuration commonly found in the literature, thereby providing a versatile and generalized framework. Following the hierarchical arrangement of the standard PX4 system, the custom controller modules are scheduled with varying priorities and frequencies. Specifically, the position outer loop operates at 50 Hz, while the inner attitude loop executes at 250 Hz. Both control stacks yield three-dimensional, normalized thrust, and torque vectors as outputs, which are then converted into pulse-width-modulation signals by the *Control Allocation* module. The normalization facilitates seamless integration with the existing mixing module, enabling effortless deployment on various UAV shapes already supported by PX4, including tilting platforms introduced in [30]. Moreover, the proposed structure is designed to be independent of the firmware's conventional behavior, thereby permitting switching between control stacks.

A. Flight modes

Flight modes[3] are a key feature of the PX4 autopilot activating different UAVs' behaviors and providing vary-

[3]https://docs.PX4.io/main/en/getting_started/flight_modes.html

ing levels of users' assistance. Flight modalities are also exploited in the presented work to manage the switching between the two control stacks.

Two new modalities are defined starting from the existing *position* and *offboard* flight modes: *PRISMAManual* and *PRISMAOffboard*. These flight modes enable the control of the UAV with the radio controller (RC) or the onboard computer while using the custom modules, respectively. The user can directly trigger the desired control stack only changing the flight modality. Most importantly, the switch to the custom flight modules excludes the standard controllers without disabling them, which is crucial for retaining full firmware functionality. In addition, such a feature allows easier and safer testing of custom modules, as users can still rely on the standard flight controller and switch to it whenever needed, even during flight.

B. Custom PX4 modules

The custom *Position* and *Attitude* modules are constructed following the conventional PX4 control module structure, and implemented by classes that inherit from:

- *ModuleBase* class for core module functionalities, such as 'start', 'stop', and 'status' commands;
- *ModuleParams* class for accessing the *param* subsystem, crucial for defining and accessing parameters for control;
- *ScheduledWorkItem* class for scheduling.

These parent classes are already implemented in the PX4 standard libraries.

These modules oversee all vital information required for the firmware's proper operation, gathering input from various uORB[4] topics and disseminating output used by other modules.

The specific controller is implemented in a separate class, providing a straightforward interface for defining multiple controllers without modifying the module core, as detailed in subsequent sections. The `run(·)` function serves as the core of the modules, executed at each iteration. Additionally, both modules feature a `parameters_update(·)` function to check any parameter variations and update their values accordingly.

The classes for the position and attitude controllers share a similar structure, illustrated in Fig. 2b. Therefore, we will use the former as a reference to describe both, highlighting key differences. A virtual base class named *PositionControlBase* (*AttitudeControlBase* for the inner loop) provides the necessary function prototypes for proper controller operation within the module. While some functions have a basic implementation that can be expanded in the derived class, others are purely virtual and require complete implementation. The derived classes include the actual control laws, with all the necessary member variables, such as the gains and the dynamic model parameters.

The proposed software structure facilitates the addition of a new controller through the following simple steps.

[4]https://docs.px4.io/main/en/middleware/uorb.html

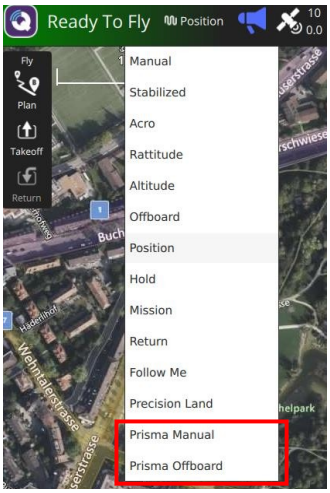

Fig. 3: QGC: custom flight mode *PRISMAManual* and *PRISMAOffboard* selection

1) Create new classes for the position and attitude controllers using the respective base ones as a reference and implement all necessary methods.
2) Define the messages exchanged from the position to the attitude controller.
3) Add macros for the new controller and set new parameters in the `parameters_update(·)` function, if applicable.

Preprocessor directives ensure flexibility in the message sent from the module to the attitude controller. Initially, a macro defines the selected controller type, with constants associated with each developed custom control law, enabling the compilation of the appropriate controller class and custom messages. Notably, it is possible to define multiple controllers and select at compile time the one to associate with custom flight modes.

C. User Interface

QGroundControl (QGC) is the default PX4 ground control station, offering comprehensive flight control and mission planning capabilities through a graphical user interface (GUI). With QGC, users can manage firmware settings, parameters, sensor calibration, and logs. Maintaining these key functionalities necessitates additional modifications to the PX4 firmware and the QGC source code.

Users can select the custom flight modes through the QGC slider (see Fig. 3) or the RC triggers, once properly configured via the ground station. The introduction of controller parameters, prefixed with *PRISMA* and added to the PX4 parameter server, enables their modification through the QGC interface. This functionality facilitates online tuning and updating of controller gains. Additionally, custom parameters can be leveraged to achieve tailored behaviors. For instance, in the case of an omnidirectional tilting drone, adjusting the *PRISMA_ANG_MODE* parameter allows users to map pitch, roll, or yaw rotation to the same RC stick, enabling manual command of desired attitude, as demonstrated in Section IV.

D. Companion Computer

A companion computer enhances the capabilities of an aerial system by providing additional flexibility and computational power onboard. PX4 flight controllers can be configured to communicate over a serial port with a companion computer through various application programming interfaces (APIs). In this paper, the ROS2 middleware handles the communication, efficiently exposing PX4 uORB messages as ROS2 topics. This integration allows PX4 to benefit from ROS2's robust middleware for high-level decision-making, complex maneuvers, and sensor integration. The ROS2 connection facilitates the custom control modules in retrieving Offboard commands, position feedback, and sensor measurements, thereby enhancing tracking capabilities and simplifying onboard sensor fusion in advanced applications. A detailed example will be presented in Section VI.

IV. Firmware usage examples

The subsequent sections present three firmware usage examples as case studies. Each example demonstrates the versatility of the proposed solution by implementing a diverse model-based control law deployed on a different aerial platform, as summarized in Table I. The implemented controllers are included in the released firmware and can be utilized in simulations and real-world experiments, as further detailed in Sections V–VI. For conciseness, not all tests will be included, but additional results are available in the accompanying video. A detailed discussion of the adopted control laws exceeds the scope of this paper; therefore, only their main characteristics and the reasons for their selection will be provided.

The main reference frames are depicted in Fig. 1. $\mathcal{B}$ is the UAV's body frame and $\mathcal{W}$ denotes the global world coordinate system. If a sensorized tool is provided, i.e. an end-effector equipped with an exteroceptive sensor, let $\mathcal{S}$ be the frame attached to the tool. Let $p, \eta \in \mathbb{R}^3$ be the position and attitude vector expressed with the roll, pitch and yaw (*rpy*) representation and $q \in \mathbb{R}^4$ the relative quaternion, $\omega \in \mathbb{R}^3$ be the angular rate vector, $u_T, \tau \in \mathbb{R}^3$ the thrust and torque vectors. In the following, $(\cdot)_d$ will indicate the desired quantities while the measured ones will appear without any subscript. Dynamical parameters are the mass $m > 0$ and the inertia matrix $J \in \mathbb{R}^{3\times3}$.

A. Case 1: Geometric tracking controller

A standard usage example is presented, focusing on standard flat UAVs. The geometric tracking controller emerges as a prominent solution for controlling these platforms, establishing a coupling between the outer and inner control loop wherein attitude set-points are generated by combining the desired yaw and the actual desired thrust. A notable feature of this approach lies in the computation of attitude error. Utilizing the rotation matrix allows the definition of errors directly in the special orthogonal group $SO(3)$, thus offering an approach devoid of singularities and circumventing complexities and ambiguities associated with other attitude representations such as Euler angles or quaternions.

Firmware Usage Examples		
#	Arial Platform	Model-based Control Law
1	Flat quadrotor	[32]
2	Omnidirectional tilting drone	[5]
3	H-shaped one-axis tilting drone with a sensorized tool.	[32] + Admittance Filter

TABLE I: Fimware usage examples

The theoretical framework adopted here aligns with that presented in [32], incorporating integral action in both thrust and torque vectors computation.

In the following, the attitude and rate errors are provided:

$$e_R(t) = \tfrac{1}{2}(R_d^T R - R^T R_d)^\vee, \quad e_\omega(t) = \omega - R^T R_d \omega_d, \tag{1}$$

where $(\cdot)^\vee$ is the vee operator [32]. This choice enables singularity-free handling, contributing to a smooth and continuous description of the system's spatial orientation.

B. Case 2: Omnidirectional controller

To extend the work in [30] another usage example is provided to control omnidirectional tilting platforms. The controller is extracted from [5] and can be divided into the following two parts.

1) Position control: This part considers the linear errors

$$e_p(t) = p(t) - p_d(t), \qquad e_v(t) = v(t) - \dot{p}_d(t), \tag{2}$$

$$e_i(t) = \int_0^t \big(e_v(\tau) + c_1\, e_p(\tau)\big) d\tau, \tag{3}$$

where $c_1 \in \mathbb{R}$ a proper gain; while the controller computes the 3D thrust vector as

$$u_T \;=\; R^T(K_P e_p + K_D e_v + K_I e_i + mge_3 + \ddot{p}_d). \tag{4}$$

2) Attitude control: This part computes the attitude error in the quaternion representation as

$$q_{err} = q_d \otimes q^* = \begin{pmatrix} q_{\omega,err} \\ q_{v,err} \end{pmatrix}, \tag{5}$$

where q^* is the inverse of the measured quaternion and $\otimes$ is the quaternion product operator [5].

A desired angular velocity is extracted from this error and then used to evaluate the desired torque vector:

$$\omega_d = K_q sign(q_{\omega,err}) q_{v,err} \tag{6}$$

$$\tau = K_r(\omega_d - \omega) - r_{off} \times u_T + \omega \times J\omega, \tag{7}$$

In (4) and (7) $K_{(\cdot)} \in \mathbb{R}^{3\times3}$ is proper control gain matrix, g is the gravity acceleration and $r_{off} \in \mathbb{R}^3$ is the offset from the center of mass.

C. Case 3 - Admittance control

A third example illustrates the firmware's applicability in the inspection and maintenance field by incorporating an indirect force control mechanism into the custom control stack. The implementation involves employing an admittance filter, described by

$$M_a \ddot{z}(t) + K_{d_a} \dot{z}(t) + K_{p_a} z(t) = f_{ext}(t), \tag{8}$$

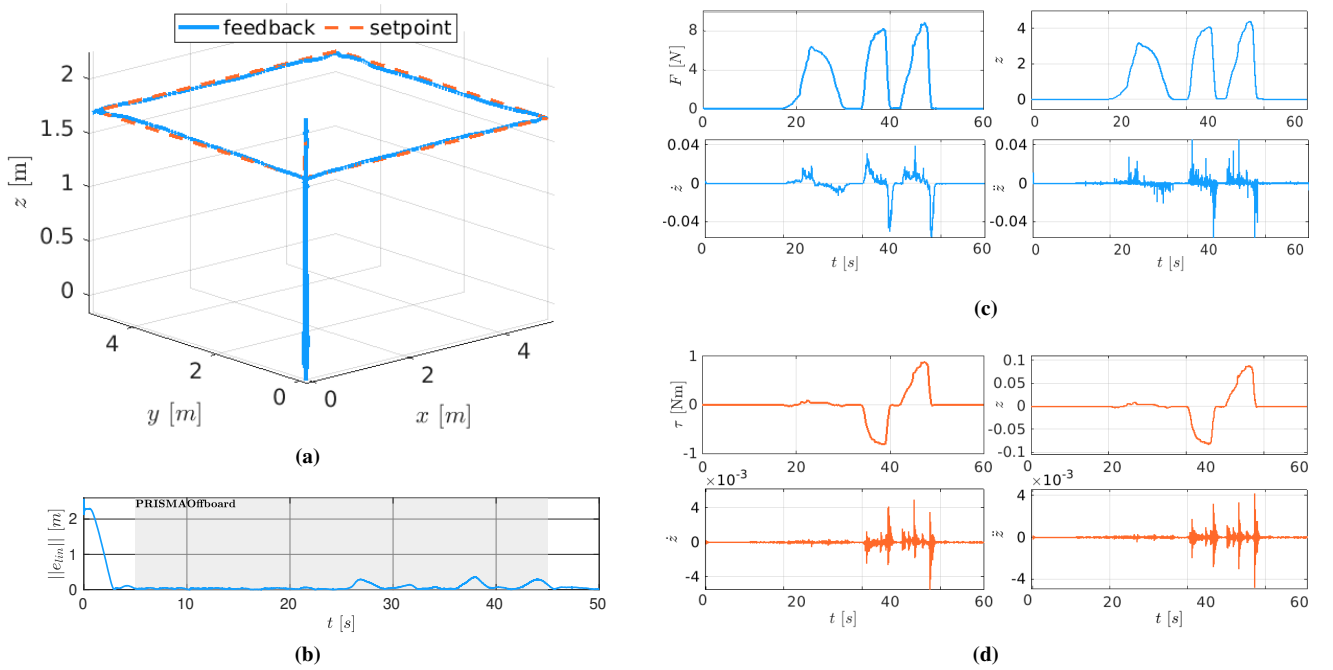

Fig. 4: Gazebo simulation studies - in the left column the position trajectory tracking results obtained with the fully actuated tiltable drone are depicted; in the right column the indirect force control results obtained with the H-shape one-tilt configuration appear. The plots represent respectively: (a)(b) the desired and measured 3D trajectory and the norm error, (c)(d) admittance filter correction computed starting both the interaction force (c) and torque measurements (d).

where M_a, K_{d_a}, and $K_{p_a} \in \mathbb{R}^{3\times3}$ denote the associated mass, damping, and stiffness matrices, respectively. Beginning with the measured interaction force $f_{ext} \in \mathbb{R}^3$, the filter adjusts the drone's planned references, computing the corresponding corrections $z(t)$, $\dot{z}(t)$, $\ddot{z}(t) \in \mathbb{R}^3$ onboard. The acceleration can be derived by inverting (8), while $\dot{z}(t)$ and $z(t)$ are computed through successive time integration.

The same principle is extended to the computation of attitude set-point corrections from the interaction torques. Notably, the firmware implementation is closely tied to the custom control module, facilitating direct usage of the filter with every implemented controller class. Moreover, in the absence of contacts, the entire PX4 control algorithm functions as a pure motion-tracking controller.

V. SIMULATIONS

The custom firmware seamlessly integrates with the ROS/*Gazebo* PX4 simulation environment. The tilting platforms, utilized as case studies, are incorporated into the PX4 firmware simulation models, while the standard *iris* drone serves as the flat quadrotor benchmark. As in the standard firmware setup, the user can command the simulated drone via the QGC virtual joystick or with a planner in the offboard mode. This provides flexibility to test the custom control stack across various flight modes.

The conducted tests utilize a ROS2 planner, which publishes linear position, velocity, and acceleration references, along with yaw setpoints on the standard PX4 `trajectory_setpoint` topic. Additionally, the planner publishes a quaternion desired attitude on a custom topic named `tilting_attitude_setpoint`, specifically required by the custom controller for tilting UAVs. An interpolation mechanism guarantees the absence of discontinuities in the commands. The simulations for study case one are omitted for brevity but are shown in the attached video. Figures 4a-4b depict the results with the omnidirectional platform. The planned trajectory, shown in Fig. 4a in orange, contains four points forming a square; the drone can track the reference perfectly thanks to the tilting capabilities, avoiding modifying the attitude during the task execution. Moreover, Fig. 4b shows that the linear tracking error norm is close to zero. The shaded regions in the plots indicate the activation of the custom controller in the *PRISMAOffboard* flight mode, while the white areas represent standard modalities. Notably, during the transition phase, there are no discontinuities in the error norm. An additional test is conducted using the simulated one-axis tilting drone equipped with a stick sensorized with a force sensor (four load cells) and integrated into the control loop to emulate real-world behavior. Figures 4c to 4d depict the interaction force in the approaching direction and the interaction torque around the tool z-axis, utilized to compute relative set-point corrections in the PX4 firmware.

VI. REAL EXPERIMENTS

A. Experimental setup

Similar experiments are replicated by uploading the custom firmware on real platforms entirely customized by the authors. For further details, the readers can refer to [27] and [30] for the first two drones, while the third one appears in Fig. 1. This latter platform is an H-shaped, one-axis tilting

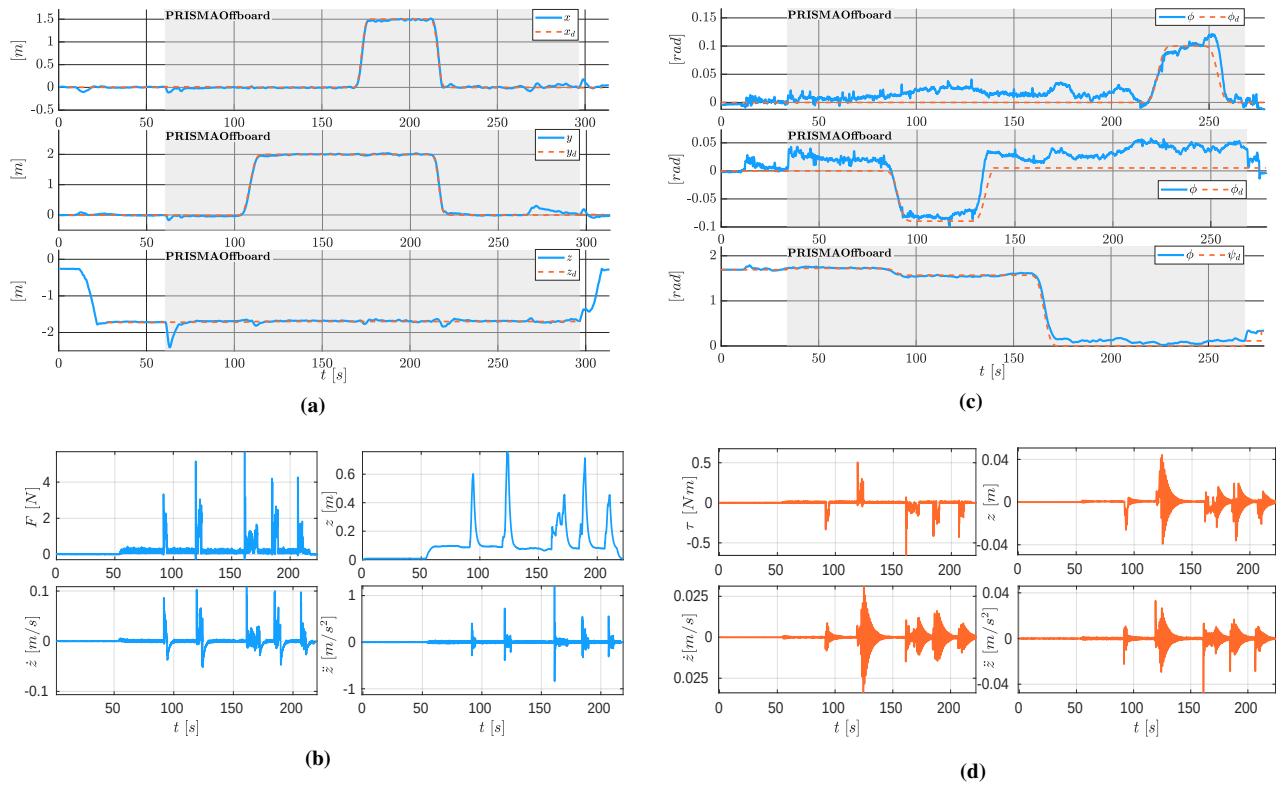

Fig. 5: Real-world case studies: on the top, the position and angular trajectory tracking results obtained with the fully actuated tiltable drone; on the bottom, the indirect force control results obtained with the H-shape one-tilt configuration. The plots represent three different tests: (a) Linear position tracking, (c) attitude tracking, (b)(d) admittance filter correction computed from both the interaction force (b) and torque measurements (d)

drone with arms controlled by coupled servos, enabling tilting around the body y-axis. A tool composed of four load cells is affixed to the front to measure the interaction force on the x-axis and the torque around its z-axis.

All experiments are performed in an indoor flight arena without any motion capture or 3D tracking systems. Position feedback is estimated using SLAM algorithms executed on the drone's onboard computer, leveraging input from two cameras: a combination of T265 and D435i RealSense cameras feed into an RTABMAP algorithm.

The custom firmware is deployed on both the stable releases v.1.13 and v.1.14, compiled, and executed on a Pixhawk 6C flight controller. Each drone utilizes a Lattepanda 3 Delta as a companion computer, operating Linux 22.04 with ROS and ROS2 docker images. The ROS2 Data Distribution Service (DDS) handles the communication with the board, while the communication with the ground-station PC relies on a Wi-Fi network and telemetry antennas.

For the third drone's sensorized tool, four load cells are connected to the autopilot via an Arduino Nano, which processes the measurements to compute the mean orthogonal interaction force and related torques around the body axis. The data is then reconstructed, converted into a force message, and published as a uORB topic for utilization in the admittance control law.

For the sake of brevity, only the experiments with the two tilting platforms will be presented, while the tests with the standard quadrotor are shown in the accompanying video. The motion controller dynamical parameters are selected from the CAD model of the drone, and in the case of the one-tilting axis drone are $m = 4$ kg, $J = \begin{bmatrix}0.29 & 0.5 & 0.55\end{bmatrix} I_3$ kgm^2. The controller gains are $K_P = \begin{bmatrix}10 & 10 & 15\end{bmatrix} I_3$, $K_D = \begin{bmatrix}4 & 4 & 7\end{bmatrix} I_3$, $K_I = \begin{bmatrix}1.28 & 1.28 & 2\end{bmatrix} I_3$, $K_R = \begin{bmatrix}1.8 & 1.8 & 0.65\end{bmatrix} I_3$, $K_\omega = \begin{bmatrix}0.3 & 0.3 & 0.1\end{bmatrix} I_3$, $K_{I_o} = \begin{bmatrix}0.1 & 0.1 & 0.05\end{bmatrix} I_3$, $M_a = 5I_3$, $K_{p_a} = 2I_3$, $K_{d_a} = 25I_3$.

B. Experimental validation

Figures 5a and 5c depict the experimental results obtained with the omnidirectional platform. Specifically, Fig. 5a illustrates the platform's capability to track the reference trajectory with minimal position error in a real-world scenario, facilitated by its four independent tilting servos. The plot also showcases the firmware's switching capability: takeoff and landing phases are executed by a human operator in the standard *Position* flight mode, while trajectory tracking occurs in the custom *PRISMAOffboard* mode, highlighted by the grey shaded areas in the figures. Figure 5c showcases the tilting platform's ability to maintain stable hovering while tracking a time-variant attitude reference. For safety reasons, integral action on the attitude control is deactivated, and small roll and pitch displacements are commanded.

An additional experiment demonstrates the one-axis tilting drone performing an inspection test, utilizing the sensorized tool to make contact with a surface. The task is semi-autonomous, with a human operator controlling the drone in

PRISMAManual flight mode. The operator lacks complete knowledge of the wall's position and orientation; once the contact occurs, the filter adjusts the drone's attitude and position to ensure interaction with all four load cells. The measured interaction force in the approaching direction and the interaction torque around the tool's z-axis are used to compute the relative set-points correction in the PX4 firmware (see Figs. 5b and 5d).

VII. CONCLUSIONS

This paper presents a modified iteration of the PX4 firmware, facilitating the integration of new controllers into a comprehensive autopilot system. Successfully tested in both simulated and indoor environments across various platforms, two integrated custom controllers demonstrate efficacy. Additionally, a notable application showcases the firmware's utilization in an aerial manipulator for nondestructive test inspections. This contribution unveils novel prospects for deploying the firmware in research and industrial settings, streamlining the implementation of embedded model-based and advanced control laws leveraging additional sensor information (e.g., force sensors). The autonomous execution of these tasks, devoid of external middleware dependencies like ROS or Matlab, holds promise for heightened performance and reduced memory overhead. Future endeavors will investigate further the safety implications of transitioning between control stacks. In addition, efforts will focus on keeping the custom firmware aligned with new versions of PX4, aiming to incorporate it as a stable release.

REFERENCES

[1] J. Kim, S. Kim, C. Ju, and H. I. Son, "Unmanned aerial vehicles in agriculture: A review of perspective of platform, control, and applications," *IEEE Access*, vol. 7, pp. 105 100–105 115, 2019.

[2] J. Cacace, A. Finzi, V. Lippiello, G. Loianno, and D. Sanzone, "Aerial service vehicles for industrial inspection: Task decomposition and plan execution," *Applied Intelligence*, vol. 42, no. 1, p. 49–62, jan 2015.

[3] G. Aiello, F. Hopps, D. Santisi, and M. Venticinque, "The employment of unmanned aerial vehicles for analyzing and mitigating disaster risks in industrial sites," *IEEE Transactions on Engineering Management*, vol. 67, no. 3, pp. 519–530, 2020.

[4] T. Mao, K. Huang, X. Zeng, L. Ren, C. Wang, S. Li, M. Zhang, and Y. Chen, "Development of power transmission line defects diagnosis system for uav inspection based on binocular depth imaging technology," in *2019 2nd International Conference on Electrical Materials and Power Equipment (ICEMPE)*, 2019, pp. 478–481.

[5] M. Kamel, S. Verling, O. Elkhatib, C. Sprecher, P. Wulkop, Z. Taylor, R. Siegwart, and I. Gilitschenski, "The voliro omniorientational hexacopter: An agile and maneuverable tiltable-rotor aerial vehicle," *IEEE Robotics & Automation Magazine*, vol. 25, no. 4, pp. 34–44, 2018.

[6] J. Cacace, S. M. Orozco-Soto, A. Suarez, A. Caballero, M. Orsag, S. Bogdan, G. Vasiljevic, E. Ebeid, J. A. A. Rodriguez, and A. Ollero, "Safe local aerial manipulation for the installation of devices on power lines: Aerial-core first year results and designs," *Applied Sciences*, vol. 11, no. 13, 2021.

[7] "PX4 Home Page," https://px4.io/.

[8] "Ardupilot Home Page," https://ardupilot.org/.

[9] "iNAV Home Page," https://github.com/iNavFlight/inav.

[10] "Betaflight Home Page," https://betaflight.com/.

[11] "dRehmFlight Home Page," https://github.com/nickrehm/dRehmFlight.

[12] M. Kamel, S. Verling, O. Elkhatib, C. Sprecher, P. Wulkop, Z. Taylor, R. Siegwart, and I. Gilitschenski, "The voliro omniorientational hexacopter: An agile and maneuverable tiltable-rotor aerial vehicle," *IEEE Robotics & Automation Magazine*, vol. 25, no. 4, p. 34–44, Dec. 2018.

[13] A. Ollero, M. Tognon, A. Suarez, D. Lee, and A. Franchi, "Past, present, and future of aerial robotic manipulators," *IEEE Transactions on Robotics*, vol. 38, no. 1, pp. 626–645, 2022.

[14] S. D'Angelo, A. Corrado, F. Ruggiero, J. Cacace, and V. Lippiello, "Stabilization and control on a pipe-rack of a wheeled mobile manipulator with a snake-like arm," *Robotics and Autonomous Systems*, vol. 171, p. 104554, 2024.

[15] "Voliro Home Page," https://voliro.com/.

[16] M. Foughali, F. Ingrand, and A. Mallet, "Genom3 templates: from middleware independence to formal models synthesis," *CoRR*, vol. abs/1807.10154, 2018. [Online]. Available: http://arxiv.org/abs/1807.10154

[17] L. Meier, D. Honegger, and M. Pollefeys, "Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 6235–6240.

[18] A. Berra, P. J. Sanchez-Cuevas, M. Trujillo, G. Heredia, and A. Viguria, "Airframe - fast prototyping framework for uavs definition," in *2023 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2023, pp. 1175–1182.

[19] M. Jovanovic and D. Starčević, "Software architecture for ground control station for unmanned aerial vehicle," 05 2008, pp. 284 – 288.

[20] M. Rubio, J. Näf, F. Bühlmann, P. Brigger, M. Hüsser, M. Inauen, N. Ospelt, D. Gisler, M. Tognon, and R. Siegwart, "Design of prismav: An omnidirectional aerial manipulator based on a 3-puu parallel mechanism," in *2023 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2023, pp. 608–615.

[21] T. Baca, M. Petrlik, M. Vrba, V. Spurny, R. Penicka, D. Hert, and M. Saska, "The MRS UAV system: Pushing the frontiers of reproducible research, real-world deployment, and education with autonomous unmanned aerial vehicles," *Journal of Intelligent & Robotic Systems*, vol. 102, no. 1, Apr. 2021.

[22] W. Saengphet, S. Tantrairatn, C. Thumthae, and J. Srisertpol, "Implementation of system identification and flight control system for uav," 04 2017, pp. 678–683.

[23] V. Gomez, N. Gomez, J. Rodas, E. Paiva, M. Saad, and R. Gregor, "Pareto optimal pid tuning for px4-based unmanned aerial vehicles by using a multi-objective particle swarm optimization algorithm," *Aerospace*, vol. 7, no. 6, 2020.

[24] Y. Jing, X. Wang, J. Heredia-Juesas, C. Fortner, C. Giacomo, R. Sipahi, and J. Martinez-Lorenzo, "Px4 simulation results of a quadcopter with a disturbance-observer-based and pso-optimized sliding mode surface controller," *Drones*, vol. 6, no. 9, 2022.

[25] C. A. Amadi, "Design and implementation of a model predictive control on a pixhawk flight controller." Ph.D. dissertation, Stellenbosch: Stellenbosch University, 2018.

[26] E. A. Niit and W. J. Smit, "Integration of model reference adaptive control (mrac) with px4 firmware for quadcopters," in *2017 24th International Conference on Mechatronics and Machine Vision in Practice (M2VIP)*, 2017, pp. 1–6.

[27] S. D'Angelo, F. Pagano, F. Ruggiero, and V. Lippiello, "Development of a control framework to autonomously install clip bird diverters on high-voltage lines," in *2023 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2023, pp. 377–382.

[28] MATLAB PX4 Toolbox, https://it.mathworks.com/help/supportpkg/px4/index.html.

[29] A. Keipour, M. Mousaei, A. T. Ashley, and S. Scherer, "Integration of fully-actuated multirotors into real-world applications," *arXiv preprint arXiv:2011.06666*, 2020.

[30] S. Marcellini, J. Cacace, and V. Lippiello, "A px4 integrated framework for modeling and controlling multicopters with til table rotors," in *2023 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2023, pp. 1089–1096.

[31] T. Lee, M. Leok, and N. H. McClamroch, "Geometric tracking control of a quadrotor uav on SE(3)," in *49th IEEE Conference on Decision and Control (CDC)*, 2010, pp. 5420–5425.

[32] F. Goodarzi, D. Lee, and T. Lee, "Geometric nonlinear pid control of a quadrotor uav on SE(3)," in *2013 European Control Conference (ECC)*, 2013, pp. 3845–3850.