DOI: 10.13009/EUCASS2025-628
11TH EUROPEAN CONFERENCE FOR AERONAUTICS AND SPACE SCIENCES (EUCASS)

Implementing Quaternion-Based Nonlinear Control for Quadrotors in the PX4 Autopilot Framework

Utku Yucel[⋆] *and Ozan Tekinalp*[⋆†]
[⋆]*Middle East Technical University*
Ankara, Turkiye
yucel.utku@metu.edu.tr · tekinalp@metu.edu.tr
[†]Corresponding author

Abstract

This study presents two workflows for integrating a previously developed Quaternion-Based Nonlinear Controller and a Linear Quadratic Tracking Velocity Controller (QNC-LQT) into the PX4 autopilot framework: (1) using the UAV Toolbox Support Package for PX4 Autopilots and (2) modifying the PX4 source code to add a custom flight mode. While the first enables rapid testing in Simulink, the second preserves core PX4 functionalities. Both approaches are validated through Software-In-The-Loop (SITL) tests and the source-code-based approach is further validated through real-world flight tests on a custom-built quadrotor using a Cube Black flight controller.

1. Introduction

The development of unmanned aerial vehicles (UAVs) created the potential for them to be used as essential platforms in fields such as logistics, surveillance, precision agriculture, and disaster response. The reliable operation of UAVs depends on robust control architectures. PX4 [8], an open-source autopilot software, provides a modular and extensible environment for UAV control. However, its default controllers may not meet the specific needs of all applications, especially in research contexts where custom control strategies are frequently proposed. Researchers often face challenges transitioning their algorithms from theory to practice due to the lack of standardized implementation workflows and the technical barrier posed by low-level autopilot development.

This manuscript addresses the lack of standardized workflows for implementing custom UAV controllers by exploring two integration approaches within the PX4 framework. The first involves using the UAV Toolbox Support Package for PX4 Autopilots [6] in Simulink, which offers a model-based development environment that simplifies design and rapid testing through SITL simulations. However, this approach disables essential PX4 features such as navigation, sensor calibration, and failsafe mechanisms, making it unsuitable for real-world deployment. The second approach directly modifies the PX4 source code to introduce a custom flight mode that activates a user-defined controller embedded as a new module. Both workflows are demonstrated using a previously published QNC-LQT controller [3] and are validated in SITL simulations. The source code modification approach is further extended to real-world testing, with the modified firmware deployed on a Cube Black flight controller mounted on a custom-built quadrotor. The focus of this study is not on benchmarking controller performance but on evaluating and documenting robust implementation strategies for real-world UAV applications.

The remainder of this paper is structured as follows: Section 2 reviews related work on PX4-based control development and existing approaches for custom controller integration. Section 3 provides an overview of the custom control architecture used in this study. Section 4 details the two implementation workflows: one using the UAV Toolbox for Simulink, and the other involving direct modification of the PX4 source code. Section 5 outlines the testing and validation strategy employed in both simulation and flight. Section 6 presents and analyzes the results obtained from SITL and real-world testing. Section 7 discusses the implications, trade-offs, and practical considerations of each workflow. Section 8 concludes the paper and suggests future directions.

2. Related work

PX4 and Ardupilot are two of the most widely adopted open-source autopilot platforms for small-scale UAVs. According to Alves (2022) [2], both systems utilize the MAVLink protocol for communication and offer similar hardware

DOI: 10.13009/EUCASS2025-628

compatibility. However, their licensing and development ecosystems differ significantly: PX4 uses a permissive BSD-3 license, allowing commercial adaptations, while Ardupilot's GPLv3 license enforces copyleft requirements. PX4 provides versatile flight modes such as Offboard mode, making it attractive for custom control research, whereas Ardupilot benefits from a larger community and greater sensor integration support. Nonetheless, Ardupilot's firmware size, which can sometimes exceed 1 MB, limits its use on certain autopilot hardware.

The flexibility of PX4 has made it a popular framework for the integration of advanced control strategies. Jing et al. (2022) [7] developed a Disturbance-Observer-Based Sliding Mode Controller (SMC) optimized using Particle Swarm Optimization (PSO) and implemented it in PX4. Their findings demonstrated superior tracking performance under disturbances and noisy measurements compared to traditional PID control. Although their implementation leveraged PX4's modularity, their validation remained in simulation using jMAVSim without real-world testing.

Niit and Smit (2017) [10] explored Model Reference Adaptive Control (MRAC) in the PX4 environment. Their work demonstrated MRAC's ability to adapt to dynamic changes in quadrotor properties, such as varying payloads, without needing retuning. While promising, their study provided limited implementation details, making replication challenging.

D'Angelo et al. (2023) adopted a different integration method by utilizing MATLAB's UAV Toolbox Support Package for PX4 Autopilots to embed a custom controller into PX4 for autonomous UAV operations. While the approach simplified the controller integration, they noted that overwriting PX4 firmware with UAV Toolbox removed critical system functionalities [5].

Similarly, Alqutami et al. (2024) [1] proposed an offboard control architecture where PX4 is coupled with ROS2 middleware to control fully actuated UAV platforms. This method provides controller flexibility but increases system complexity and introduces dependency on reliable high-bandwidth communication, which can be a limiting factor in real-world deployments.

Finally, the controller adopted in this study is based on the work of Ariyibi and Tekinalp (2020) [3], who proposed a hierarchical two-loop control system for quadrotors tasked with load transport missions. Their design combines a LQT velocity controller for thrust vector generation with a Quaternion-Based Nonlinear Controller for attitude regulation. Their framework offers an attractive alternative to PX4's default control stack and was validated in a theoretical and simulated context.

3. Controller overview

The custom controller implemented in this study follows a hierarchical two-loop structure, combining a LQT velocity controller with a Quaternion-Based Nonlinear Attitude controller. This architecture, adapted from Ariyibi and Tekinalp [3] (2020), enables robust and accurate trajectory tracking by addressing both translational and rotational dynamics separately.

3.1 Outer loop: LQT velocity controller

The outer loop controller generates a thrust vector that guides the vehicle's translational motion. The quadrotor's translational dynamics are first linearized around a hover condition, resulting in a state-space representation:

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{1}$$

where:

- $x(t) \in \mathbb{R}^3$ is the state vector containing velocity,
- $u(t) \in \mathbb{R}^3$ is the control input,
- A, B are system matrices derived from the linearization.

The LQT problem is formulated to minimize a performance index of the form:

$$J = \frac{1}{2} \int_0^\infty \left(e(t)^T Q e(t) + u(t)^T R u(t) \right) dt \tag{2}$$

where:

- $z(t)$ is the velocity setpoint,
- $y(t)$ is the measured velocity,

DOI: 10.13009/EUCASS2025-628

- $Q \geq 0$ and $R > 0$ are weighting matrices.

Solving this optimal control problem results in the following control law:

$$u(t) = Kx(t) + K_z z(t) + K_f f(t) \tag{3}$$

where:

- K is the state feedback gain, obtained by solving the Algebraic Riccati Equation (ARE),
- K_z is the feedforward gain associated with the reference trajectory,
- K_f compensates for constant disturbances like gravity.

The control law is obtained by solving the Algebraic Riccati Equation to find the matrix P:

$$A^\top P + PA - PBR^{-1}B^\top P + Q = 0 \tag{4}$$

The solution leads to the computation of the state feedback gain, the reference gain, and the disturbance compensation gain:

$$\begin{aligned} K &= -R^{-1}B^t P \\ K_z &= R^{-1}B^t \left[PE - A^t\right]^{-1} W \\ K_f &= -R^{-1}B^t \left[PE - A^t\right]^{-1} P \end{aligned} \tag{5}$$

where

$$\begin{aligned} E &= BR^{-1}B^t \\ W &= C^t Q \end{aligned} \tag{6}$$

are arbitrarily defined matrices by Naidu (2002)[9].

The virtual control input $u(t)$ essentially defines the desired specific force the quadrotor must produce to track the reference. This thrust vector is normalized to later determine the required attitude.

The **inner loop** handles attitude control. It constructs a desired "to-go" quaternion that aligns the quadrotor's body frame with the thrust vector generated by the outer loop. This quaternion is computed based on minimizing the misalignment between the body z-axis and the thrust direction, while also tracking a desired yaw angle [4].

3.2 Desired orientation generation

The desired orientation, referred to as the *to-go quaternion*, is generated to align the quadrotor's thrust vector with the desired thrust direction computed by the outer-loop velocity controller.

The thrust vector of the quadrotor, expressed in the body-fixed frame, is always aligned with the negative z-axis:

$$\vec{B} = -k \tag{7}$$

The desired thrust vector, calculated in the Earth-fixed frame and normalized, is:

$$\vec{E} = \frac{F_x^E \hat{i} + F_y^E \hat{j} + F_z^E \hat{k}}{F_{\text{total}}} \tag{8}$$

where $F_{\text{total}} = \sqrt{(F_x^E)^2 + \left(F_y^E\right)^2 + \left(F_z^E\right)^2}$.

To align the quadrotor's body thrust vector with the desired direction, the transformation from the Earth frame to the body frame is needed:

$$\vec{B} = C_{BE}\vec{E} = q^{-1} \otimes \vec{E} \otimes q \tag{9}$$

where q is the current quaternion orientation.

The rotation angle θ required to align the thrust vectors is computed as:

$$\vec{B} \cdot \vec{B}_{\text{desired}} = \cos(\theta) \tag{10}$$

and the rotation axis η_B is found via:

$$\vec{B} \times \vec{B}_{\text{desired}} = -\mathbf{k} \times \vec{B}_{\text{desired}} = \eta_B \sin(\theta) \tag{11}$$

DOI: 10.13009/EUCASS2025-628

The quaternion **s** responsible for adjusting roll and pitch (neglecting yaw) is constructed as:

$$\begin{aligned} s_4 &= \cos\left(\frac{\theta}{2}\right) = \sqrt{\frac{1+B_z}{2}} \\ s &= \frac{-\mathbf{k}\times\vec{B}}{2s_4} = \frac{B_y\hat{i} - B_x\hat{j}}{2s_4} \end{aligned} \tag{12}$$

Separately, the quaternion representing the yaw rotation is defined as:

$$y_4 = \cos\left(\frac{\Delta\psi}{2}\right), \quad y = \mathbf{k}\sin\left(\frac{\Delta\psi}{2}\right) \tag{13}$$

where $\Delta\psi$ is the difference between the current and commanded yaw angles.

Finally, the complete *to-go* quaternion t is obtained by multiplying the two quaternions:

$$\mathbf{t} = \mathbf{s} \otimes \mathbf{y} \tag{14}$$

This quaternion is used to command the necessary attitude for aligning the quadrotor's thrust vector with the desired thrust and simultaneously achieving the desired yaw.

3.3 Inner loop: Nonlinear attitude controller

The attitude error is regulated using a quaternion-based nonlinear control law:

$$\tau = \begin{bmatrix} U_2 \\ U_3 \\ U_4 \end{bmatrix} = G^{-1}\left[J^{-1}(M\,\mathrm{Im}(\mathbf{t}) - N\omega) - S\right] \tag{15}$$

where:

- τ is the commanded torque,
- $\mathrm{Im}(\mathbf{t})$ is the imaginary part of the to-go quaternion,
- ω is the angular velocity,
- M, and N are gain matrices,

and,

$$S = \begin{bmatrix} (I_x - I_z)\,qr/I_x \\ (I_z - I_x)\,pr/I_y \\ \left(I_x - I_y\right)pq/I_z \end{bmatrix}, G = \begin{bmatrix} d/I_x & 0 & 0 \\ 0 & d/I_y & 0 \\ 0 & 0 & 1/I_z \end{bmatrix}. \tag{16}$$

This control structure offers several benefits. The outer LQT controller ensures smooth and optimal translational motion, while the nonlinear quaternion-based inner loop avoids singularities inherent to Euler angle representations and ensures global stability properties over large attitude changes.

The full controller was first validated in Simulink simulation using a dynamic quadrotor model. Then, it was embedded into the PX4 software stack using two workflows: one using the UAV Toolbox, and the other through direct source code modification.

4. Implementation workflows

4.1 Simulink-Based implementation using UAV toolbox

The first approach explored in this study involves using the UAV Toolbox Support Package for PX4 Autopilots provided by MathWorks [6]. This toolbox facilitates the integration of custom controllers developed in Simulink directly into the PX4 firmware through automatic code generation.

DOI: 10.13009/EUCASS2025-628

4.1.1 Modeling and Setup

The custom controller was first implemented and validated in the Simulink environment alongside PX4's native controller, which was also duplicated in the same environment through studying the open-source code. Both controllers were tasked with commanding the position of a dynamic model of a quadcopter. Both controllers were designed to take in sensor inputs and output pulse-width modulation (PWM) outputs.

To evaluate the controllers within a more realistic PX4-compatible environment, the UAV Toolbox was utilized. This toolbox provides functionality to automatically generate C++ code from a Simulink model and deploy it to PX4 either on real hardware (via Hardware-in-the-Loop, HITL) or in a simulated setting (via SITL). In this study, the SITL workflow was selected, allowing the controllers to be tested using the jMAVSim simulator, which emulated the UAV's dynamics. This setup enabled comprehensive validation of control performance within the PX4 framework without requiring physical flight hardware.

4.1.2 Workflow

The workflow consisted of four main steps. First, the custom controller blocks were designed and tuned within a Simulink model, referencing the dynamic equations of motion and interfacing them with PX4-compatible input and output ports. This step was required to establish communication with other PX4 modules. Next, External Mode communication was set up to allow Simulink to run the controller in real time while monitoring and interacting with the PX4 SITL simulation environment. Once the model was ready, the UAV Toolbox's code generation tools were used to automatically convert the Simulink design into C++ code, which was then compiled and deployed onto the PX4 Host Target running on the local machine. Finally, the setup was tested using a range of predefined mission scenarios such as hover stabilization, waypoint navigation, orbit tracking, and step response tests. These scenarios were used to evaluate the controller's behavior and conduct iterative controller tuning.

4.2 Source code modification

The second workflow involves directly modifying the PX4 open-source autopilot firmware to embed a custom control logic as an internal module. Unlike the model-based Simulink approach, this method requires the developer to work within PX4's C++ codebase and architecture, but offers the benefit of retaining all built-in PX4 features including the ability to switch back to the native controllers in-flight. The required changes in the source code are summarized in Table 1. Figure 1 shows the original cascaded control architecture of PX4 and Figure 2 shows the custom controller architecture where the newly added layers are highlighted with a darker color.

The process begins by setting up the PX4 development environment. A new module named `mc_lqt_control` is created within the PX4 source tree. This module includes all source files necessary for the custom controller, such as position control logic, attitude computation, and torque generation. These are implemented across multiple source files, mirroring the structure and conventions of PX4's existing modules. The activation of the newly added module is done through activating a newly created flight mode called "LQT". This involves modifications in several parts of the PX4 firmware, including the `commander`, `vehicle_status`, and `vehicle_control_mode` files. A new flag, `flag_control_lqt_to_go_enabled`, is defined to indicate when the custom controller should be active. Additionally, a new value in the `NAVIGATION_STATE` enumeration is added to represent the newly added mode.

To pass data between the new module and other parts of PX4, a new uORB message type (`vehicle_local_position_setpoint_lqt`) is defined. This message structure contains desired position, velocity, acceleration, torque, and quaternion commands. The main controller module publishes this message, which can be read by downstream attitude or logging modules. The logging of uORB messages enable the developer to troubleshoot potential problems with plotting tools.

The controller conforms to the PX4 framework to run periodically during flight, triggered via the PX4 work queue. Once integrated, the entire setup is validated using SITL simulation in jMAVSim. After successful SITL tests, the firmware is compiled for the Cube Black target using `make px4_fmu-v3_default upload` and deployed to a physical quadrotor for real-world validation. The source code is publicly available. [1]

[1] https://github.com/straycat11/PX4-LqtControl

DOI: 10.13009/EUCASS2025-628

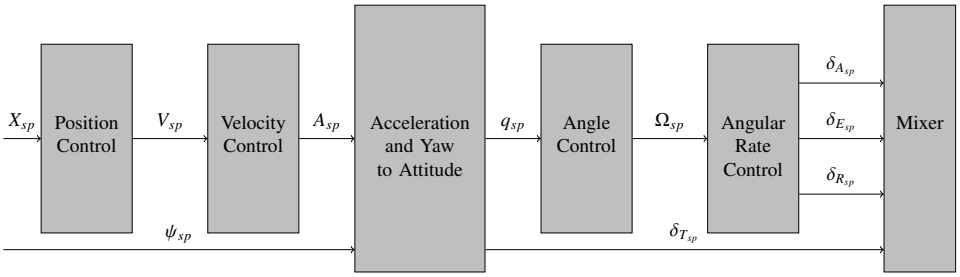


Figure 1: Cascaded control architecture of PX4

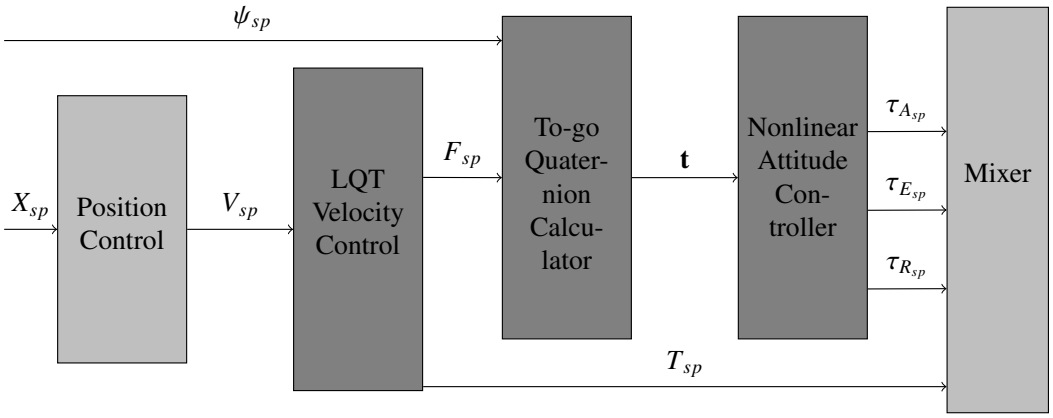


Figure 2: Custom controller architecture which replaces the blocks highlighted.

Table 1: PX4 Components Modified for Custom Controller Integration

File / Component	**Purpose / Action**
`mc_lqt_control/` module directory	Contains the control logic for the custom controller
`VehicleStatus.msg`	Added a custom navigation state for new flight mode
`VehicleControlMode.msg`	Added a flag to enable the controller
`Commander.cpp`	Defined activation of the new mode via state machine logic
`mode_requirements.cpp`	Defined sensor requirements for the custom mode
`FlightTask/FlightTaskLqt`	Optional: Defined a new flight task compatible with FlightTaskManager
`px4_custom_mode.h`	Assigned a numerical code to the custom mode
`control_mode.cpp`	Ensured the controller is activated correctly under custom mode
`conversions.hpp`	Added the conversion from `vehicle_status` to `navigation_mode` (uORB topics)
`ui.hpp`, `enums.json`	Updated the UI elements to display the new mode
`default.px4board`	Enabled the module for board configurations
`module.yaml`, `CMakeLists.txt` `rc.mc_apps`	Registered the new module for building and execution

5. Testing and validation

5.1 SITL testing setup

To evaluate the integration and performance of the custom controllers implemented through both workflows, SITL simulations were conducted using the jMAVSim simulator. PX4 SITL was developed by the community for testing of control algorithms within the PX4 autopilot stack, allowing access to actual PX4 modules without requiring physical hardware. The UAV model was based on the 3DR Iris quadcopter, which is the default airframe in jMAVSim. The simulated vehicle was equipped with standard PX4 sensor drivers and estimators. In both workflows, QGroundControl was used to monitor flight tests.

DOI: 10.13009/EUCASS2025-628

For the UAV Toolbox workflow, a setup that allows comparing the native and the custom controllers was built. Using this environment, the controllers were tested under identical position command sequences to evaluate their ability to stabilize and track desired trajectories. The maneuvers in the flight tests of this workflow were predefined position trajectories.

Predefined maneuvers were designed in the first workflow to evaluate controller performance:

- Hover: Commanding a fixed altitude to test stabilization,
- Orbit: Generating a circular trajectory to test lateral tracking,
- Step Response: Testing transient response by commanding a sudden position change.

For the source code modification workflow, the modified PX4 firmware containing the custom flight mode and control module was built and run in SITL. Log data was collected via PX4's built-in logging infrastructure. Here, the maneuvers were not predefined trajectories; rather, the quadcopter was controlled via joystick inputs.

Joystick control allowed for the following maneuvers:

- Hover,
- Manual horizontal movement test: Giving back and forth position commands,
- Step Response.

The test data was processed to visualize performance metrics, such as position error and trajectory tracking.

5.2 Real-world flight test

Following the successful completion of SITL testing, the source code modification workflow was selected for real-world validation. The custom-built quadrotor platform shown in Figure 3 was assembled to carry the modified PX4 firmware. The drone was built using an F450 frame with a diagonal motor-to-motor length of 450 mm and a total takeoff weight of approximately 1500 grams. It was equipped with four T-Motor 40PRO KV1600 brushless motors, generic 40A ESCs, a Here 2 GPS module, and a Cube Black flight controller running PX4.

Figure 3: Custom-built quadcopter.

The modified firmware included the custom controller implemented as a dedicated module and a new flight mode for activation. Firmware deployment was performed using PX4's standard build and deployment system. Once uploaded, the system was configured and calibrated via QGroundControl.

Initial flight tests were conducted in a controlled outdoor environment. The UAV was first flown using standard PX4 flight modes to verify correct system behavior and stability. The custom flight mode was then engaged mid-flight to observe the behavior of the custom controller. Flight test scenarios included hovering at a fixed altitude and performing simple translational maneuvers. The aim was to evaluate the real-world response of the controller in terms of stability, responsiveness, and compatibility with PX4's onboard estimation and sensor systems. Data was collected using onboard logging for post-flight analysis.

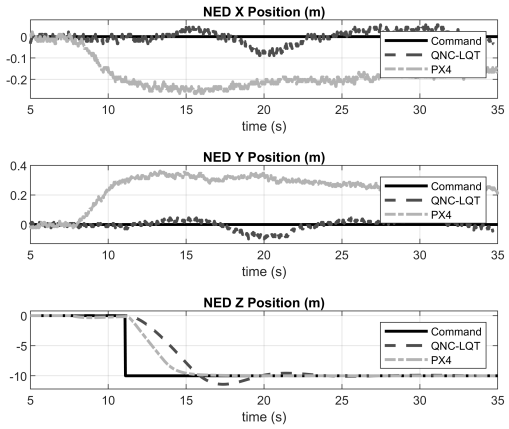
DOI: 10.13009/EUCASS2025-628

6. Results and Evaluation

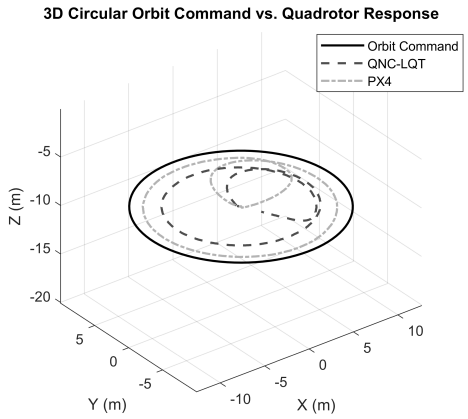
6.1 UAV Toolbox approach

6.1.1 Simulation results

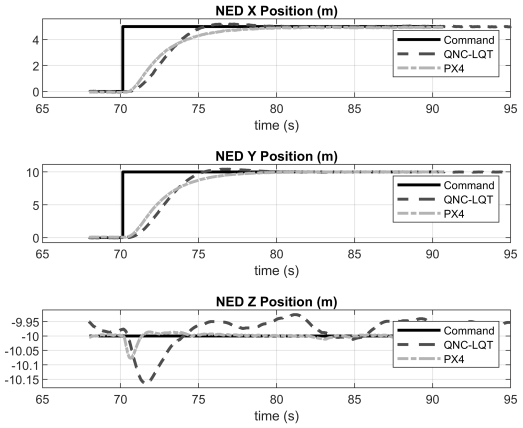
This section compares the performance of the PX4 default controller and the custom QNC-LQT controller across key flight tasks in SITL simulations. The simulated data for demonstrating the custom controller is generated by following the first workflow. Although there are significant differences in performance, both controllers perform comparably, proving the viability of the workflow.



(a) Hover acquisition performance



(b) Orbit trajectory comparison



(c) Step response in position

Figure 4: SITL test results of the first workflow

Figure 4a shows the vertical position tracking performance of both controllers during the hover acquisition phase. The custom controller reaches the desired altitude with a longer rise time and increased overshoot compared to PX4's default which demonstrates a more aggressive but controlled thrust profile. Minor oscillations are observed in the custom controller during settling, whereas the native controller converges more smoothly.

Figure 4b depicts the orbit trajectory of the UAV under both controllers. The QNC-LQT controller produces a visibly less precise orbit, with noticeable deviation from the desired circular path. In contrast, the PX4 controller maintains tighter conformity to the planned path, showing better lateral tracking performance in coordinated maneuvers.

To examine responsiveness, both controllers were subjected to a sudden change in position setpoint. As shown in Figure 4c, the custom controller exhibits a quicker transition and higher overshoot. The PX4 controller takes longer to settle. The custom controller struggles more than the native controller to hold its vertical position.

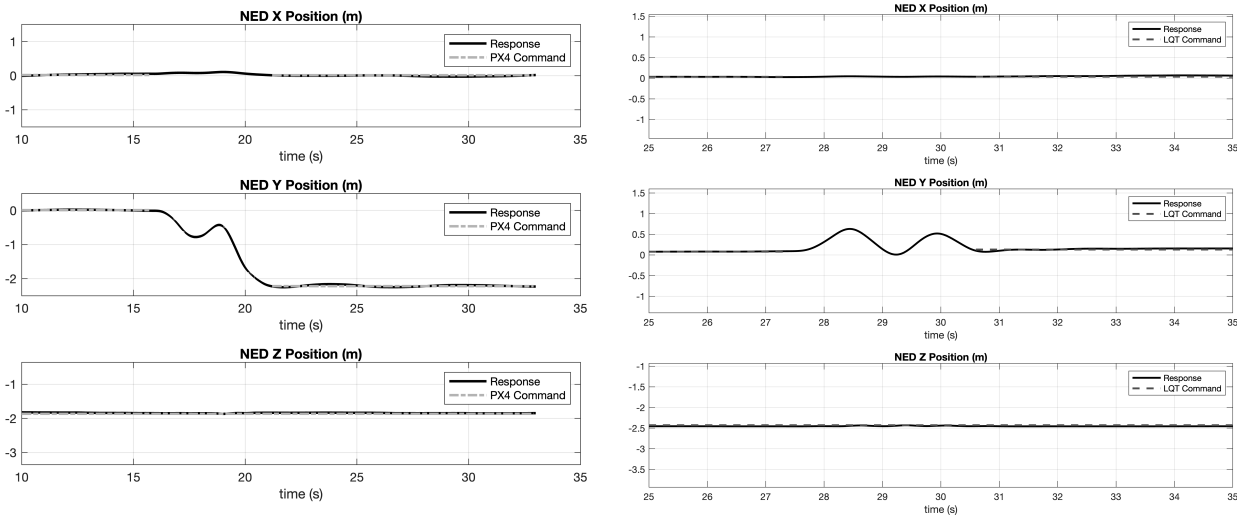
DOI: 10.13009/EUCASS2025-628

6.2 Source code modification approach

6.2.1 Simulation results

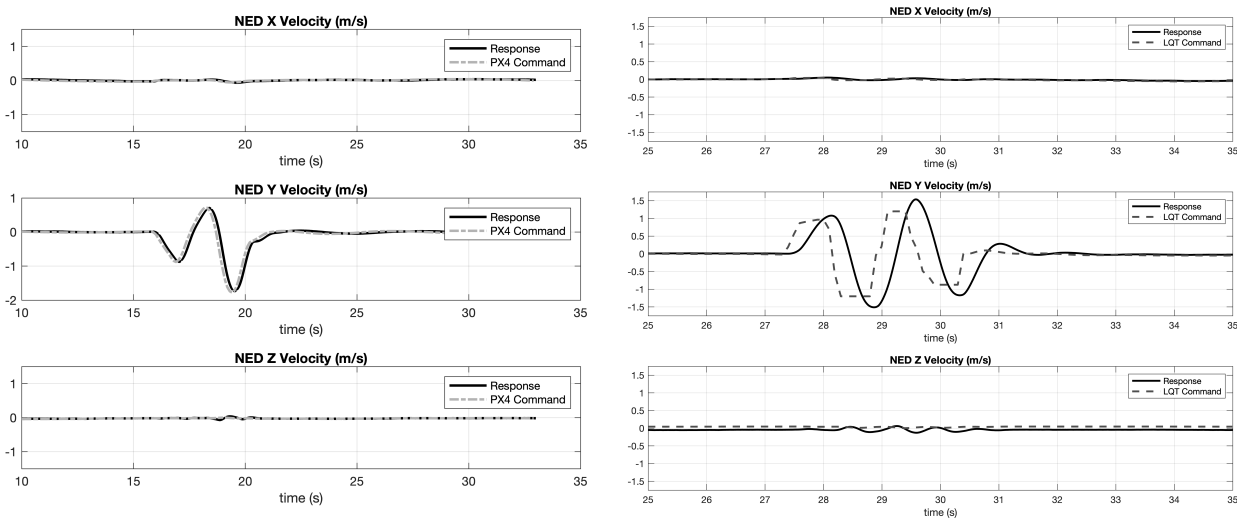
To evaluate the functionality and performance of the custom controller integrated directly into the PX4 firmware, a series of SITL simulations were conducted using PX4's native SITL environment. In the initial phase of testing, the unmodified PX4 source code was built and executed in SITL. Standard position control maneuvers—such as hover, step response, and horizontal trajectory tracking—were carried out using the default PX4 position controller. The performance of these maneuvers served as a baseline, highlighting the behavior of the PX4 control stack. Following this, the modified firmware was built and launched in the same SITL setup, and the identical set of position control maneuvers was repeated.

Although no hardware was involved in this phase, the SITL helped ensure the custom mode integrated seamlessly with the PX4 stack and behaved predictably. This setup also enabled fast iteration before proceeding to real-world flight testing.



(a) Native controller performance (b) Custom controller performance

Figure 5: SITL horizontal maneuver test position tracking results of the second workflow



(a) Native controller performance (b) Custom controller performance

Figure 6: SITL horizontal maneuver test velocity tracking results of the second workflow

DOI: 10.13009/EUCASS2025-628

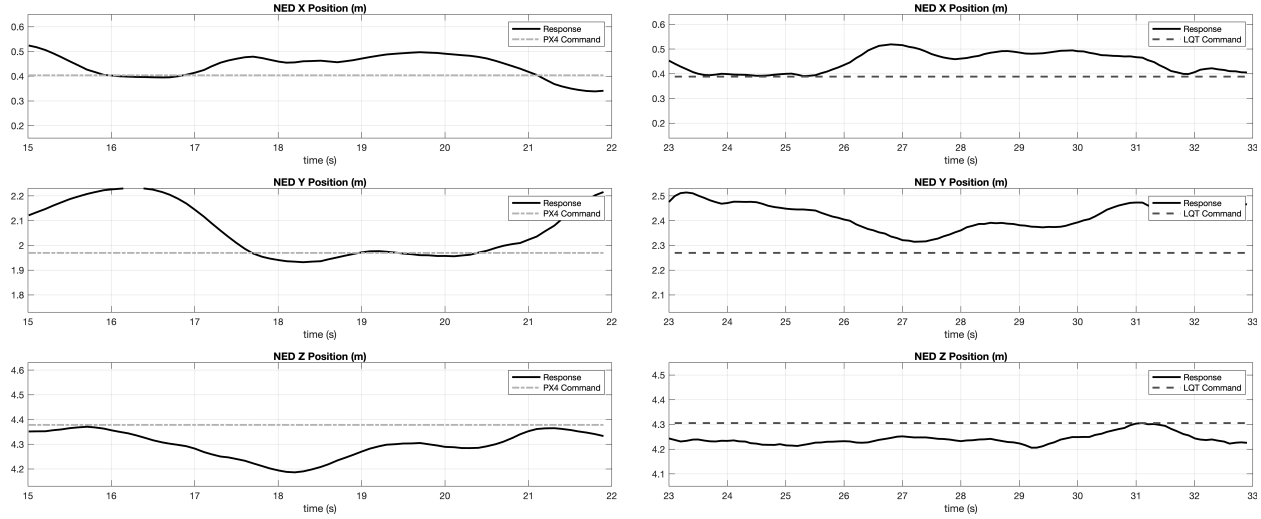
Figure 5 illustrates that both controllers successfully stabilize the vehicle's position once the control inputs return to neutral. Despite simulated disturbances and non-ideal dynamics, the vehicle consistently settles after initial transients, indicating that the custom controller has been correctly integrated and operates reliably within the PX4 software architecture.

Similarly, as shown in Figure 6, both the PX4 native controller and the QNC-LQT respond effectively to velocity commands. Upon releasing the sticks, the vehicle returns to a stable hover, demonstrating proper handling of velocity inputs and confirming the intended behavior of both control strategies. The custom controller does not respond as quickly to the commanded velocities as the native, suggesting the need for further tuning.

6.2.2 Flight test results

Flight tests were conducted outdoors in GPS-available conditions using the quadcopter described in Section 5 where ambient wind remained a non-negligible factor and contributed to the real-world performance evaluation. The PX4 controller was flown with default gain values, while the custom controller was tested using the gain settings derived from the SITL simulations.

Tests replicated the methodology used in the SITL experiments to ensure comparability. The quadcopter was first stabilized in hover and switched to the desired flight mode (Position or LQT) before executing each maneuver. While environmental disturbances and RC handling imperfections inevitably affect ideality, the focus of the tests was to demonstrate that the implementation of custom autopilot modes within PX4 is feasible and robust in real-world conditions.



(a) Native controller performance

(b) Custom controller performance

Figure 7: Hover position tracking during flight test — PX4 vs. custom controller

Figure 7 compares the position-holding behavior of the two controllers under moderate wind conditions. The controllers demonstrate similar position fluctuations around their commanded positions. The velocity-tracking behavior of the controllers are shown in Figure 8. The custom controller exhibits a twitchier response to the commanded velocities. This shows that although the controller manages to hold position, its performance may not be optimal.

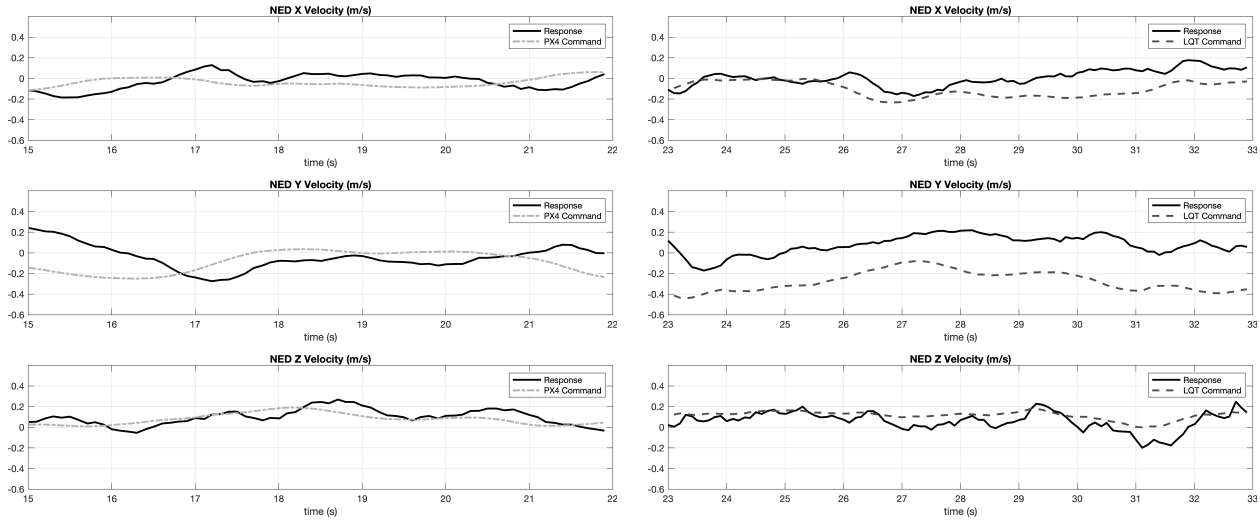
Figure 9 and Figure 10 show the response of each controller to a 3211-like input. Both controllers tracked their current positions when the RC sticks are centered and they started tracking velocity commands as soon as the maneuvers were conducted. The commanded constant vertical position is kept by both of the controllers with minimal deviations.

The native controller tracks in a smoother and more precise way than the custom controller. The latter exhibits delays in velocity tracking, which results in the quadcopter not moving as much as when the former controls it. The more damped characteristic of the custom controller can be observed by looking at its velocity tracking behavior.

7. Discussion

Overall, the results confirm that both workflows produced valid implementations of the custom controller, with the source code modification approach proving viable for real-world deployment. Differences in position offset, recovery

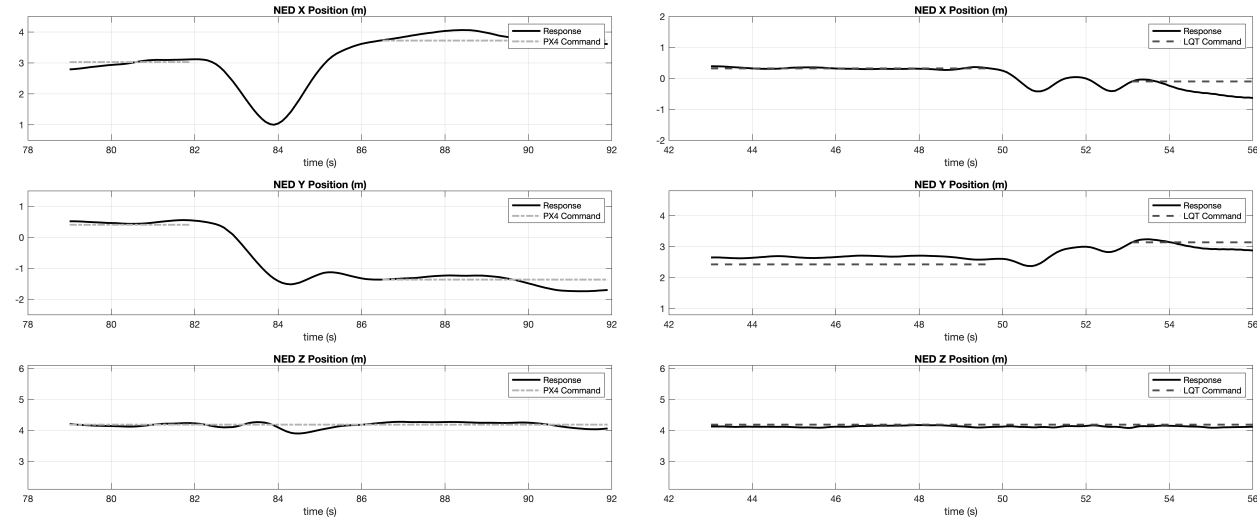
DOI: 10.13009/EUCASS2025-628



(a) Native controller performance

(b) Custom controller performance

Figure 8: Hover velocity tracking during flight test — PX4 vs. custom controller



(a) Native controller performance

(b) Custom controller performance

Figure 9: Horizontal maneuver position tracking during flight test — PX4 vs. custom controller

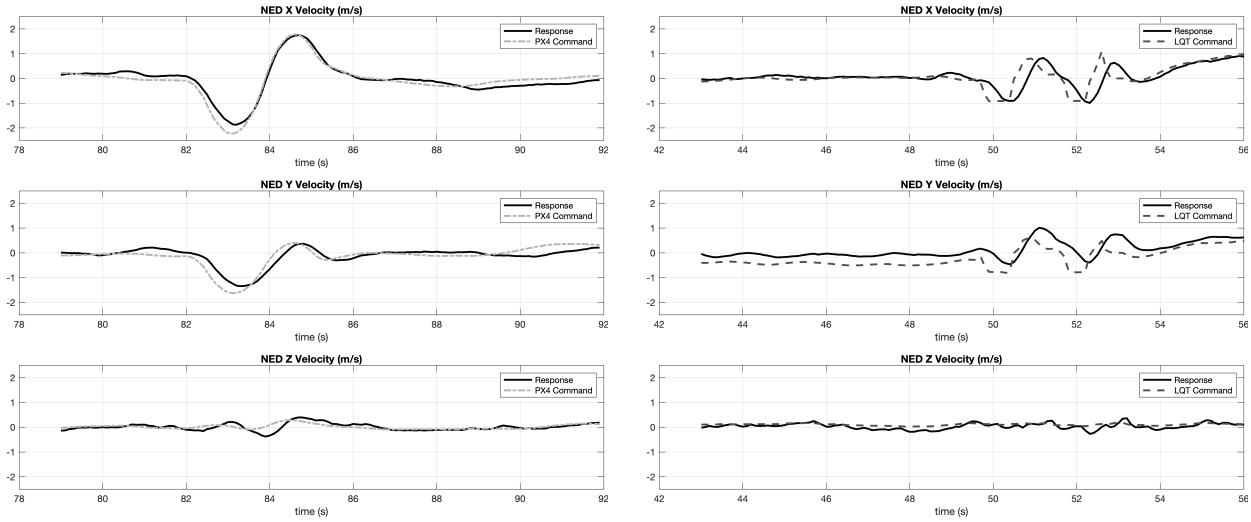
behavior, and oscillation levels are evident, but these may be attributed to differing control strategies and tuning. Importantly, both implementations handled the manual maneuver without divergence or control loss, supporting the reliability of both systems under dynamic, real-world flight conditions. While full velocity and error plots are omitted for brevity, they are consistent with the trends shown here and can be found in the thesis[2].

One key observation is the tradeoff between fidelity and automation. The UAV Toolbox workflow, although it lacks the depth of many PX4 features, excels in enabling structured and repeatable testing. Through the Simulink interface, a developer can easily run batch simulations, adjust parameters programmatically, and benefit from MATLAB's post-processing tools to perform detailed analysis. This creates a testing environment that is ideal for controller tuning and early-stage validation, particularly when controller performance must be evaluated across a wide range of input scenarios. Furthermore, SITL tests using the UAV Toolbox can be standardized, ensuring that environmental conditions, initial states, and control inputs remain identical across iterations — something that is harder to guarantee in manual testing setups.

However, this flexibility comes at the cost of realism. The UAV Toolbox workflow overrides many PX4 mechanisms. As such, controllers implemented this way are not directly portable to real hardware without substantial

[2]See author's thesis for complete result set.

DOI: 10.13009/EUCASS2025-628



(a) Native controller performance

(b) Custom controller performance

Figure 10: Horizontal maneuver velocity tracking during flight test — PX4 vs. custom controller

adaptation. This limits its use to pre-deployment verification rather than final integration.

In contrast, the source code modification approach retains the full capabilities of PX4. The workflow, though more involved, allows the custom controller to be treated as a first-class component within the flight control stack. This makes it more suitable for deployment and testing in real-world conditions, especially when adherence to PX4's internal conventions and safety systems is critical.

Taken together, the two approaches are best viewed as complementary. The UAV Toolbox provides a testing platform that helps iterate on controller logic with ease. Once the controller design stabilizes, the source code modification workflow provides the path toward field deployment. For researchers and developers aiming to bring novel control algorithms into operational use, this two-phase workflow offers both speed and reliability.

8. Conclusion and further work

This study presented two structured workflows for integrating custom autopilot controllers into the PX4 open-source framework. Rather than aiming to enhance PX4's native flight control performance, the focus was on demonstrating how novel control architectures can be deployed using two practical methods: one based on the UAV Toolbox Support Package for PX4 Autopilots, and another through direct modification of PX4's source code.

The custom controller implemented combines a LQT velocity controller with a quaternion-based nonlinear attitude controller. The outer loop generates velocity commands using an optimal control formulation, while the inner loop computes orientation commands in quaternion space and applies a nonlinear control law for attitude stabilization. This architecture was first validated using a custom-built dynamic model and then deployed within both workflows.

The UAV Toolbox workflow allowed for rapid development in a SITL environment but removed several core PX4 functionalities. It proved valuable for early-stage validation and controller tuning. In contrast, the source code modification workflow involved a more elaborate integration process, including defining a new flight mode, adding custom modules, implementing uORB communication, and integrating parameter configuration. Despite the added complexity, this workflow preserved all PX4 features, enabling robust real-world deployment. The modified firmware was successfully deployed on a Cube Black-based quadrotor, and the custom controller was flight-tested under various manual maneuvers, validating the simulation findings.

These results demonstrate that the proposed source code modification workflow provides a reliable method for transitioning theoretical control designs into real-world UAV applications. It also illustrates the strengths of PX4's modular design and the power of open-source collaboration in accelerating UAV innovation.

8.1 Future Work

Several extensions can build upon this work:

DOI: 10.13009/EUCASS2025-628

- **Hardware-in-the-Loop Testing:** Introducing HITL can add an incremental step between simulation and flight, enabling higher-fidelity testing in real-time without risk to physical assets.
- **Autonomous Mission Integration:** Future development can extend the custom controller to operate within PX4's mission framework for automated flight modes such as takeoff, landing, and waypoint navigation. Uploading predefined trajectories as MAVLink missions can make performance tests more repeatable and comparable across environments.
- **Data-Driven Tuning:** Machine learning or system identification techniques could be used to refine control gains based on real flight data.
- **Airframe Generalization:** While the custom controller theoretically supports multiple airframes, empirical testing across different UAV platforms would validate its robustness and adaptability.

References

[1] Tareq Aziz Hasan Alqutami, Matthew W. Dunnigan, and Yvan Petillot. "Modeling and Motion Control of a Fully Actuated Multirotor for Aerial Manipulation Using PX4 and ROS2". English. In: *9th International Conference on Mechatronics Engineering (ICOM)*. 9th International Conference on Mechatronics Engineering 2024, ICOM 2024 ; Conference date: 13-08-2024 Through 14-08-2024. United States: IEEE, Sept. 2024, pp. 396–401. DOI: 10.1109/icom61675.2024.10652521.

[2] Joao Filipe Gomes Moreira Alves. "Software Architecture for Low-cost UAVs: An application considering automatic target tracking mission scenarios Military and Aeronautical Sciences-Aviator Pilot Examination Committee". MA thesis. Academia Da Forca Aerea, May 2022. URL: https://comum.rcaap.pt/handle/10400.26/50790.

[3] Segun O. Ariyibi and Ozan Tekinalp. "Quaternion-based nonlinear attitude control of quadrotor formations carrying a slung load". In: *Aerospace Science and Technology* 105 (Oct. 2020), p. 105995. ISSN: 12709638. DOI: 10.1016/j.ast.2020.105995.

[4] Omer Atas and Ozan Tekinalp. "ATTITUDE CONTROL OF AN EARTH ORBITING SOLAR SAIL SATELLITE TO PROGRESSIVELY CHANGE THE SELECTED ORBITAL ELEMENT". In: *25th American Astronautical Society (AAS)/American-Institute-of-Aeronautics-and-Astronautics (AIAA) Space Flight Mechanics Meeting*. Vol. 155. American-Institute-of-Aeronautics-and-Astronautics (AIAA), 2015. URL: https://hdl.handle.net/11511/54433.

[5] Simone D'Angelo et al. "Efficient Development of Model-Based Controllers in PX4 Firmware: A Template-Based Customization Approach". In: *2024 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, June 2024, pp. 1155–1162. ISBN: 979-8-3503-5788-2. DOI: 10.1109/ICUAS60882.2024.10556938.

[6] The MathWorks Inc. *UAV Toolbox Reference*. Mathworks. 2024. URL: https://www.mathworks.com/help/pdf_doc/uav/uav_ref.pdf.

[7] Yutao Jing et al. "PX4 Simulation Results of a Quadcopter with a Disturbance-Observer-Based and PSO-Optimized Sliding Mode Surface Controller". In: *Drones* 6 (9 Sept. 2022). ISSN: 2504446X. DOI: 10.3390/drones6090261.

[8] Lorenz Meier et al. *PX4/PX4-Autopilot: Stable Release v1.14.0*. Version v1.14.0. Oct. 2023. DOI: 10.5281/zenodo.10023585. URL: https://doi.org/10.5281/zenodo.10023585.

[9] D.S. Naidu. *Optimal Control Systems*. Electrical Engineering Series. Taylor & Francis, 2002. ISBN: 9780849308925. URL: https://books.google.com.tr/books?id=hGxurdEZVtkC.

[10] Edgar A. Niit and Willie J. Smit. *2017 24th International Conference on Mechatronics and Machine Vision in Practice (M2VIP)*. IEEE, 2017. ISBN: 9781509065462. DOI: https://doi.org/10.1109/M2VIP.2017.8211479.