Politecnico di Torino

Department of Mechanical, Aerospace
and Automotive Engineering

Master of Science in Aerospace Engineering

Master Thesis

Design and Implementation of an on-board control system for quadrotors

Supervisor
Prof. Elisa CAPELLO
Dott. Dmitry IGNATYEV

Candidate
Paolo Cesano

Academic Year
2024-2025

Contents

1 Abstract

This thesis presents the design, implementation, and validation of an Incremental Nonlinear Dynamic Inversion (INDI) controller for a quadrotor using the PX4 framework, following a V-shaped model validation approach. The primary objective of this research is to develop a robust controller capable of executing autonomous missions under model uncertainties and external disturbances, while also assessing its fault-tolerant capabilities and response to aggressive manoeuvres.
The INDI controller is designed in combination with a proportional-derivative (PD) controller to enhance stability and robustness. The validation process begins with Model-in-the-Loop (MIL) simulations, conducted in MATLAB and Simulink, to establish an initial tuning of the control parameters based on a high-fidelity quadrotor model. These simulations provide a first insight into the expected closed-loop response and dynamic behavior of the system.
Following MIL validation, Software-in-the-Loop (SIL) simulations are performed to ensure compatibility between the generated C++ code and the PX4 autopilot software architecture. At this stage, the controller is integrated into the PX4 firmware, and its performance is evaluated within the simulation environment, verifying that the numerical implementation remains consistent with the initial theoretical formulation.
The next phase involves Processor-in-the-Loop (PIL) simulations, where the controller is executed on the CubePilot Black FMUv3 flight controller to assess real-time computational feasibility and onboard execution performance. This step is crucial to ensure that the embedded system can handle the control law in real-world scenarios without excessive delays or computational bottlenecks.
The controller's performance is evaluated through a series of increasingly complex trajectory tracking experiments, including circular trajectories, figure-eight patterns, and waypoint-based navigation tasks. The experimental results are analyzed to quantify the controller's accuracy, robustness, and ability to reject disturbances. Furthermore, additional test scenarios explore the response variations in the system's mass and inertia.
This research not only demonstrates the feasibility of implementing advanced nonlinear control strategies on resource-constrained embedded flight controllers but also provides a structured methodology for bridging the gap between theoretical control design and real-world deployment in aerial robotics.

2 State of art

2.1 Types of Controllers Used

A briefly description on dynamic system is done, Before that the INDI control algorithm would be explained. A dynamic system is a system characterized by a set of variables whose states evolves over time according to physical law, and are often described using mathematical model such as differential equations or state-space equation.

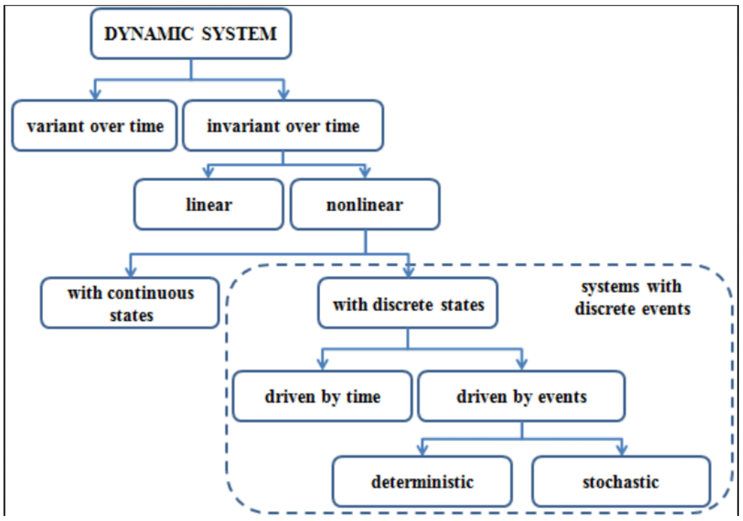


Figure 2.1: Scheme of Dynamic Systems[3].

In this specific case the state-space formulation would be introduced to describe linear and non-linear time-invariant model.

Linear System Dynamic Suppose the system dynamics is represented as a linear time-invariant(LTI) model:

$$\begin{cases} \dot{x} = Ax + Bu \\ y = Cx \end{cases} \tag{2.1}$$

where $x \in \chi \subseteq \mathbb{R}^n$ is a state vector, $u \in \chi \subseteq \mathbb{R}^m$ are the control inputs, $y \in \chi \subseteq \mathbb{R}^q$ is the output vector, $A \in \mathbb{R}^{n \times n}, B \in \mathbb{R}^{n \times m}, C \in \mathbb{R}^{n \times q}$ are matrices with constant real entries. In this case, the control system can be straightforwardly using various classical design tools such as pole placement, eigenstructure assignment, etc[2].
This arise from Taylor linearization around an equilibrium point, steady-state condition, and it a reliable method to design a linear controller but can not cover the whole domain of the system, for UAV usually called flight envelope, where can occur highly non-linear phenomenons. One solution was offered by gain scheduling the flight envelope[22].
Usually the gain scheduling method is implemented through several steps, first a portion of the flight envelope is necessary to define different operational region where an equilibrium point is defined and linearized around it. Different linear controllers are defined for each region, sometimes the gain values are blended between operational region.
A global FCS is defined as single autopilot unified, normally a satisfactory performance is achieved across the flight envelope, the single LTI is activated when the current operating point

match an equilibrium point. The global autopilot still a linear system but with varying parameter through the time, parameter that usually are altitude, speed or high operating angle. This system is mathematically defined as linear parameter-varying (LPV) model:

$$\begin{cases} \dot{x} = A[\theta(t)]x + B[\theta(t)]u \\ y = C[\theta(t)]x \end{cases} \tag{2.2}$$

As mentioned before A, B, C are not fixed as in the LTI system, instead varying over time, θ is the defined parameter and has to be a "slow variable" that means that the changes in $\theta(t)$ should occur at a much slower pace than the changes in the system's variables x.
However, in the flight envelope can occurs high non linear phenomenal, as shock waves or operating on high angle, that still difficult to discern from typical non-linear behaviour and disturbances. In that region x states deviate significantly from the linearized ones restabilizing the closed-loop. Non-linear dynamics can be incorporated using a quasi-linear time-varying (QLTV) models:

$$\begin{cases} \dot{x} = A(x,t)x + B(x,t)u \\ y = C(x,t)x \end{cases} \tag{2.3}$$

or in a more implicit time dependence quasi-linear parameter-varying (QLPV) model:

$$\begin{cases} \dot{x} = A[x,\theta(t)]x + B[x,\theta(t)]u \\ y = C[x,\theta(t)]x \end{cases} \tag{2.4}$$

Sliding Mode Control Sliding Mode Control (SMC) is one of the most widely used non-linear control algorithms for UAVs[15]. SMC is known for its robustness against unmodeled dynamics, parametric uncertainties, and external disturbances[20]. The dynamic behaviour of a UAS can be decomposed into slow and fast dynamics. In this context, the system equations can be reformulated in a strict feedback form as follows:

$$\begin{cases} \dot{x}_1 = f_1(x_1) + g_1(x_1)u_1 \\ \dot{x}_2 = f_2(x_1,x_2) + g_2(x_1,x_2)u_2 \end{cases} \tag{2.5}$$

where $x_i \in X_i \subseteq \mathbb{R}^n$ represents the system states, corresponding to the slow and fast dynamics, respectively. The variables $u_i \in U_i \subseteq \mathbb{R}^m$, denotes the control inputs, while f_i, and g_i represent the system dynamics and control efficiency functions, respectively. For notational convenience, the concatenated vectors $x = [x_1, x_2]^T, f = [f_1, f_2]^T, g = [g_1, g_2]^T, u = [u_1, u_2]^T$ are introduced. A sliding mode controller applicable to both fast and slow dynamics can be designed for the system described, the sliding surface is defined using a proportional-integral feedback approach[14]:

$$S = e + K \int_0^t e dt \tag{2.6}$$

where $e = x - x_r$ represents the tracking error, and K is a constant gain matrix. To ensure system stability, a Lyapunov candidate function is introduced as a positive definite function:

$$V = \frac{1}{2}S^T S \tag{2.7}$$

Taking the derivate of V along the system trajectories yields:

$$\dot{V} = S^T S = f + gu - \dot{x}_r + Ke \tag{2.8}$$

To ensure that $\dot{V}$ remains negative, the following sliding mode control law is implemented:

$$u = g^{-1}[-f + \dot{x}_r - Ke - c_1 S - c_2 sgn(S)] \tag{2.9}$$

where c_1 and c_2 are positive-definite and positive-semidefinite diagonal matrices. For most UASs, the functions $g_1 and g_2$ remain non-zero within the operational flight envelope, ensuring the existence of the inverse transformation of g. Substituting this control law into the Lyapunov derivative yields:

$$\dot{V} = -S^T c_1 S - c_2 \|S\| < 0, \quad \forall S \neq 0 \tag{2.10}$$

which guarantees global uniform asymptotic stability. SMC compels the system trajectory to reach a predefined subspace of the state space, known as the sliding surface, within a finite time[17]. Once the system reaches this surface, the state variables asymptotically converge to zero. If the trajectory deviates above or below the surface, SMC applies a discontinuous switching control law to drive the trajectory back onto the sliding surface[7].
However, the inherent discontinuous nature of the SMC induces the chattering phenomenon, characterized by high-frequency oscillations in the control signals. From a practical perspective, excessive chattering can excite unmodeled high-frequency system dynamics as structural modes, cause actuator or onboard systems degradation, potentially leading to instability or incapacity of fulfilling the mission. This chattering effect originates from the discontinuous signum function $sgn(S)$ employed in the control law.
A commonly adopted mitigation strategy involves replacing the signum function with a saturation function $sat(y, \epsilon)$, defined as follow[7].

$$sat(y, \epsilon) = \begin{cases} y, & if|y| \leq 1 \\ sgn(y), & if|y| > 1 \end{cases} \tag{2.11}$$

where ϵ is a positive constant. The slope of the linear region of $sat(y, \epsilon)$ is $\frac{1}{\epsilon}$, and thus, ϵ should be minimized to approximate the signum function as closely as possible[14].
The presence of disturbances, variations in mass/inertia properties and environmental factor such as as wind gusts can significantly alter system dynamics, introducing uncertainties. To enhance disturbance rejection capabilities a combination of PD control technic and SMC or use an adaptive SMC strategy could be used.

Backstepping Backstepping is a widely adopted methodology for design control strategies in non linear system particularly for trajectory tracking applications[14].
Consider the construction of backstepping control for the following nonlinear strict-feedback

system:

$$\begin{aligned}
\dot{x} &= f(x) + g(x)\xi_1 \\
\dot{\xi}_2 &= f_1(w_1) + g_1(w_1)\xi_2 \\
\dot{\xi}_3 &= f_2(w_2) + g_2(w_2)\xi_3 \\
&\vdots \\
\dot{\xi}_{k-1} &= f_{k-1}(w_{k-1}) + g_{k-1}(w_{k-1})\xi_k \\
\dot{\xi}_k &= f_k(w_k) + g_k(w_k)u
\end{aligned} \quad (2.12)$$

(2.13)

where $x \in \mathbb{R}^n$ represents the states vector, $\xi_i \in \mathbb{R}$ for $i = 1, ..., k$ are auxiliary states and $u \in \mathbb{R}$ is the control input. The functions f and f_i describe the system dynamics, and g_i are the control efficiency functions. For compact notation, let us define the auxiliary variables $w_i = [x^T, \xi_1, ..., \xi_i]^T$. This structure is termed "strict-feedback" because the non linearities in the equations governing x depend exclusively on x, whereas those affecting ξ_i are functions of x and preceding state variable $\xi_1, ..., \xi_i$, thus forming a feedback loop.
To design a backstepping controller, we first analyse the subsystem consisting of (x, ξ_1) in this case, ξ_1 is treated as a virtual control input. A Lyapunov function candidate is introduced as:

$$V_1(x, \xi) = V(x) + \frac{1}{2}|\xi - a(x)|^2 \quad (2.14)$$

Where $V(x)$ is a positive definite Lyapunov function ensuring stability for $\dot{x} = f(x)$, and $a(x)$ is a stabilizing function designed for the system. These functions serve as intermediate control laws, also known as stabilizing functions.

$$a_1(x, \xi_1) = \frac{1}{g_1(x, \xi_1)}(-c_1[\xi_1 - \alpha(x)] - \frac{\partial V}{\partial x}(x)g(x) - f_1(x, \xi_1) + \frac{\partial \alpha}{\partial x}(x)[f(x) + g(x)\xi]) \quad (2.15)$$

where $c_1 > 0$ is a positive gain. Various alternative definitions for a_1 can be employed, even in cases where $g_1(x, \xi)$ is zero in certain regions.
This methodology extends recursively to the subsystems $\xi_1, ..., \xi_k$, where at each stage, an auxiliary control function a_i is defined to ensure stabilization. The final control law u is then expressed as:

$$u = \frac{1}{g_k}(-c_k[\xi_k - \alpha_{k-1}] - \frac{\partial V_{k-1}}{\partial \xi_{k-1}} g_{k-1} - f_k + \frac{\partial \alpha_{k-1}}{\partial X_{k-1}}[F_{k-1} + G_{k-1}\xi_k] \quad (2.16)$$

However, this approach requires evaluating multiple time derivatives of the system model, which can lead to highly complex control formulations, especially for high-dimensional systems.[4]

Non-linear Dynamic Inversion Nonlinear Dynamic Inversion(NDI) is a well-established and widely implemented control technique for nonlinear systems, particularly in aerospace and robotics application[22]. The fundamental idea behind NDI is to compensate for the inherent non-linearities of a system by mathematically inverting its dynamic[1]. This approach effectively

transform the system into a linear one within a specific operating region, thereby simplifying control design and analysis.
Taking into account, the dynamic system in [2.1] could be seen as the derivative output is evaluated as[1]:

$$\dot{y} = \frac{\partial h}{\partial x}\frac{dx}{dt} = \frac{\partial h}{\partial x}f(x) + \frac{\partial h}{\partial x}g(x)u = F(x) + G(x)u \tag{2.17}$$

At this point the control output can be evaluated as follow:

$$u_c(t) = G^{-1}(x)(\dot{y}_{des}(t) - F(x)) \tag{2.18}$$

However, a key assumption underlying this method is that the system model is both accurate and complete. Any discrepancies between the mathematical model and the actual system, such as parametric uncertainties in the dynamics of unmodeled conditions, can significantly affect the performance and stability of the controller[8].

2.2 Research aims and objectives

The aim of this research is to implement an Incremental Non-linear Dynamic Inversion(INDI) controller, incorporating a linear PD controller to evaluate the input, within the well-established PX4 autopilot framework. This implementation seeks to achieve robust control over the drone's behaviour, enabling it to successfully complete aggressive manoeuvre[24], even in presence of uncertainties or unmodeled dynamics. Additionally, this research aims to test the fault-tolerant capabilities of the INDI controller against propellers degradation or even complete engine failure on one of the motors,
While the long-term goal of this research is to explore both of this topics, this thesis focuses specifically on achieving experimental flight using INDI algorithms. Notably, the experimental setup will not use real RPM values as feedback; instead, a virtual RPM value will be evaluated within the code. The results and the methodology in this thesis will lay the foundation for future work, advancing the integration of real-time feedback and testing under more complex conditions.

3 Theoretical background

3.1 V-model

The V-model is a structured approach to system development that emphasizes rigorous validation and verification at every stage of the design process. Widely adopted in the development of control system, the V-model provides a clear pathway from initial concept to system deployment. In this paragraph will be discuss the theoretical application of a V-model based approach that will be used as a reference through the stages of this research.
The V-model is characterized by its V-shaped structures, where the left side branch represents system design and decomposition from high level requirements to specific hardware requirements. The right branch instead focuses on integration, verification and validation. Moving instead from the top to the down part of the diagram the focus goes from the whole system to specific units of the system. So the diagram can be read in the both direction, x label is the time, y label how deep the stage is in the system. V-shape is due to that for every stage of the design process correspond to a validation activity, ensuring that requirements are met at all stages[18].

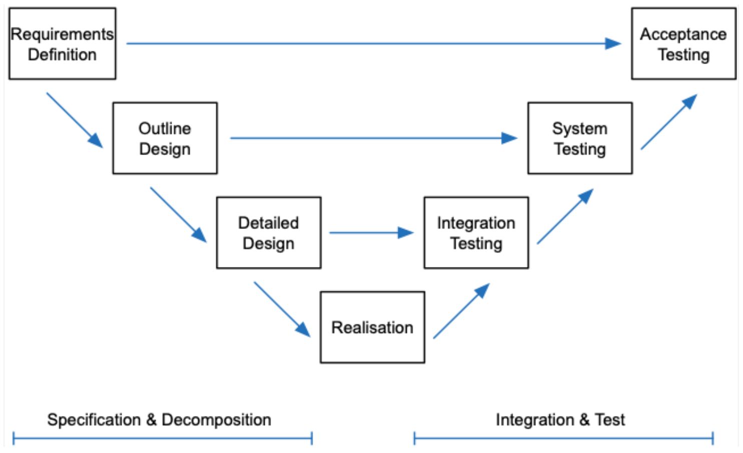


Figure 3.1: V-model framework diagram.

Design stages of the V-model The left side of the V-model will be briefly discuss here for completeness even if it's not the focus on these research, for that part let see[13]. The stages are mainly divided in three section: System Requirements Definition, System Design and Detail Design.
The System Requirements Definition is a the apex of the V-model, system requirements are defined based on the desired functionality, constraints and operational scenarios. These requirements guide the subsequent stages of the design process.
The System Design is where the system architecture is developed to meet the defined requirements. This stage includes partitioning the system into subsystems and defining their interactions. In the Detailed Design stage individual components are designed in detail, specifying algorithms, control laws, and hardware specifications.

MITL Model-In-The-Loop(MITL/MIL) simulation is the first step in verifying the control system design. In this stage, the control algorithms are implemented and tested within a simulation environment. Using high-fidelity models of the system dynamics, MIL enables the validation of control strategies without requiring physical hardware.
The key benefits in this stage are an early detection of design flaws[21], rapid iteration of control algorithms, and cost-effective testing in a virtual environment.
Simulation tools such as MATLAB/Simulink are commonly employed for MIL testing. These platforms offer extensive libraries and tools to model dynamic systems and analyse their behaviour under various scenarios.

SITL In Software-In-The-Loop(SITL/SIL) simulation, the actual control software is executed on the development computer while interfacing with the virtual system model. This stage serves as a bridge between algorithm design and embedded implementation[25].
The transition from MIL to SIL is a critical phase for identifying uncovered issues that may arise due to code generation, or hardware specific constraints. The primary objectives of SIL simulation is verifying the compatibility of control algorithms with real-time constrains, ensuring correct communication protocols and data handling, and testing software performance under various operating conditions.
Software commonly used in this stage includes platforms capable of recreating an high-fidelity environments as Gazebo, Jvamsim. These tools enable to create even complex word, including disturbances like gust or obstacle like urban structures, providing a robust environment testing control code.

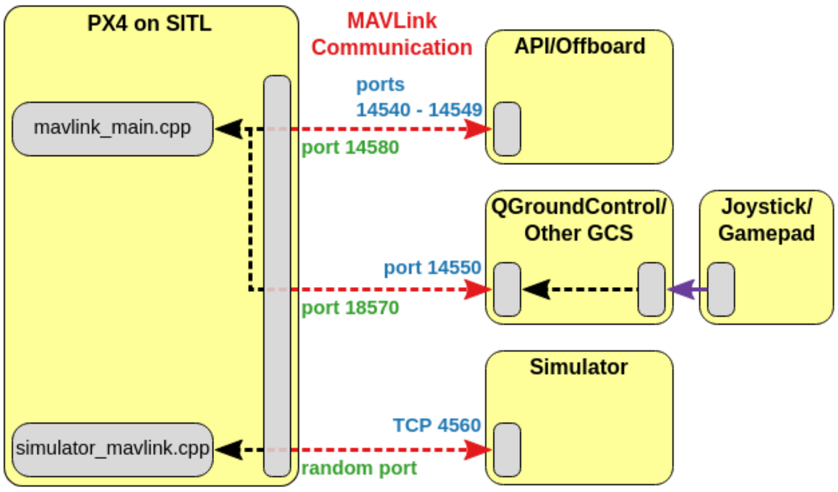


Figure 3.2: Scheme of the SIL simulation for PX4 Guide.

HITL Hardware-In-The-Loop(HITL/HIL) simulation incorporates actual hardware components into the testing loop. Here, the control software runs on the target embedded platform, interfacing with a real-time simulator that emulates the system dynamics[23].
HIL is performed because enable to realistic evaluation of the control system under near-operational conditions, validation of hardware compatibility and communication interfaces and, identification of hardware induced latency or noise effects.

HIL platforms such as dSPACE and NI PXI systems are widely used for these simulations. They provide real-time capabilities to simulate complex systems with high fidelity.

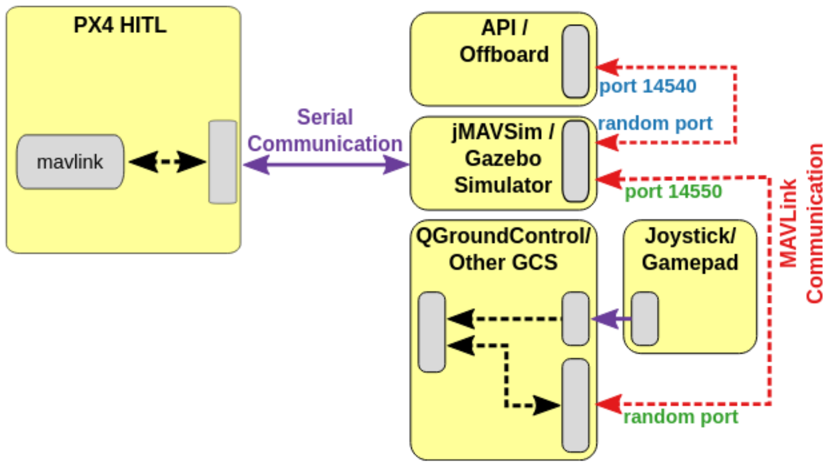


Figure 3.3: Scheme of the HIL simulation from PX4 Guide.

Experimental Flight

3.2 INDI

Incremental Nonlinear Dynamic Inversion(INDI) is an incremental control scheme for Non Linear Dynamic Inversion(NDI) control. As described in the previous chapter the NDI controller is a model-based controller so the efficiency and the final performance achieved by the control algorithm depends on the high fidelity of the mathematical model used to describe the dynamic system used to design the controller. To increase the robustness of the controller an incremental approach is used. This can be achieved by using sensor and actuator measurements for feedback and can significantly reduce dependency on accurate knowledge on UAS dynamic[2]. Using a Taylor series around (x_0, u_0) the new state vector can be describe as follow:

$$\dot{x} \simeq \dot{x}_0 + A_x \Delta x + B_u \Delta u \tag{3.1}$$

where

$$A_x = \frac{\partial(f(x) + g(x)u)}{\partial x}|_{x=x_0} \tag{3.2}$$

$$B_u = \frac{\partial(f(x) + g(x)u)}{\partial u}|_{x=x_0} \tag{3.3}$$

are the partial derivatives and $\Delta x = x - x_0$ and $\Delta u = u - u_0$ are the increments. Four important assumption needs to be done about the system to properly use an INDI controller. First one, states changes in the derivative state has to be faster than in the state, $\Delta \dot{x} >> \Delta x$. The second assumption is that the actuator dynamics is faster than the state dynamic, $\Delta u >> \Delta x$.

The third one, state derivates need to be available from sensor or estimator measurement. The last one, since the control matrix has to be inverted has to be a non-singular matrix for every state. In the quadrotor case all this assumption are always true[2].
The INDI so is no more a model-based controller but it's a sensor-based one, this enable to have a more robust control against the uncertainties through the model and the changes in the model dynamic, as a change of mass or even a change in the actuators dynamic. On the opposite site a sensor-based controller requires a precise measurements of the derivates.
A briefly theoretical dissection will follow on how to evaluate the acceleration along the $Z^{\mathbb{E}}$ axis. Let introduce $\cos\theta\cos\phi \simeq 1$, k is referred to a specific sample time, and LPF will be used to reference to a low pass filter action.
The z acceleration will be:

$$\ddot{z}_k = g - \frac{f_k}{m} + a^{\mathbb{E}}_{z,ext,k} \tag{3.4}$$

A LPF will be used to estimate the direct measurement of the previous control input, in the quadrotor case the RPM is measured and the force is evaluated from that value. So the filtered value will be:

$$f_f = LPF[T_m \cdot f_k + (1 - T_m) \cdot \bar{f}_{k-1}] \tag{3.5}$$

Where $\bar{f}_k$ is the actual thrust force, f_k is the last command of thrust, and T_m is the time constant of the filter.
Assuming also that the system has a fast dynamic, the acceleration on the Earth frame can be considered with no changes through a simple time

$$\Delta a^{\mathbb{E}}_{z,ext} = a^{\mathbb{E}}_{z,ext,k+1} - a^{\mathbb{E}}_{z,ext,k} \simeq 0 \tag{3.6}$$

$$a^{\mathbb{E}}_{z,ext,k+1} = a^{\mathbb{E}}_{z,ext,k} \tag{3.7}$$

$$a^{\mathbb{E}}_{z,ext,k+1} = \ddot{z}_k - g + \frac{f_f}{m} \tag{3.8}$$

Through this equation the INDI controller can reject the external forces caused by unmodeled dynamic of the system, because these disturbances are now measured and rejected through the incremental value in the control law. in this particular case[2] the incremental part is evaluated with a PD controller as follow:

$$\ddot{z}_{k+1} = \ddot{z}_f + \Delta\ddot{z}_k \tag{3.9}$$

$$\ddot{z}_{k+1} = \ddot{z}_f + \frac{1}{m}(f_f - f_{k+1}) \tag{3.10}$$

$$\Delta f_f = K_{vel}(K_{pos}(z_{ref} - z_k) - \dot{z}_k) \tag{3.11}$$

3.3 PX4 architecture

In the diagram below is provided an high level review of the PX4 system and each part connect to the other. There are more complex architecture where a companion computer is attached to the flight controller, also called as "mission computer". the companion computer is an external computer installed on board that is commonly used to implement more complex navigation strategies as obstacle avoidance or trajectory optimisation.

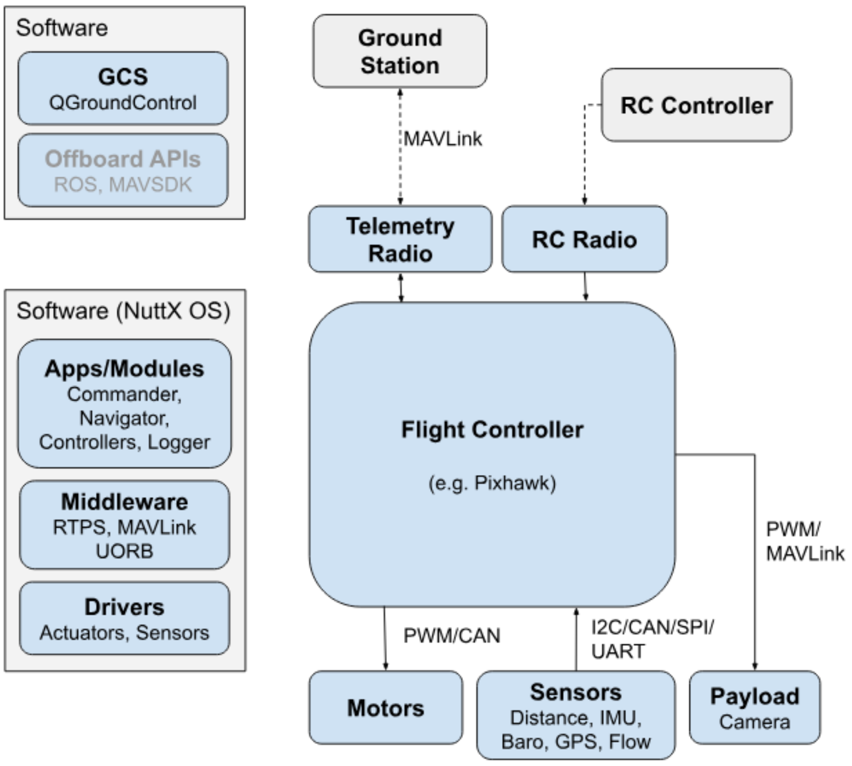


Figure 3.4: Scheme of PX4 architecture from PX4 Guide[13].

As can be seen in the diagram the communication between the Ground station and the telemetry radio on board is handled by MAVlink protocols, this data will fuse with the command from the RC radio to get the desired command to send to the flight controller.
Sensors are connect via I2C but since all the experimental flight will be evaluated in a drone arena with a Vicon system all the sensors, even the flight control internal as IMU, will be disable and all the data will be send through Wi-Fi to the drone.
The output from the autopilot in this scenario are the signal send to the motor through PWM output to the motor ESC.
In these section the focus will be more on the control architecture how the PID controller is implemented in the PX4 code. These controller is divided in three main and independent blocks, called modules. These modules communicate with each other through a uORB bus with a publish-subscribe methodology.

3.3.1 Multicopter control architecture

The control strategy implemented in the control algorithm is a cascade of P-PID controller for both of the outer and inner loop. In the PX4 code the inner loop is also divided by attitude control and rate control in two different modules because they have different operating frequency, meanwhile the position and velocity loop are in the same module.

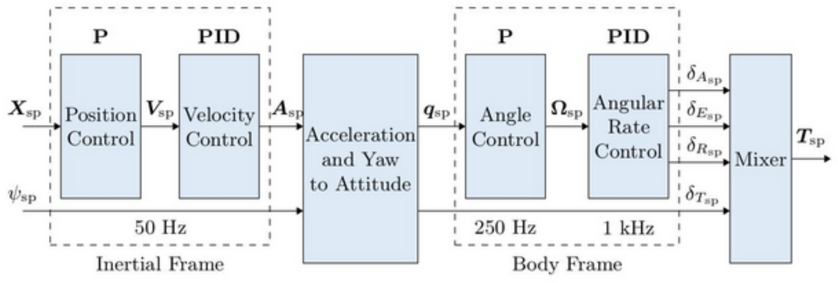


Figure 3.5: Scheme of Multicopter control architecture from PX4 Guide[13].

Navigation and position As the diagram below shows the navigation and position control block is a cascade P-PID controller, from the position error through a proportional gain the desired velocity is evaluated. The same methodology is implemented to evaluate the desired acceleration with the PID controller.
Depending on the mission mode a feedforward signal can be introduced as jerk imitator.
The operating frequency is $50Hz$

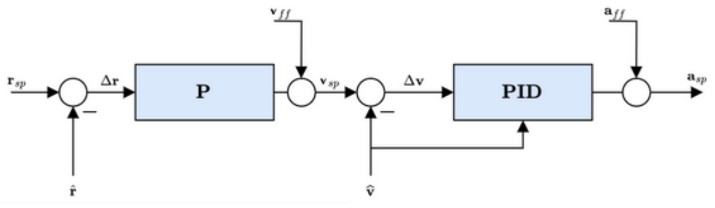


Figure 3.6: Navigation and position control architecture from PX4 Guide[13].

Acceleration to attitude In this block considering the actual yaw attitude ψ_{sp} and the acceleration A_{sp} evaluated by the position controller send as a output a desired quaternion Qsp and this value is sent to the attitude controller. The quaternion notation is introduced to avoid the gimbal lock that could happen with the Euler angle notation.

Altitude and rate control In the attitude module, that has an $250Hz$ operating frequency, a simple proportional gain P controller is introduced to evaluate the desired rate after the quaternion error is evaluated. As the diagram shows the rate command is saturated.

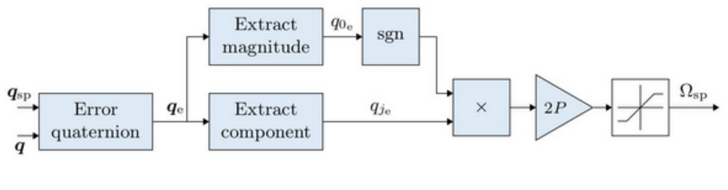


Figure 3.7: Attitude control architecture from PX4 Guide[13].

As shown for the velocity module a PID controller is implemented in the rate loop. In this block the angular acceleration are evaluated an a uORB message is publish for torque and also the

thrust from the position block and are sent to the motor mixer but before all the values are normalize respect to the maximum allocation control moment before been publish.
The operating frequency of these controller is the higher in the whole closed loop and it is set to $1kHz$.

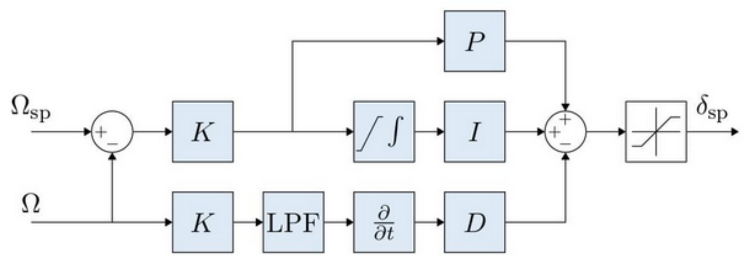


Figure 3.8: Rate control architecture from PX4 Guide[13].

4 Mathematical model of a Quad-copter

In this chapter the non-linear dynamic model of a quad-rotor is presented. The model is composed of three parts, divided in different blocks in Simulink: the kinematics block with the equation to move from an inertial frame to the body frame, and the dynamics block with Newton and Euler equation of motion and the actuator model along with the corresponding motor mixer. This chapter will therefore illustrate the equations used for the development of the model.

4.1 Attitude representation

Within the kinematics block, the reference frames used are defined: an inertial frame and a body frame. The inertial frame, known as the Earth frame, is fixed and aligned with the ground station. Its axes are defined as $E = [O, X, Y, Z]$ and are abbreviated with a superscript E. The Body frame, which is attached to the drone, is defined as $B = [CoG, x, y, z]$ and abbreviated with the superscript B. The chosen orientation follows the North-East-Down(NED) convention commonly used in aeronautics [16], for both of the frames, aligning the Z axis with the gravitational force g [19] .

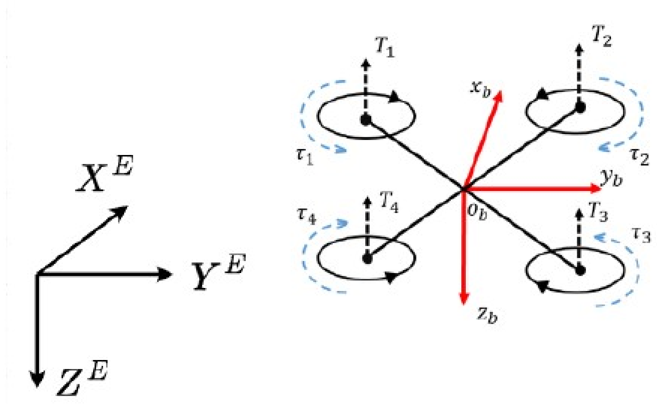


Figure 4.1: Earth and Body frames aligned in NED orientation[2].

4.2 Kinematic model

To represent the drone's attitude in space, the Euler method has been chosen with respect to the E frame, along with their derivatives for angular velocities.

$$\eta = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} \dot{\eta} = \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{4.1}$$

.
Body rates and corresponding angular acceleration have been defined as follows:

$$\Omega = \begin{bmatrix} p \\ q \\ r \end{bmatrix} \dot{\Omega} = \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} \tag{4.2}$$

A rotation matrix is used to establish a relationship between the angular rates defined in the body frame B and those of the Earth frame E.

$$\begin{bmatrix} r_x^b \\ r_y^b \\ r_z^b \end{bmatrix} = Rr_e^b \begin{bmatrix} r_x^e \\ r_y^e \\ r_z^e \end{bmatrix} = R_x(\Phi)R_y(\Theta)R_z(\Psi) \begin{bmatrix} r_x^e \\ r_y^e \\ r_z^e \end{bmatrix} \tag{4.3}$$

The selected rotation sequence follows the Yaw-Pitch-Roll convention. The rotation matrix is derived by sequentially combining the individual rotations around each of the three axes.

$$R_x(\Phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\phi & sin\phi \\ 0 & -sin\phi & cos\phi \end{bmatrix} \tag{4.4}$$

$$R_y(\Theta) = \begin{bmatrix} cos\theta & 0 & -sin\theta \\ 0 & 1 & 0 \\ sin\theta & 0 & cos\theta \end{bmatrix} \tag{4.5}$$

$$R_z(\Psi) = \begin{bmatrix} cos\psi & sin\psi & 0 \\ -sin\psi & cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.6}$$

By inverting the matrix, it is possible to derive the relationship between the angular rates in Body frame B and those in the Earth frame E.

$$\Omega = R \cdot \dot{\eta}$$

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \cos\theta\sin\phi \\ 0 & -\sin\phi & \cos\theta\cos\phi \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{4.7}$$

$$\dot{\eta} = W \cdot \Omega$$

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \tan\theta\sin\phi & \tan\theta\cos\phi \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi/\cos\theta & \cos\phi/\cos\theta \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{4.8}$$

As seen from this notation, a singularity occurs, known as Gimbal Lock, which prevents the unambiguous definition of the rotation. This happens specifically when $\cos\theta = \pi/2$. To address this issues, the PX4 attitude is defined using quaternions, which eliminate the singularity. For the purposes of this discussion, and up to the Model In The Loop(MITL) phase, Euler angles are used for simplicity, as aggressive manoeuvrers are not required.
Additionally, a rotational matrix will be employed to convert the linear velocities, as outlined

in[2]. For completeness, the full matrix is provided below given by[17].

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{z}\end{bmatrix} = \begin{bmatrix}\cos\psi\cos\theta & \cos\psi\sin\phi\sin\theta - \cos\phi\cos\sin\psi & \cos\phi\cos\psi\sin\theta + \sin\theta\sin\psi\\ \sin\psi\cos\theta & \sin\psi\sin\psi\sin\theta + \cos\phi\cos\psi & \cos\phi\sin\psi\sin\theta - \cos\psi\sin\phi\\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi\end{bmatrix}\begin{bmatrix}u\\ v\\ w\end{bmatrix} \tag{4.9}$$

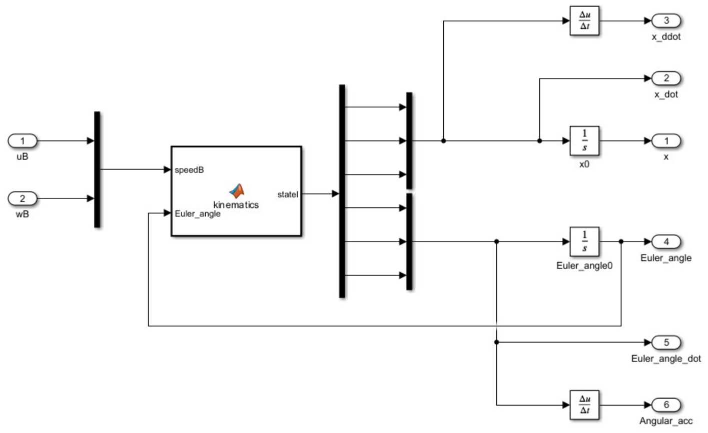


Figure 4.2: Kinematics block on Simulink.

4.3 Dynamic model

The dynamic model considers the forces, moments, masses, moments of inertia, and their associated linear and angular accelerations, investigating the relationships between these quantities. Treating the drone as a rigid body, Newton's and Euler's equations have been implemented as described in [88-90]. In this notation the Newton's and Euler's equations are presented in the Body frame to align with the standard representation often used in aeronautical engineering. For completeness, both the linear and angular equations will be provided in full, expressed in both Body and Inertial frames at the end of the chapter.

$$m(\dot{v}^b + \Omega \times v^b) = R_e^b G^e - T^b + F_{ext}^b$$

$$m\begin{bmatrix}\dot{u} + qw - rv\\ \dot{v} - pw + ru\\ \dot{w} + pv - qu\end{bmatrix} = Rb_e\begin{bmatrix}0\\0\\0\end{bmatrix} - \begin{bmatrix}0\\0\\f\end{bmatrix} + F_{ext}^b \tag{4.10}$$

In the equations presented above, v represent the linear velocities in the Body frame, g denotes the gravitational force, f denotes the force generated by the drone's propellers, and F_{ext}^b corresponds to the external forces arising from model uncertainties or, as will be discussed later, disturbances such as gusts.
The rotational dynamics are governed by Euler's equations[2].

$$J\dot{\Omega} = -\Omega \times (J\Omega) + \tau - \Gamma + M_{ext} \tag{4.11}$$

τ represents the vector of moments generated by the actuators, expressed as $\tau = [L, M, N]_T$, where Γ denotes the gyroscopic effect of the propellers[19], which resists changes in attitude.

$$\Gamma = \sum J_p \begin{bmatrix} q \\ -p \\ 0 \end{bmatrix} (-1)^{i+1}\omega_i \tag{4.12}$$

M_{ext} represents the external moments that arise from external phenomena or unmodeled dynamics.
The following are the expanded Euler equations:

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = J^{-1}(-\begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} J_{xx}p \\ J_{yy}q \\ J_{zz}r \end{bmatrix} + \begin{bmatrix} L \\ M \\ N \end{bmatrix} - J_p \begin{bmatrix} q \\ p \\ 0 \end{bmatrix} \omega_\Gamma + \begin{bmatrix} M_{x,ext} \\ M_{y,ext} \\ M_{z_ext} \end{bmatrix})$$

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} (j_{yy} - J_{xx})qr/J_{xx} \\ (j_{zz} - J_{xx})pr/J_{yy} \\ (j_{xx} - J_{yy})pq/J_{zz} \end{bmatrix} + \begin{bmatrix} L/J_{xx} \\ M/J_{yy} \\ N/J_{zz} \end{bmatrix} - J_p \begin{bmatrix} q/J_{xx} \\ -p/J_{yy} \\ 0 \end{bmatrix} \omega_\Gamma + \begin{bmatrix} M_{x,ext}/J_{xx} \\ M_{y_ext}/J_{yy} \\ M_{z_ext}/J_{zz} \end{bmatrix} \tag{4.13}$$

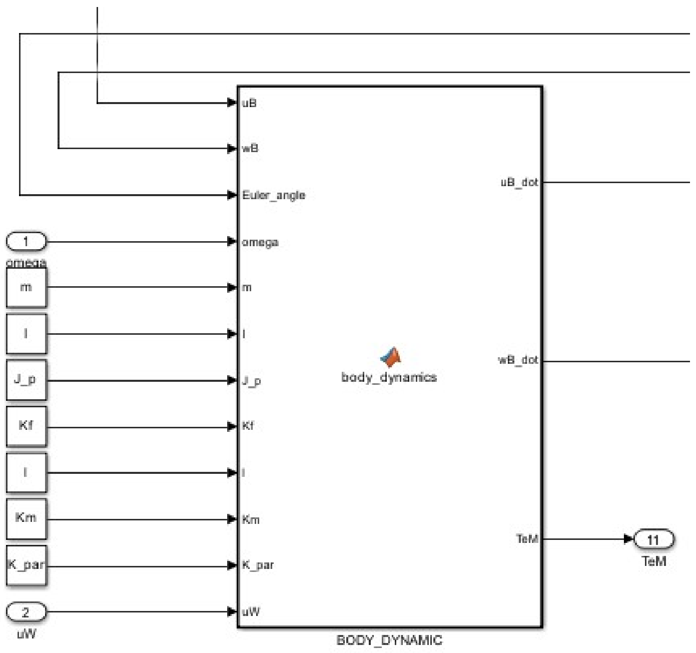


Figure 4.3: Dynamic block in Simulink.

4.4 Actuator model

Within the actuator block, two critical aspects of multicopter dynamic modelling are addressed: the motor mixer and the time constant of the motor used.
The thrust and moment generated by each propeller are expressed using dimensionless coefficient, C_T and C_M. The thrust coefficient C_T represents the lift generated by the propellers,

while the moment coefficient C_M accounts for the drag produced by propellers. These coefficients are approximated using a quadratic relationship with the propellers' angular rate.

$$f_i = C_T \omega_i^2 \tag{4.14}$$

$$M_i = C_M \omega_i^2 \tag{4.15}$$

The summation of the contributions from the i-th propellers generates the total lift of the drones.

$$f = \sum f_i = C_T(\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2 +) \tag{4.16}$$

The generation of moments depends on the differences in the angular rates of the individual propellers. For the roll and pitch moment, the arm length of the drone is also taken into account. Quad-copters are typically categorized based on the chosen propellers configuration, which can either be in an 'X' or 'cross' configuration[19]. The latter is the configuration selected for the mathematical model presented here. Consequently, the matrix defining the motor mixer specified as follows.

$$\tau = \begin{bmatrix} L \\ M \\ N \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{2}df_1 & -\frac{\sqrt{2}}{2}df_2 & -\frac{\sqrt{2}}{2}df_3 & \frac{\sqrt{2}}{2}df_4 \\ \frac{\sqrt{2}}{2}df_1 & \frac{\sqrt{2}}{2}df_2 & -\frac{\sqrt{2}}{2}df_3 & -\frac{\sqrt{2}}{2}df_4 \\ M_1 & -M_2 & M_3 & -M_4 \end{bmatrix} \tag{4.17}$$

In this context, d denotes the length of the quad-rotor's arm.
The propellers are arranged to rotate in opposing pairs, two rotate clockwise, while the other two rotate counter-clockwise. This configuration effectively cancels out the yaw moment generated by the rotors when all propellers spin at the same speed. However, when the rotational speeds differ, a yaw moment is produced. The roll and pitch moment are also generated and controlled by adjusting the relative speeds of the propellers. Unlike yaw, these moments are derived from the lift force produced by the propellers rather than the drag.
The motor mixer can be reformulated in the state-space representation as follows:

$$\dot{x} = Ax + Bu \tag{4.18}$$

$$Bu = \begin{bmatrix} C_T & C_T & C_T & C_T \\ \frac{\sqrt{2}}{2}dC_T & -\frac{\sqrt{2}}{2}dC_T & -\frac{\sqrt{2}}{2}dC_T & \frac{\sqrt{2}}{2}dC_T \\ \frac{\sqrt{2}}{2}dC_T & \frac{\sqrt{2}}{2}dC_T & -\frac{\sqrt{2}}{2}dC_T & -\frac{\sqrt{2}}{2}dC_T \\ C_M & C_M & C_M & C_M \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} \tag{4.19}$$

Here B represents the control matrix and u denotes the command vector, which corresponds to the desired rotational speeds.
Finally, for completeness, it is essential to account for the dynamics of the actuators, specifically the delay introduced by the motor in reaching the commanded velocity. This dynamic behaviour is approximated using a first-order filter, so a first-order transfer function, characterized by a time constant as modelled in the PX4 firmware.

$$\omega = \frac{1}{T_m s + 1}\omega_{com} \tag{4.20}$$

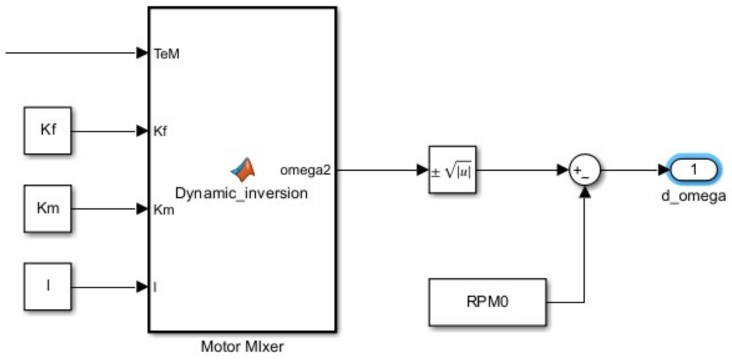


Figure 4.4: Motor mixer in Simulink.

Where T_m is the time constant.
This behaviour is attributed to parasitic current generated in the brushless motor, as well as mechanical play that introduces delays, which can destabilize the control system being defined around this model, particularly in the case of the INDI controller, which is sensitive to system delays.

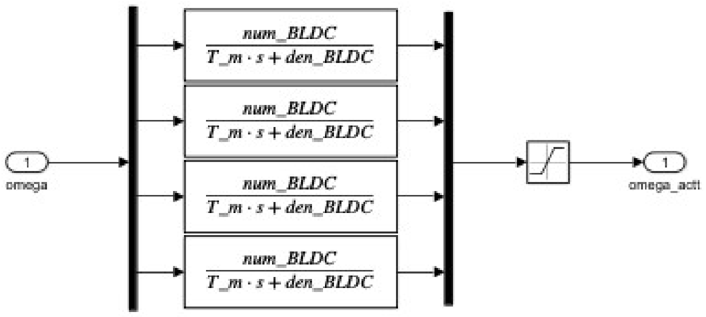

1
omega
num_BLDC / T_m · s + den_BLDC
num_BLDC / T_m · s + den_BLDC
num_BLDC / T_m · s + den_BLDC
num_BLDC / T_m · s + den_BLDC
1
omega_actt

Figure 4.5: Time constant and saturation implementation in Simulink.

4.5 Summary of the equation of motion

The following presents the equations for the complete 6 DOF system, both in the Body and Inertial frames, along with the corresponding rotation matrix[2].

$$
\begin{cases}
\dot{u} = rv - qw - g\sin\theta + \frac{f^b_{x,ext}}{m} \\
\dot{v} = pw - ru - g\sin\phi\cos\theta + \frac{f^b_{y,ext}}{m} \\
\dot{w} = qu - pv - g\cos\phi\cos\theta + \frac{f^b_{z,ext}}{m} - \frac{f}{m} \\
\dot{p} = \frac{(J_{yy}-J_{zz})qr}{J_{xx}} + \frac{L+M_{x,ext}}{J_{xx}} - J_p\frac{q}{J_{xx}}\omega_\Gamma \\
\dot{q} = \frac{(J_{zz}-J_{xx})qr}{J_{yy}} + \frac{M+M_{x,ext}}{J_{yy}} + J_p\frac{p}{J_{yy}}\omega_\Gamma \\
\dot{p} = \frac{(J_{xx}-J_{yy})qr}{J_{zz}} + \frac{N+M_{x,ext}}{J_{zz}}
\end{cases}
$$

$$\begin{cases} \ddot{x} = -\frac{f}{m}(\sin\psi\sin\phi + \cos\psi\sin\theta\cos\psi) \\ \ddot{y} = -\frac{f}{m}(-\cos\psi\sin\phi + \cos\phi\sin\theta\sin\psi) \\ \ddot{z} = g - \frac{f}{m}(\cos\theta\cos\phi) \\ \ddot{\phi} = \dot{\theta}\dot{\psi}\frac{J_{yy}-J_{zz}}{J_{xx}} - \frac{J_p}{J_{xx}}\dot{\theta}\omega_\Gamma + \frac{L}{J_{xx}} \\ \ddot{\theta} = \dot{\psi}\dot{\phi}\frac{J_{zz}-J_{xx}}{J_{yy}} + \frac{J_p}{J_{yy}}\dot{\phi}\omega_\Gamma + \frac{m}{J_{yy}} \\ \ddot{\psi} = \dot{\psi}\dot{\theta}\frac{J_{xx}-J_{yy}}{J_{zz}} + \frac{N}{J_{zz}} \end{cases}$$

$$\begin{cases} \ddot{x} = w(\sin\phi\sin\psi + \cos\psi\cos\phi\sin\theta) - v(\cos\phi\sin\psi - \cos\psi\sin\phi\sin\theta) + u(\cos\psi\cos\theta) \\ \ddot{y} = v(\cos\phi\cos\psi + \sin\phi\sin\psi\sin\theta) - w(\cos\psi\sin\phi - \cos\phi\sin\psi\sin\theta) + u(\cos\theta\cos\psi) \\ \ddot{z} = w(\cos\phi\cos\theta) - u\sin\theta + v(\cos\theta\sin\phi) \\ \ddot{\phi} = p + r(\tan\theta\cos\phi) + q(\tan\theta) \\ \ddot{\theta} = q\cos\theta - r\sin\phi \\ \ddot{\psi} = r\frac{\cos\phi}{\cos\theta} + q\frac{\sin\phi}{\cos\theta} \end{cases}$$

5 Parameter identification

After defining the key elements required for developing the quad-copter's dynamic model, it is essential to determine the values of the aerodynamic coefficients, masses, and other relevant parameters.

5.1 Thrust and Torque curve

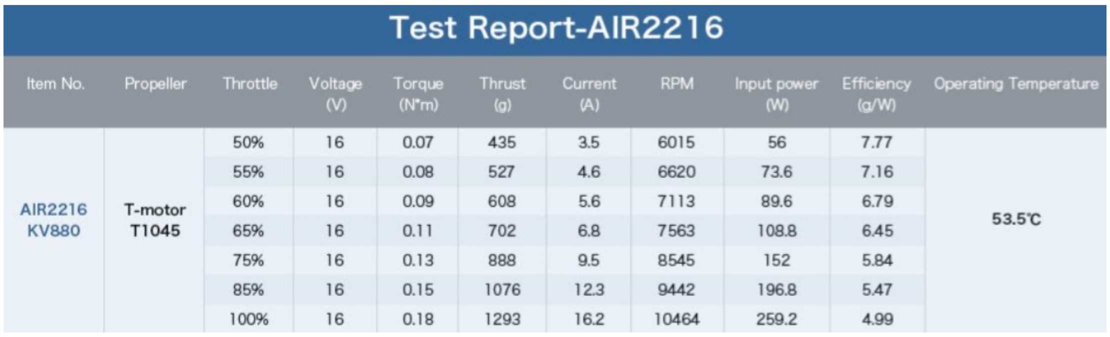
The aerodynamic coefficients of the propellers can be determined either through theoretical modelling or experimental testing using a test bench [19]. In this study, the required data for the T-MOTOR T1045 propellers and T-MOTOR Air 2216 KV880 motor were already available[13].

Figure 5.1: T-MOTOR T1045 propellers[2].

Figure 5.2: T-MOTOR Air 2216 880KV motor[2].

At this stage, interpolation was used to determine the value of C_T that satisfies the following relation:

$$T = C_T \omega^2 \tag{5.1}$$



Test Report-AIR2216

Item No.	Propeller	Throttle	Voltage (V)	Torque (N*m)	Thrust (g)	Current (A)	RPM	Input power (W)	Efficiency (g/W)	Operating Temperature
AIR2216 KV880	T-motor T1045	50%	16	0.07	435	3.5	6015	56	7.77	53.5℃
		55%	16	0.08	527	4.6	6620	73.6	7.16	
		60%	16	0.09	608	5.6	7113	89.6	6.79	
		65%	16	0.11	702	6.8	7563	108.8	6.45	
		75%	16	0.13	888	9.5	8545	152	5.84	
		85%	16	0.15	1076	12.3	9442	196.8	5.47	
		100%	16	0.18	1293	16.2	10464	259.2	4.99	

Figure 5.3: Simple of the experimental data-sheet[2].

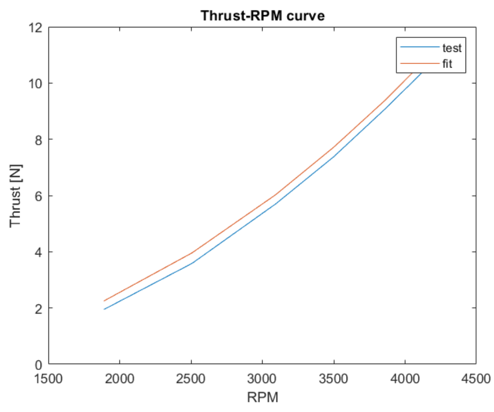
Using the same methodology, the moment coefficient C_M was also determined to satisfy the following relation:

$$M = C_M \omega^2 \tag{5.2}$$

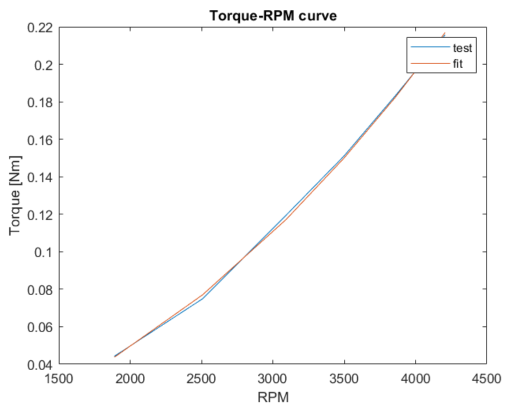
The values obtained are the following:

$$C_T = 1.055 \cdot 10^{-05} \quad C_M = 1.375 \cdot 10^{-07} \tag{5.3}$$

Where ω is the value of the RPM converted in rad/s.
In the imagine below, the experimental and approximated Thrust-RPM and Torque-RPM curves are shown.



(a) Thrust-RPM curve.

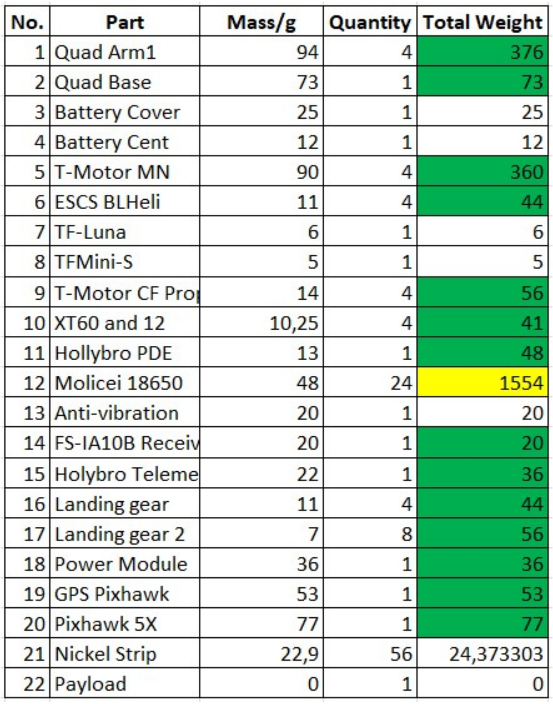


(b) Torque-RPM curve.

5.2 Mass and Inertia

To estimate the mass and inertia of the quad-rotor, two reference models were considered. The first is derived from[13], which provides an accurate weight table for the components used but lacks data on drone's inertia.

$$Total\ weight = 2.966\ kg \quad Total\ weight\ less\ battery = 1.800\ kg$$
$$Implemented\ weight = 1.708\ kg \quad weight\ diff_{\%} = 5.12\%$$



No.	Part	Mass/g	Quantity	Total Weight
1	Quad Arm1	94	4	376
2	Quad Base	73	1	73
3	Battery Cover	25	1	25
4	Battery Cent	12	1	12
5	T-Motor MN	90	4	360
6	ESCS BLHeli	11	4	44
7	TF-Luna	6	1	6
8	TFMini-S	5	1	5
9	T-Motor CF Pro	14	4	56
10	XT60 and 12	10,25	4	41
11	Hollybro PDE	13	1	48
12	Molicei 18650	48	24	1554
13	Anti-vibration	20	1	20
14	FS-IA10B Receiv	20	1	20
15	Holybro Teleme	22	1	36
16	Landing gear	11	4	44
17	Landing gear 2	7	8	56
18	Power Module	36	1	36
19	GPS Pixhawk	53	1	53
20	Pixhawk 5X	77	1	77
21	Nickel Strip	22,9	56	24,373303
22	Payload	0	1	0

Figure 5.5: Weight table.

The data highlighted in green represent the values that were correctly implemented in the model. Conversely, the data not highlighted was not directly integrated but instead retained the default values from CAD model. The data related to the batteries, highlighted in yellow, was scaled down to one-fourth of the value listed in the table. This adjustment was made because the battery pack specified in the table was used for optimization test and is over-sized for the objectives of this study[11]. The weight data was imported into the CAD model to obtain an estimate. As shown in the image below, the battery pack and the autopilot were incorporate as a single block positioned beneath the drone's structure, approximating the geometry of the components.

$$Total\ mass\ evaluated\ in\ CAD\ model : 2.06kg$$
$$Total\ inertia\ matrix\ evaluated\ in\ CAD\ model : [0.0293; 0.0293; 0.0554]^T$$

5.3 Time constant

Due to the inability to conduct testes and lacking data from specification for identifying the motor time constant, the time constant of the Iris quadcopter was used instead. The Iris is the default drone for simulation with PX4.
RPM saturation has been taken from the data-sheet specification.

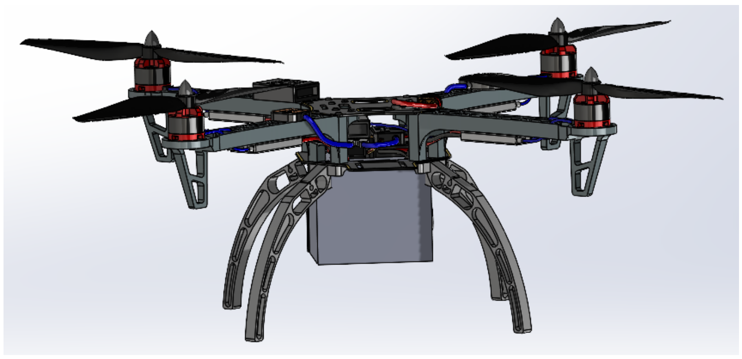


Figure 5.6: CAD used to evaluate mass and inertia.

6 Model in the loop

As mentioned in $[2.3]$, the first phase of simulation and controller tuning was carried out in MATLAB and Simulink, where the mathematical model $[4.5]$ was implemented. Sections dedicated to navigation, control, and sensors were added to the model to achieve the model in the loop simulation(MIL). The navigation and sensors block are simplified version based on the works in [16][2][5], but were implemented to provide an initial estimate of the values, which will be updated and refined as the the V-model is progressed.

6.1 Control block

The controller block was designed to be easily compatible with the PX4 PID structure$[3.3]$, thus following its architecture. It is composed of attitude control, rate control ,commonly both together are also called as inner loop, and position control, commonly called also as outer loop, components.

Position control As mention in $[3.2]$ the INDI controller regulates the acceleration and enable to use a linear controller to evaluated the acceleration stabilizing the dynamic behaviour of the system, so has been decide to implement the same control strategy of the PX4 code. It has been implemented a proportional gain P for the position error and a PD controller instead of the PID controller for the velocity to have a faster dynamic. The position control is based on following the equation:

$$\Delta\ddot{x}_{control} = K_{vel}(K_{pos}(x_{des} - x_{est,prev}) - \dot{x}_{est,prev}) - K_{acc}\ddot{x}_{est,prev} \tag{6.1}$$

Where x_{des} is the desired position, generated by the navigation block, while $x_{est}, \dot{x}_{est}, \ddot{x}_est$ represent the estimated velocities from the previous cycle[3.2]. The acceleration is directly multiplied by a proportional gain in order to reduce the noise generated by the measurement of the acceleration itself, same tests have been done during the software in the loop simulation, that are not been reported here but that show as this is best configuration to reduce the noise from the sensors. Each variable is preceded by a block that adjusts the frequency to match the default frequency of the PX4 code block[3.3].

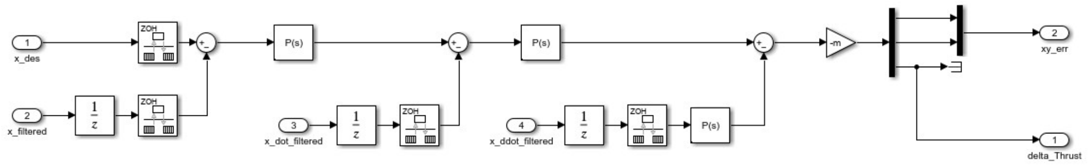


Figure 6.1: Position controller on Simulink.

The block outputs two values the desired acceleration along Z_E axis, and estimated desired accelerations along the X_E and Y_E axes. These two latter values play a crucial role, as they will be used to estimate the necessary rotation that must be applied to properly align the system with the desired position. This process ensures that the quad copter can accurately track and follow the intended trajectory.

Acceleration to attitude block The desired angles are estimated through the following equations, which are based on the acceleration calculated by the previous block and the yaw angle of the drone. These equations take into account both the dynamic behaviour of the quad-rotor and its current orientation, ensuring that the desired attitude adjustment are accurately computed. The results of these calculations are essential for determining the correct orientation needed to track the desired position.

$$\begin{bmatrix}\phi_c \\ \theta_c\end{bmatrix} = -\frac{m}{f_{est}} \begin{bmatrix}\sin\psi - \cos\psi \\ \cos\psi \sin\psi\end{bmatrix} \begin{bmatrix}\ddot{x}_c \\ \ddot{y}_c\end{bmatrix} \tag{6.2}$$

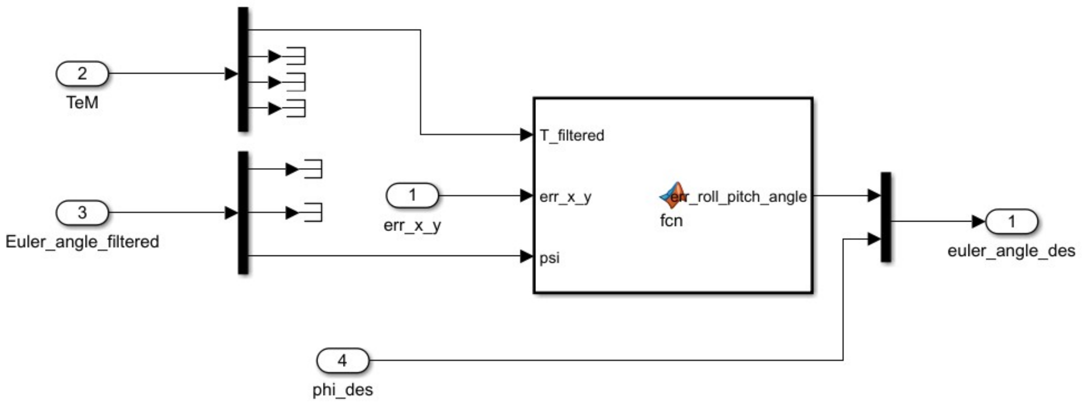


Figure 6.2: Acceleration to attitude in Simulink.

At the end of the block, as illustrated in the figure above, the desired Euler angles array is generated by incorporating the define desired yaw angle into the function output. In this simulation, a dedicated block has not been implemented for the yaw angle. As a result, all simulations are conducted with a fixed desired yaw angle. As mention in $[3.3]$ the PX4's yaw is not fixed so during the SITL yaw control gain could be retune to better values.

Attitude control This architecture is maintained even through the attitude control is implementation as a simple proportional gain that generates the desired angular rates. As mentioned in $[3.3]$, attitude representation in PX4 uses quaternions. Within the controller, the difference between quaternions is computed and then converted to a three-dimensional array, representing the desired angular rates.
However, for a simplified implementation in Simulink, where Euler angle are used, quaternion based calculation are not applied. Instead, the error is directly derived from the Euler angles and used to generate the desired angular rates. This approach simplifies the implementation process in Simulink while preserving the core functionality of the controller, allowing for effective testing and validation within the simulation environments.
The attitude control law is implemented has been made in[2]:

$$\Omega_{des} = K_{att}(\eta_{des} - \eta_{prev}) \tag{6.3}$$

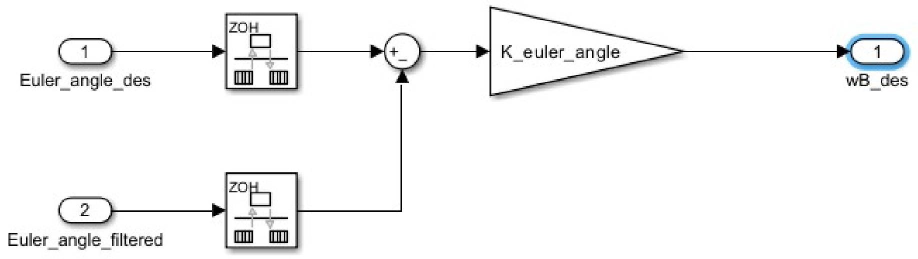


Figure 6.3: Attitude control in Simulink.

Rate control As for the velocity control a PD controller has been implemented to evaluate the desired angular acceleration. Also here the operating frequency has been change to $1kHz$ the same of the default PX4 controller.
The follow control law has been implemented:

$$\Delta\dot{\Omega}_{des} = K_{rate}(\Omega_{des} - \Omega_{prev}) - K_{acc}\dot{\Omega}_{prev} \tag{6.4}$$

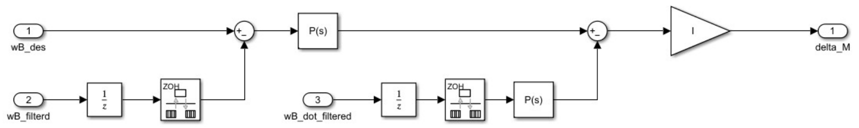


Figure 6.4: Rate controller on Simulink.

INDI implementation In this block the INDI explained in the previous chapter is implemented. The incremental input evaluated through the linear control is added to the previous control output. Previous thrust and torque comes directly from the sensor block where a RPM is simulated with the same filter used for the sensors, to synchronize the delays on the feedback line. Here INDI control implemented is the following:

$$M_{k+1} = M_k + \Delta M_k \tag{6.5}$$

Where $M_k = [T, M_x, M_y, M_z]^T$ and ΔM_k is evaluated in the previous block. The control force

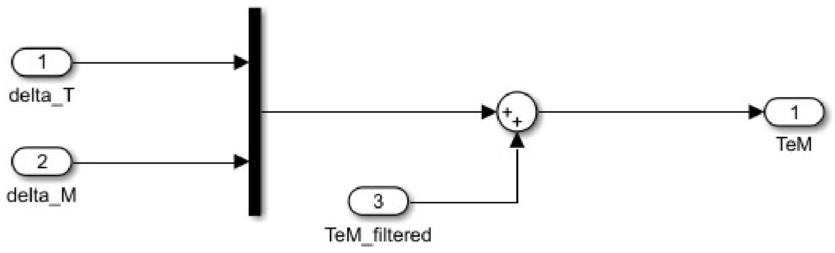


Figure 6.5: INDI controller on Simulink.

and moments are sent to the motor mixer, that will send it to the plant block, in the MIL simulation has not been included the PWM module.

6.2 Navigation block

From this block, the desired trajectories to be tested are generated. Specifically, two types of trajectories were tested: a circular trajectory and a figure-eight trajectory. These trajectories were chosen to evaluate the system's ability to handle both smooth and continuous motion on a simple pattern.

$$\begin{bmatrix} x_{des} \\ y_{des} \\ z_{des} \end{bmatrix} = \begin{bmatrix} 4\sin(\frac{t}{6}(1-\exp-\frac{t}{5})) \\ 4\sin(\frac{t}{6}(1-\exp-\frac{t}{5}))\cos(\frac{t}{6}(1-\exp-\frac{t}{5})) \\ -1.5 \end{bmatrix} \tag{6.6}$$

An exponential function has been incorporated into the equation for the figure-eight trajectory to ensure a smooth and continuos transition between the hover conditions and the desired trajectory defined by the function. This modification has been made to prevent abrupt discontinuous desired values that could compromise stability and performance of the system.

$$\begin{bmatrix} x_{des} \\ y_{des} \\ z_{des} \end{bmatrix} = \begin{bmatrix} 3\sin(\frac{t}{6}) \\ 3\cos(\frac{t}{6}) \\ -1.5 \end{bmatrix} \tag{6.7}$$

6.3 Sensor block

As outlined in [16], the sensor block is designed to emulate an IMU. A second-order filter is included to reduce noise on the acceleration signals, effectively replicating the behaviour of the extended Kalman filter used in the PX4's firmware, the values for the filter has been taken from [5]:

$$H(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \tag{6.8}$$

$$\omega_n = 50 rad/s \qquad \zeta = 0.55$$

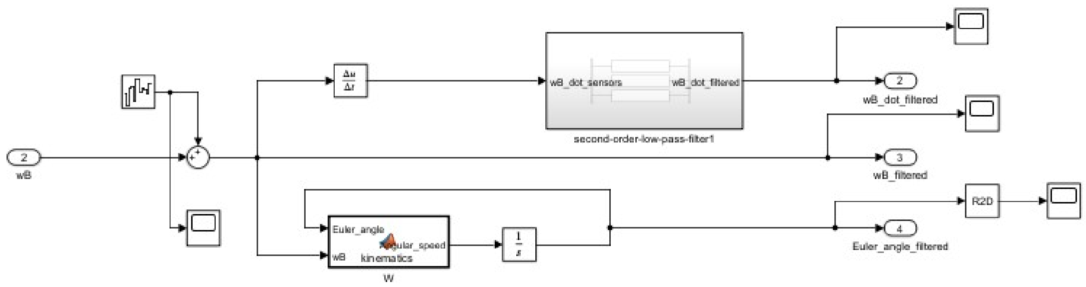


Figure 6.6: Sensor implementation for the attitude determination.

As observed $[3.2]$ and demonstrated in [16],[5] implementing an Incremental nonlinear dynamic inversion(INDI) controllers feedback on the generated torque and thrust is required. In this case, the feedback is implemented using RPM value. However, since direct measurement is not currently feasible, the signal is reconstructed based on the current derived from the PWM output. As a result, the feedback loop is not fully closed in the real implementation. This limitation will become more evident during the transition from Model-in-The-Loop(MITL) to

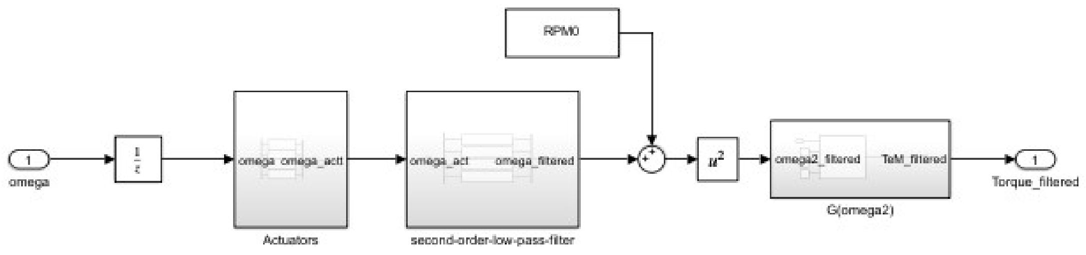


Figure 6.7: Feedback line for the RPM values.

Software-In-The-Loop(SITL) simulations, where the systems performance and the feedback loops accuracy can be further evaluated and refined.
A time constant has been added to the feedback line to synchronize the delays with the feed-forward path[2]. Additionally, the same filter used for the acceleration signals was applied to the feedback loop for completeness[16]. These adjustment ensure a more consistent and realistic simulation of the system dynamics, accounting for potential delays and noise present in a real-world scenario.

6.4 Simulation

In this chapter, a figure-eight trajectory has been simulated with the aim of evaluating the control gain values of the PD controllers required to accomplish the mission.
The following pages present two different graphs showing the states of angles, rates, positions, and velocities. The first graph compares the desired values with the values achieved by the drone, while the second graph displays the errors.
As mentioned previously, the mission trajectory begins with an exponential function, which facilitates a smooth transition from hover to trajectory without any discontinuities. However, as observed, the maximum error occurs during this initial phase of the mission. Since the focus of the MIL is not achieved a fully structured mission and the maximum error is minimum and the controller can handle it, other missions will be tested to unsure the robustness of the controller.
In the following table the control gains are shown:

	roll axis	pitch axis	yaw axis
P-angle	6	6	6
P-rate	4	4	4
P-angular acceleration	0.5	0.5	0.5

Table 1: Control gains values for attitude control loop.

	X axis	Y axis	Z axis
P-position	3	3	3
P-velocity	2	2	2
P-acceleration	1	1	1

Table 2: Control gains values for position control loop.

RMSE position [m]	RMSE angle [deg]
0.0745	0.0249

Table 3: Comparison RMSE

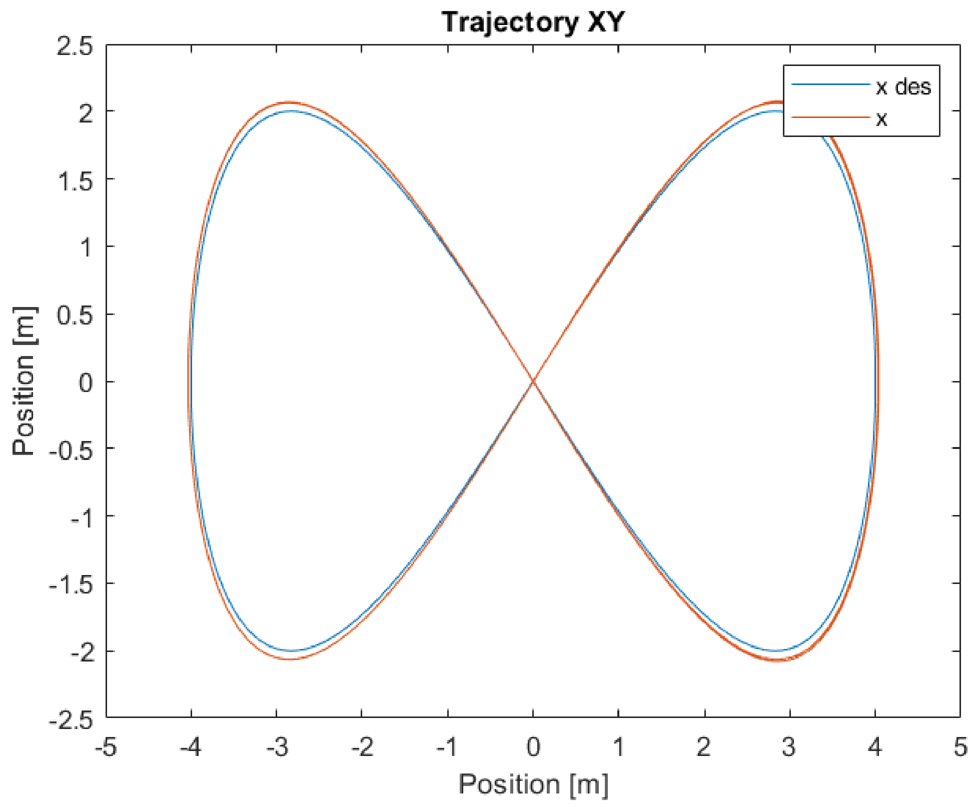


Figure 6.8: MIL trajectory.

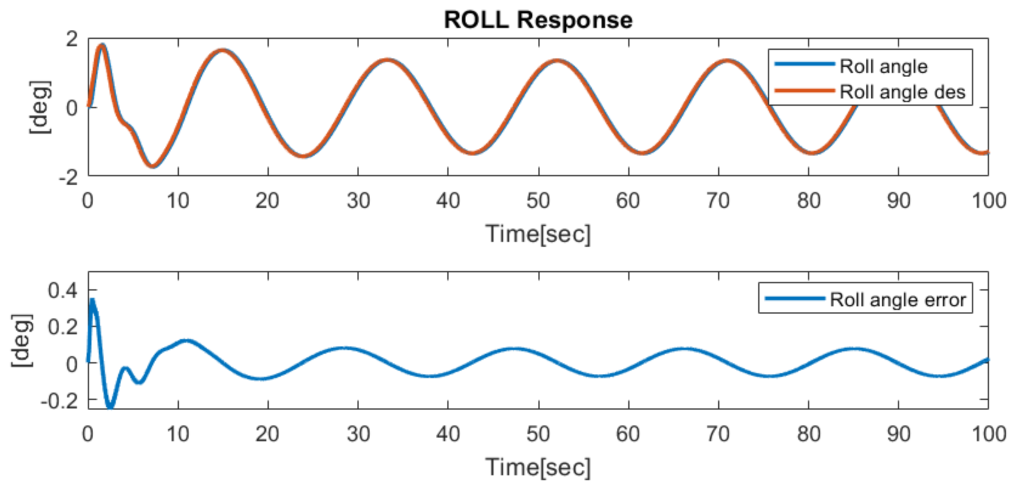


Figure 6.9: MIL roll angle response.

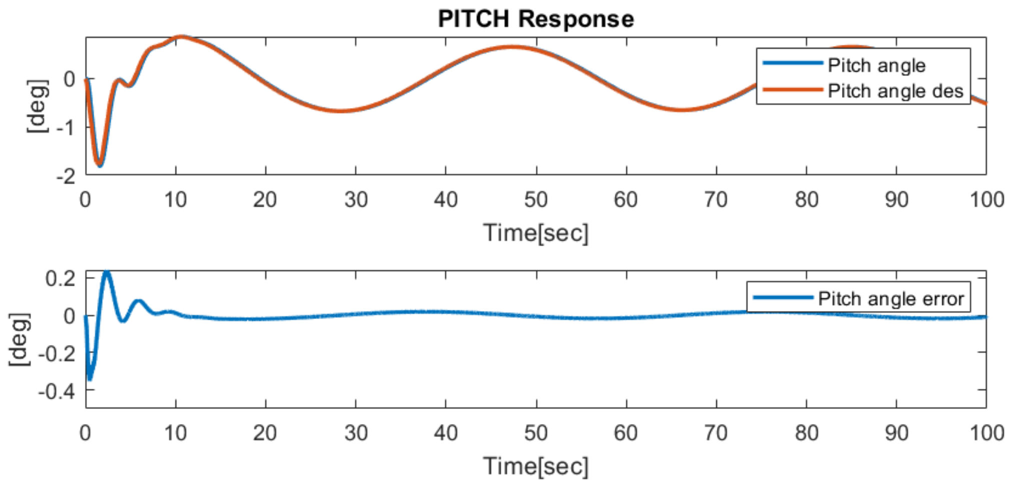


Figure 6.10: MIL pitch angle response.

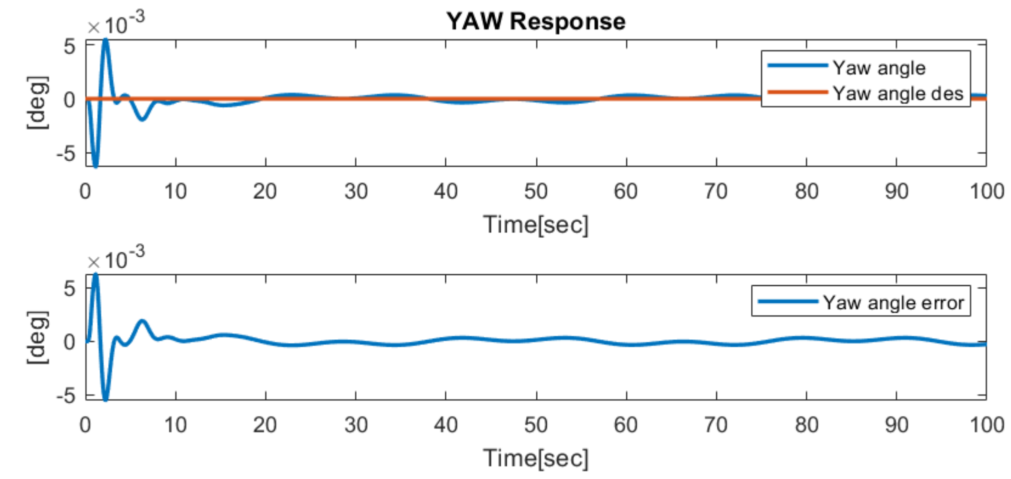


Figure 6.11: MIL yaw angle response.

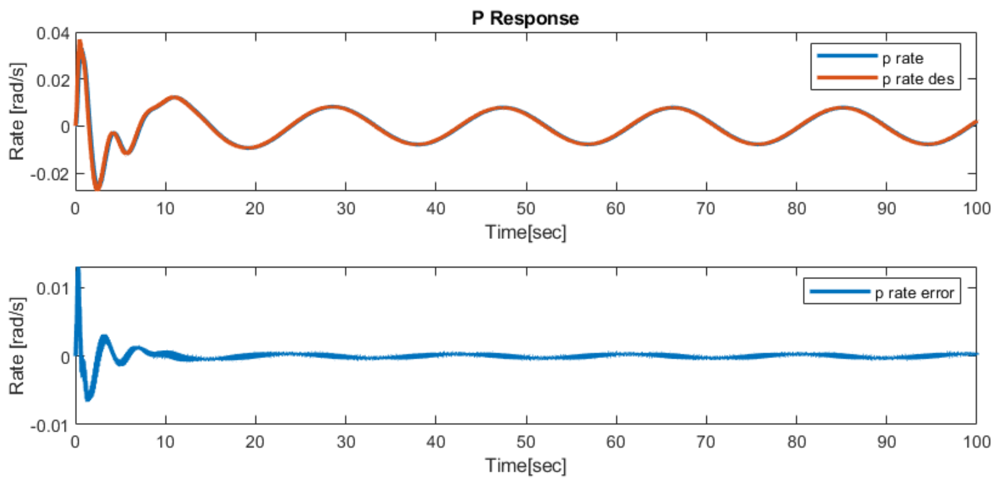


Figure 6.12: MIL p rate response.

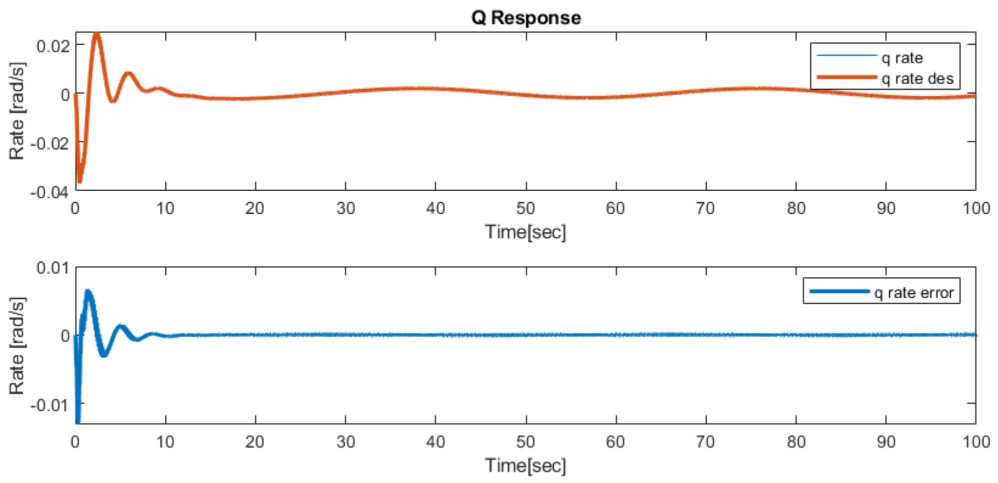


Figure 6.13: MIL q rate response.

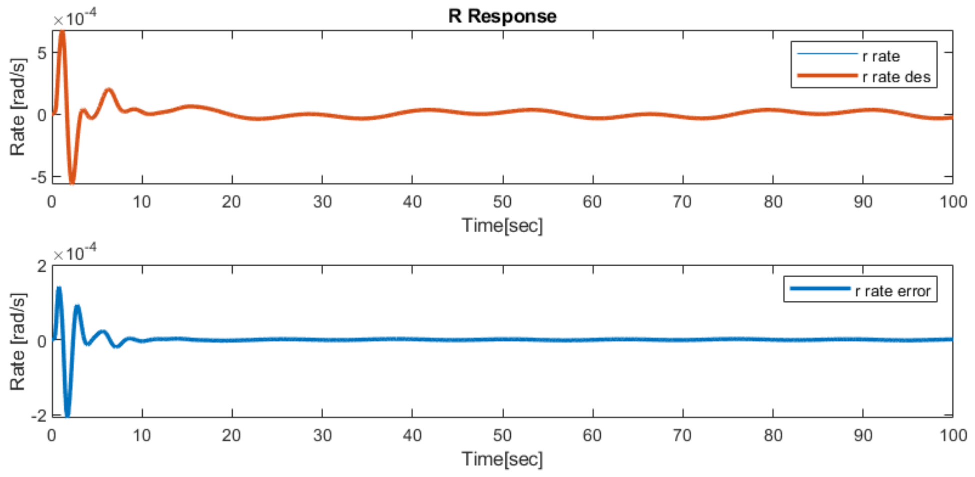


Figure 6.14: MIL r rate response.

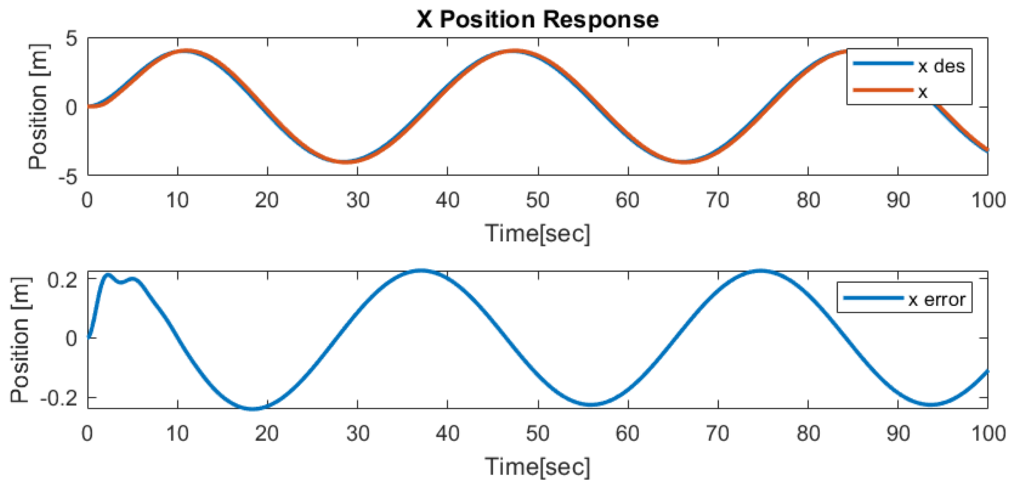


Figure 6.15: MIL X position response.

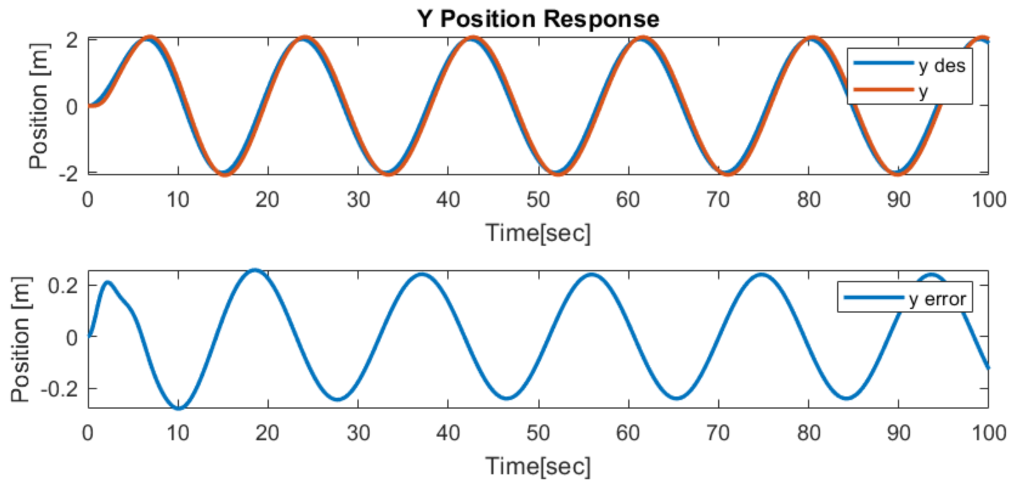


Figure 6.16: MIL Y position response.

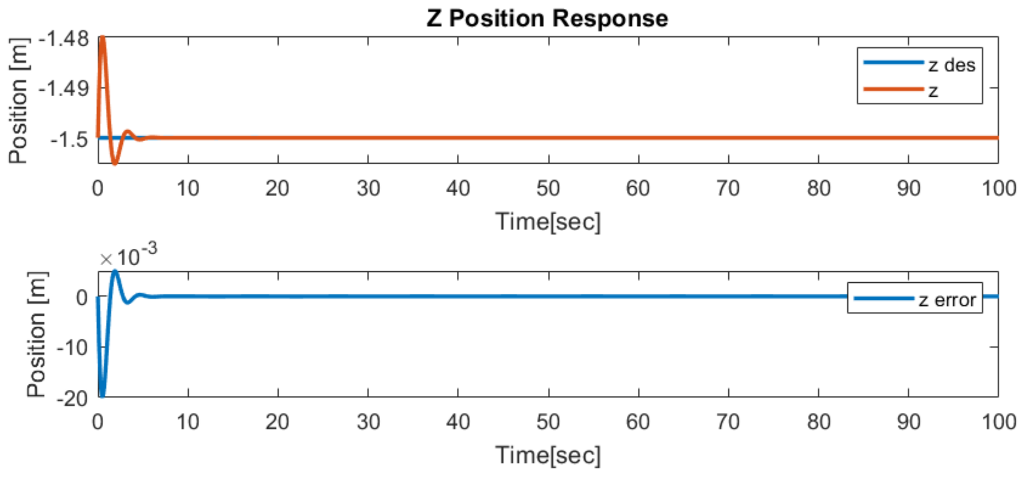


Figure 6.17: MIL Z position response.

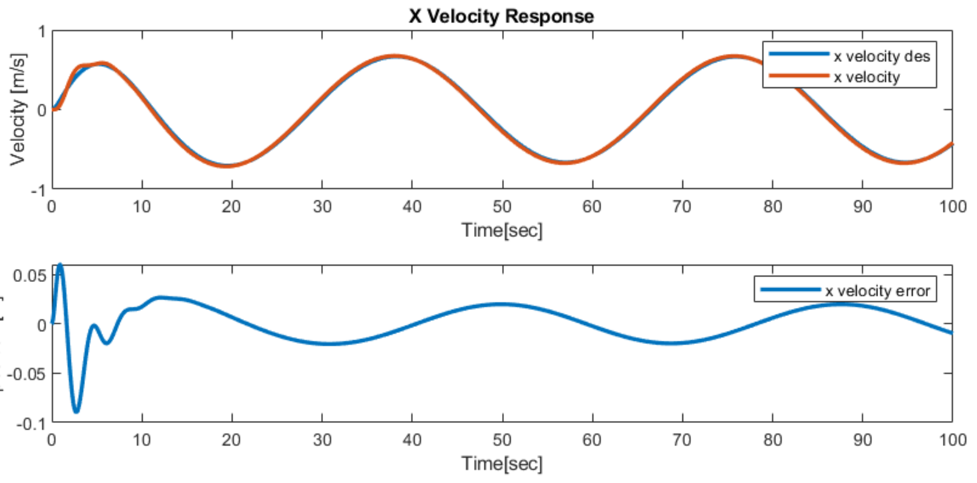


Figure 6.18: MIL X velocity response.

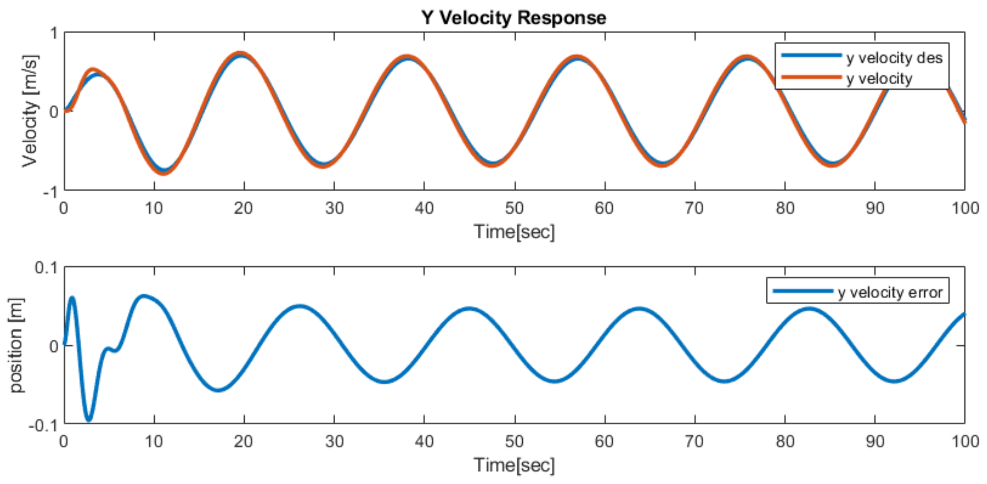


Figure 6.19: MIL Y velocity response.

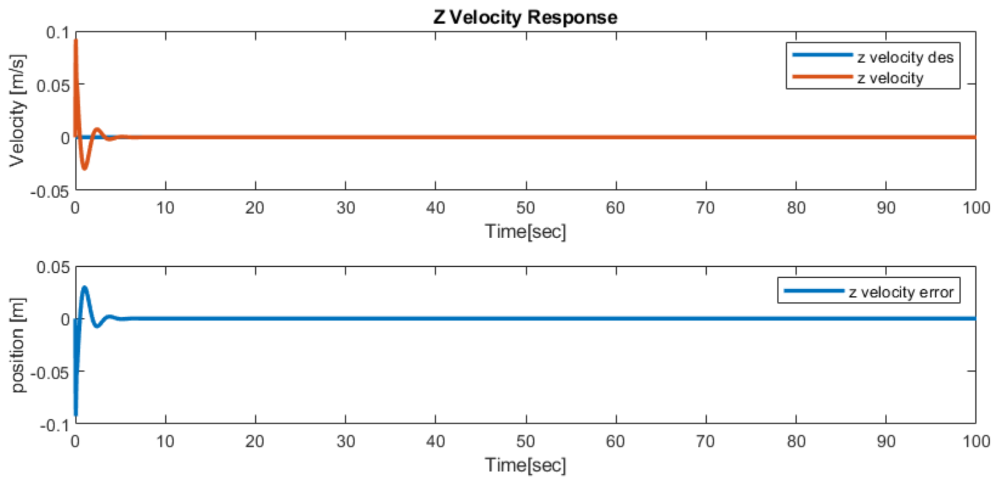


Figure 6.20: MIL Z velocity response.

6.5 Simulation with uncertainties

In this section, some of the results from simulation conducted to test the robustness of the controller are presented. The simulations were performed with consistent controller gains while varying parameters in areas where uncertainties are most likely to arise. At the end of the section a table with the root-mean-square error(RMSE) compares the different simulations.

Noise The most critical simulation involved testing the system's tolerance to sensor noise, as this will be an important consideration when transitioning from MIL to SIL. In the next phase, sensor noise and delays will be simulated with greater fidelity. In this simulation has been tested the system for the maximum noise in the sensor, particularly the IMU noise, above this limit has been seen that the response of the system is too degraded to achieve a good performance during the mission.

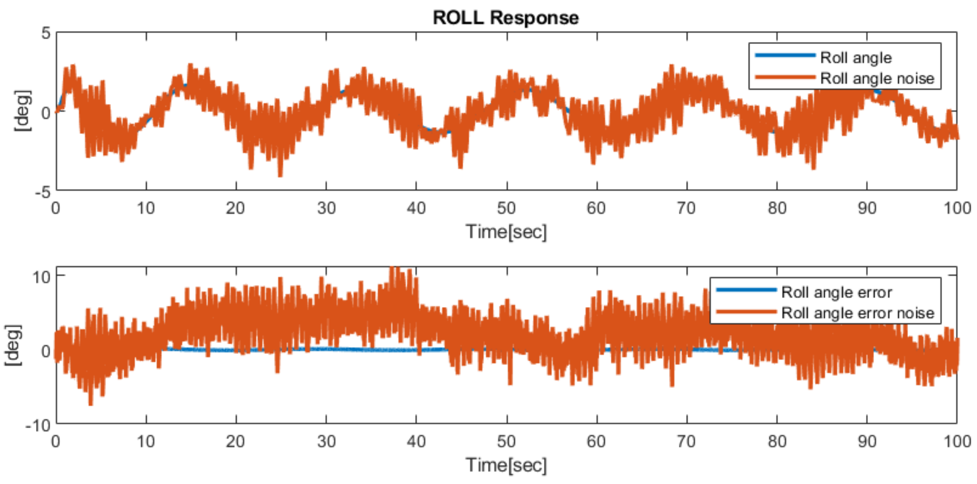


Figure 6.21: MIL roll angle response with noise disturbances.

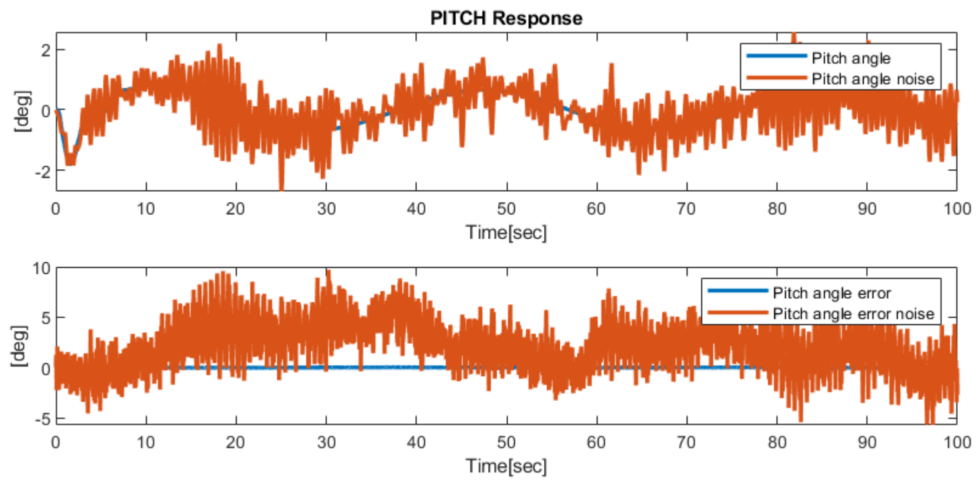


Figure 6.22: MIL pitch angle response with noise disturbances.

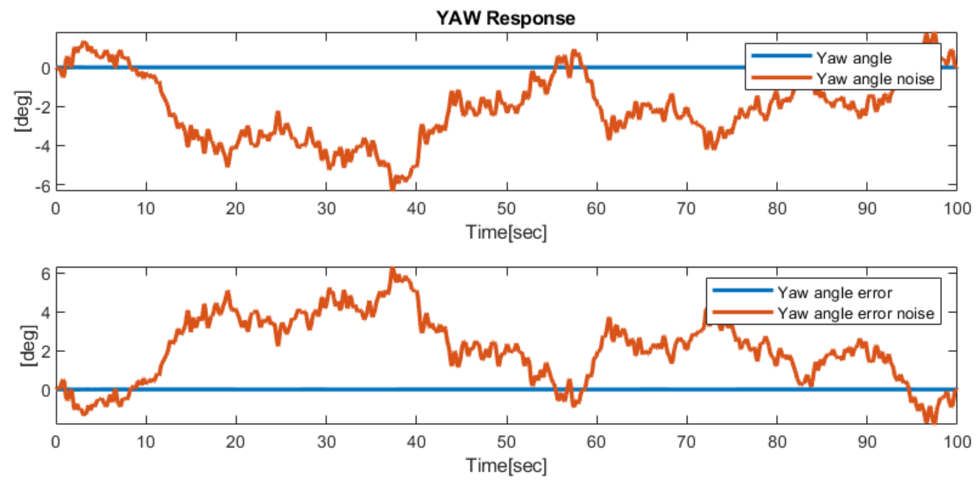


Figure 6.23: MIL yaw angle response with noise disturbances.

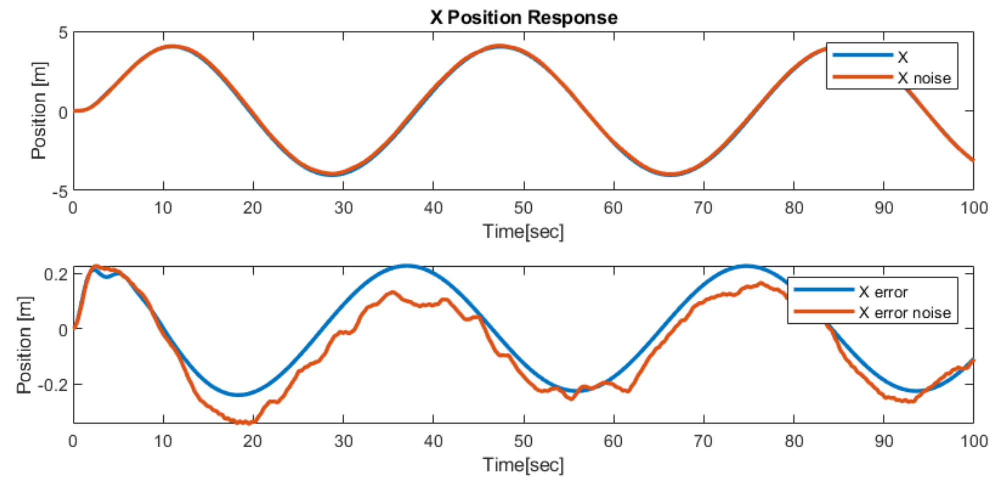


Figure 6.24: MIL x position response with noise disturbances.

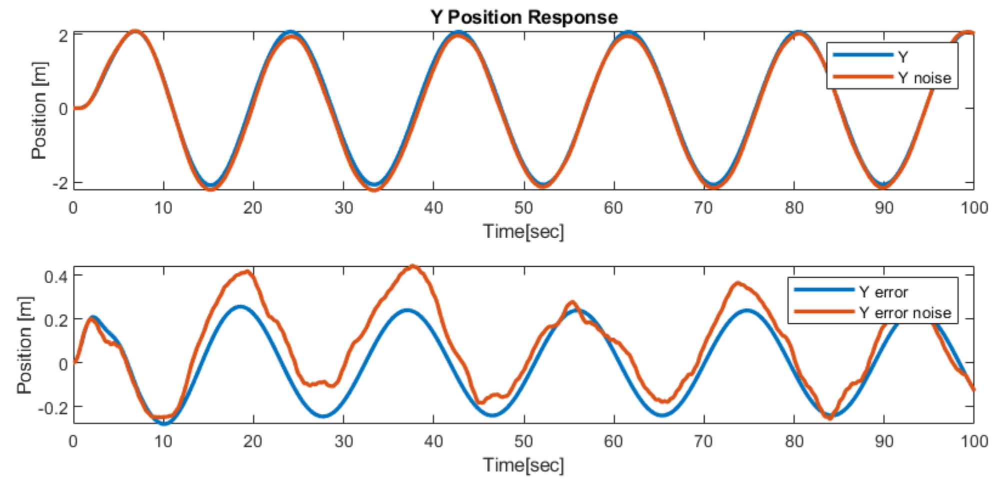


Figure 6.25: MIL y position response with noise disturbances.

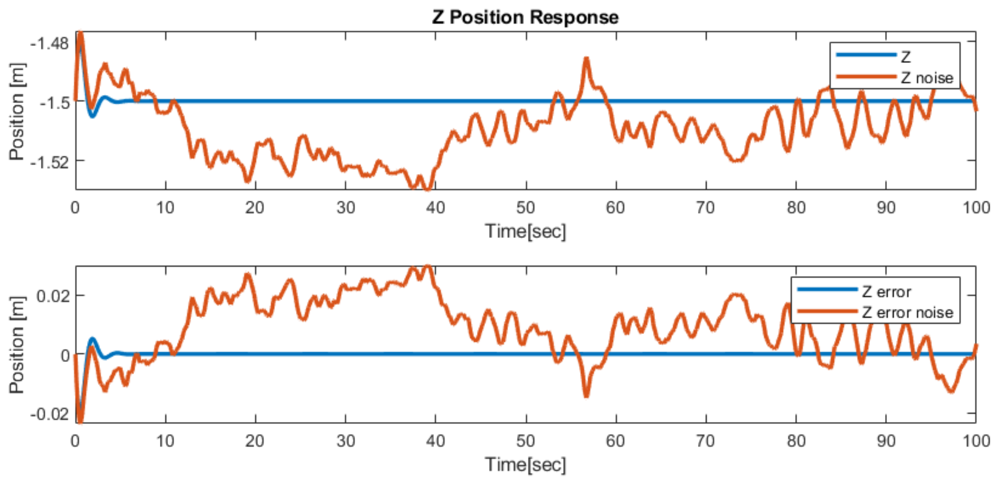


Figure 6.26: MIL z position response with noise disturbances.

Mass changes and Inertia changes Changes in mass or inertia, particularly inertia, are difficult to asses without access to a detailed CAD model, which was unavailable during testing phase. The main issue observed is that chattering disturbances arise when there is a significant difference in the inertia matrix, meanwhile the positions seam not affected by the chattering. The changes in mass and inertia have been tested to the maximum changes, $[+50\% - 25\%]mass, [+20\% - 50\%]inertia$, handled by the controller; above these values the controller was unable to accomplish the mission with satisfactory performance.

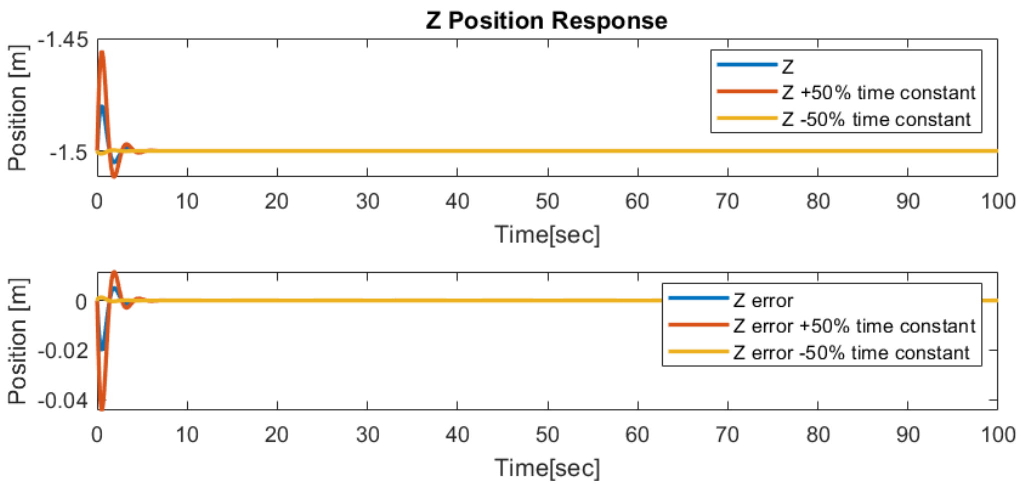


Figure 6.27: MIL Z position response with mass uncertainties.

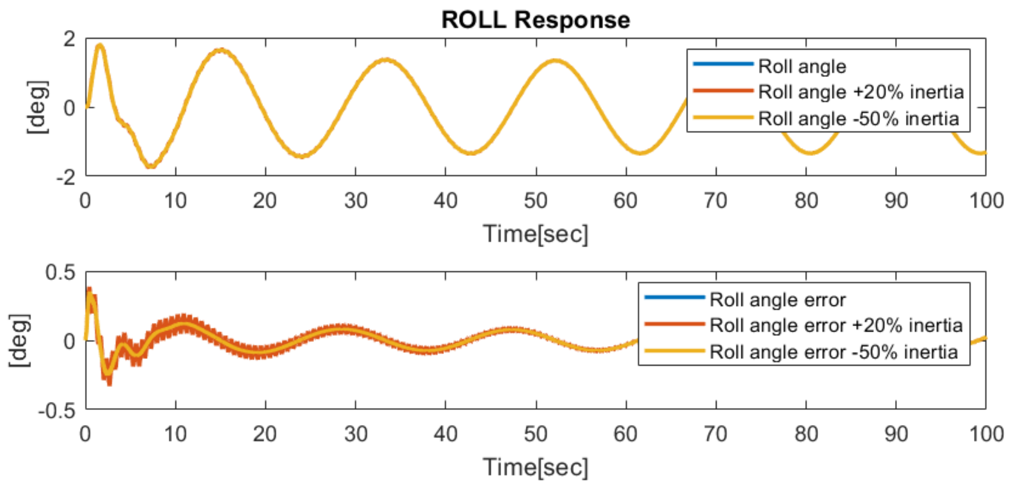


Figure 6.28: MIL roll angle response with inertia uncertainties.

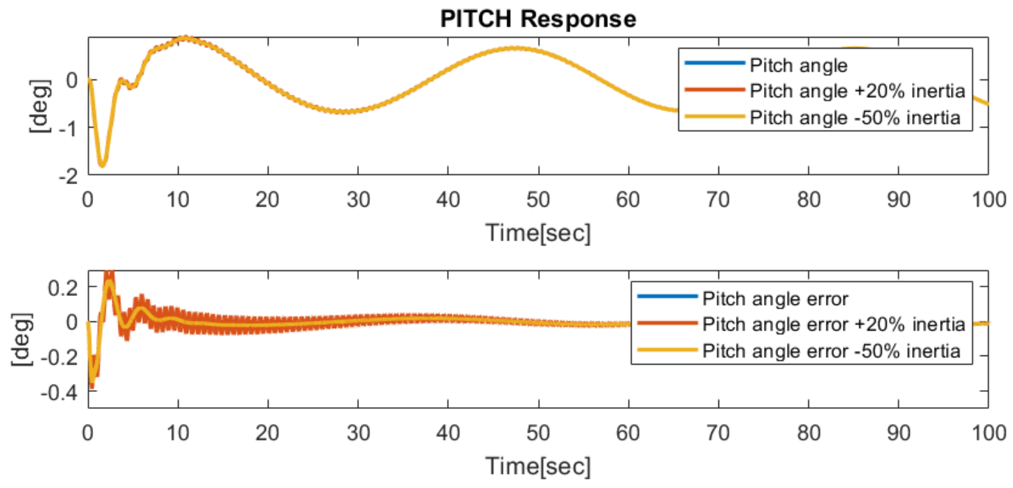


Figure 6.29: MIL pitch angle response with inertia uncertainties.

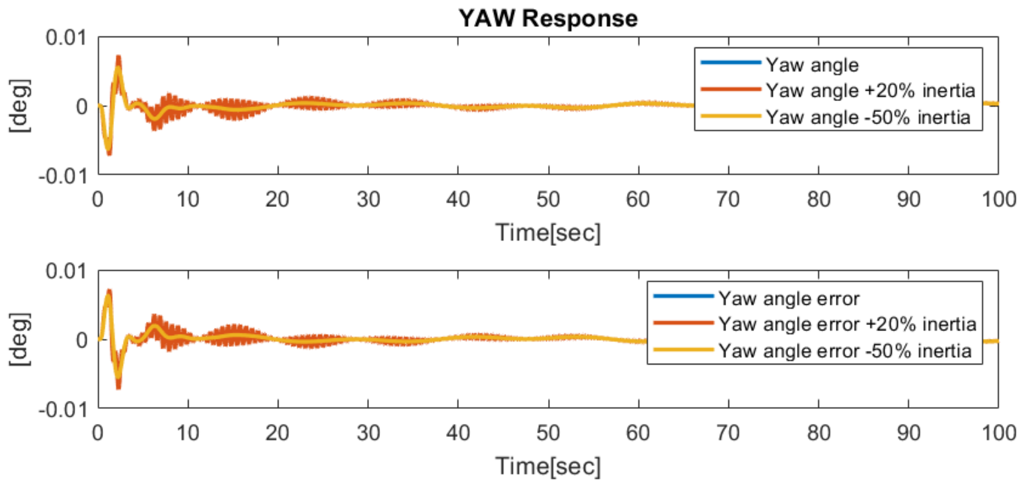


Figure 6.30: MIL yaw angle response with inertia uncertainties.

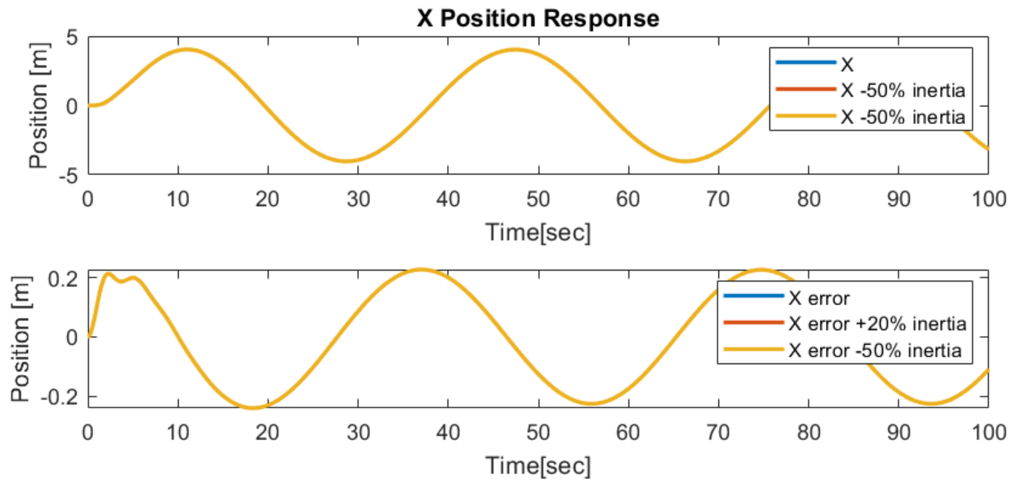


Figure 6.31: MIL x position response with inertia uncertainties.

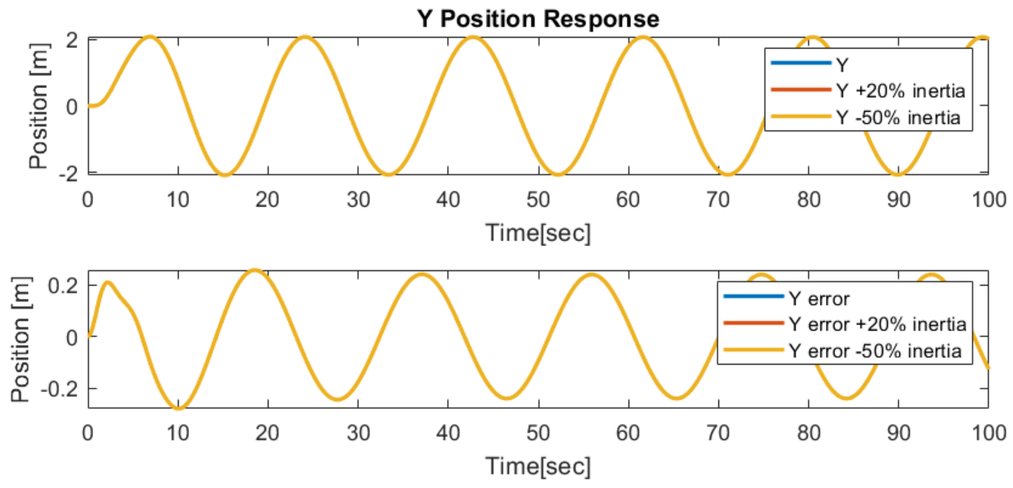


Figure 6.32: MIL y position response with inertia uncertainties.

Time constant As for the inertia in this section has been tested uncertainties for the time constant of the motors. The values are tested in $[+50\%, -50\%]$ initial time constant value. As for the inertia changes, the inner loop is more sensible to these changes.

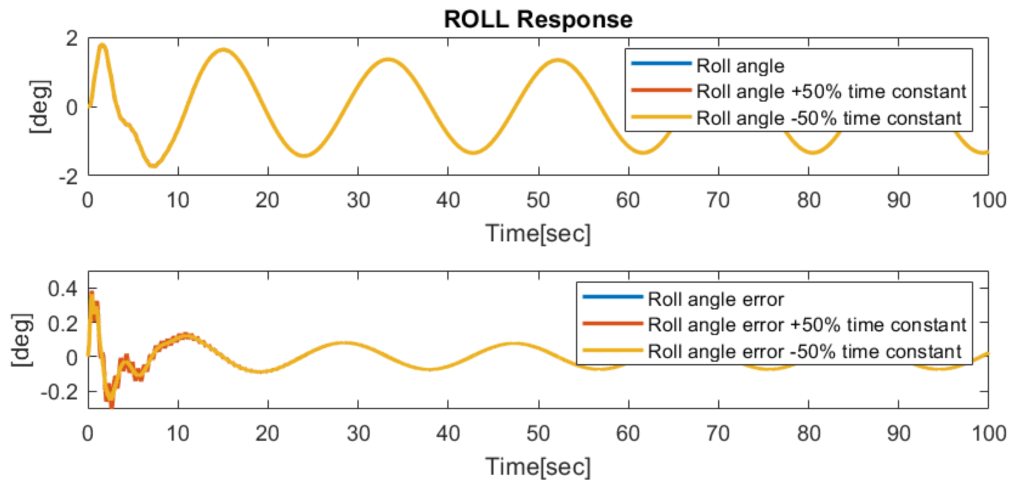


Figure 6.33: MIL roll angle response with time constant uncertainties.

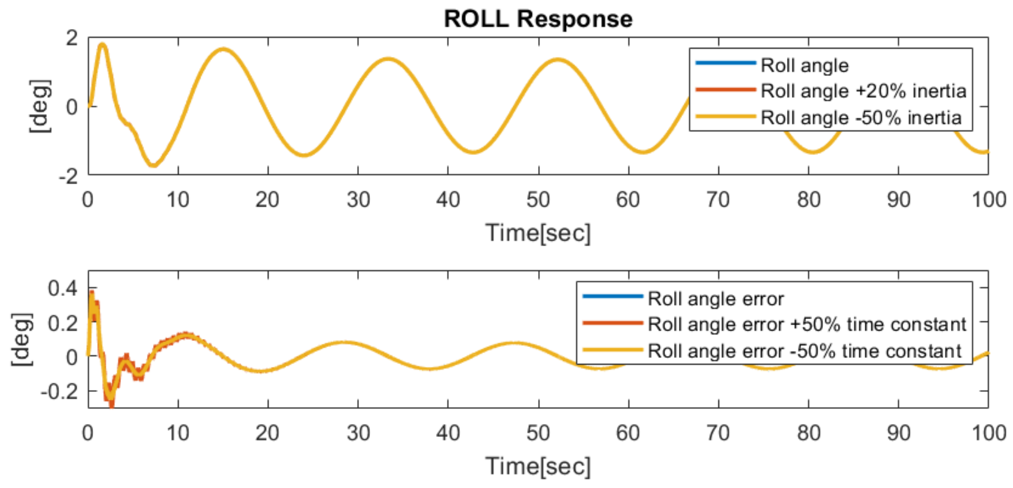


Figure 6.34: MIL pitch angle response with time constant uncertainties.

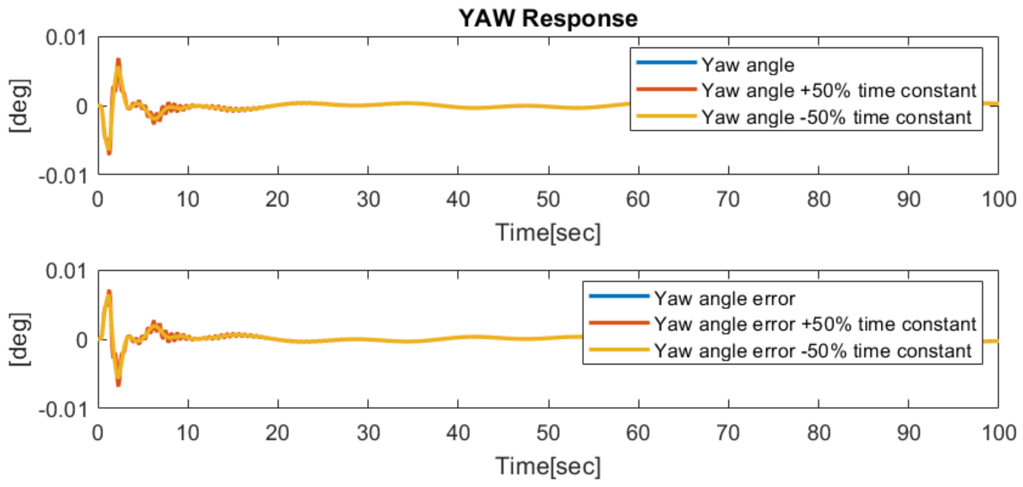


Figure 6.35: MIL yaw angle response with time constant uncertainties.

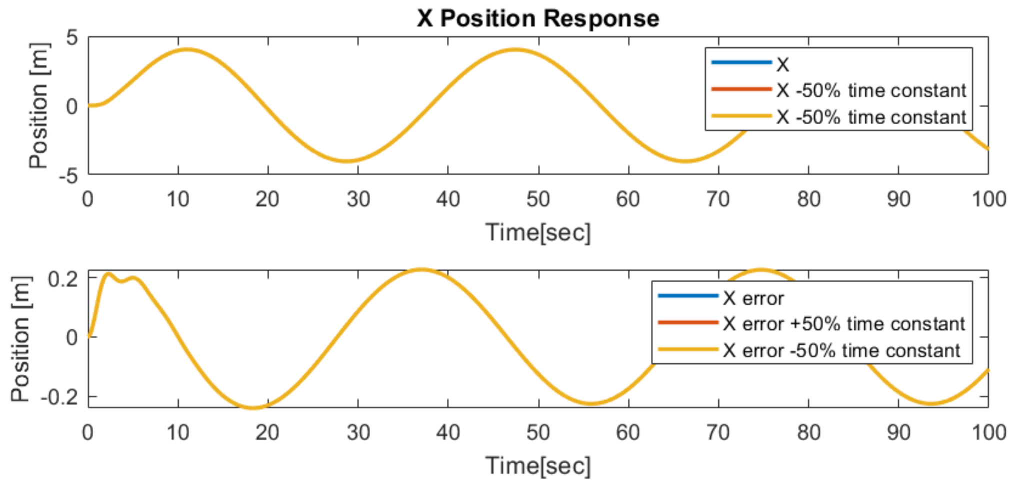


Figure 6.36: MIL x position response with time constant uncertainties.

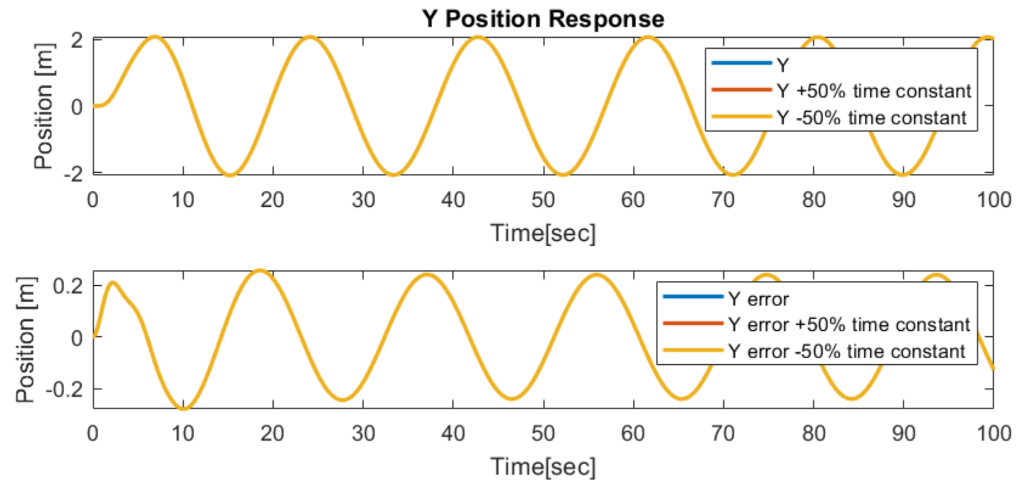


Figure 6.37: MIL y position response with time constant uncertainties.

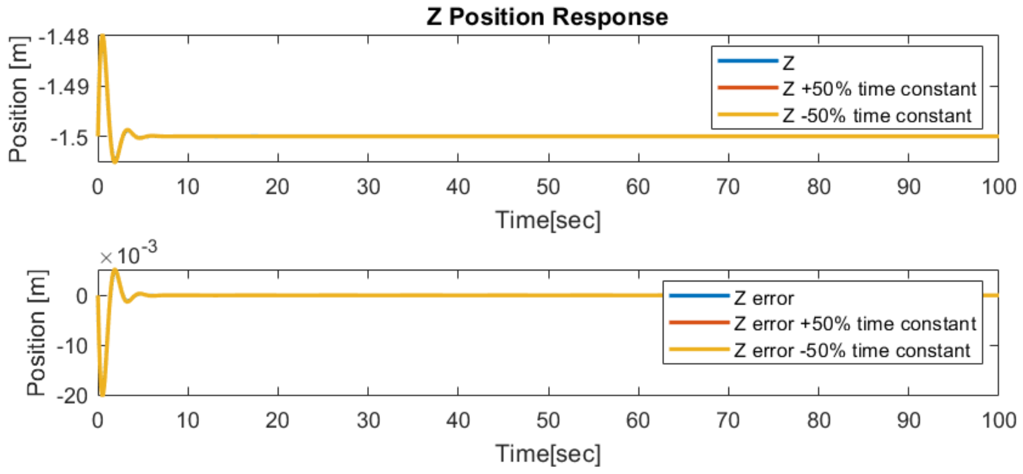


Figure 6.38: MIL y position response with time constant uncertainties.

	RMSE position [m]	RMSE angle [deg]
nominal	0.0745	0.0249
+50% mass	0.0764	0.0255
-25% mass	0.0737	0.0362
+20% inertia	0.0745	0.0272
-50% inertia	0.0745	0.0250
max noise	0.0834	1.60
+50% time constant	0.0745	0.0253
-50% time constant	0.0745	0.0249

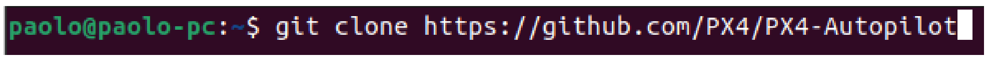
Table 4: Comparison RMSE with uncertainties

7 Software in the loop

Ones achieved the MIL simulations have been achieved, the next step is to implement the control algorithm in the PX4 code and test it, this step is commonly called Software-in-the-loop(SIL) simulation.
The objective is to provide a code that will fit in the PX4 code, this step ensures that the code is properly debugged and the controller is retuned in a more high-fidelity environment that will provide a more realistic delay and noise from the sensors.
To achieve a proper SIL simulation is needed, the firmware of the code where the control algorithm has to be implement, a virtual environment with drone's data and a ground station to plan the mission, send the desired path and log the response of the quadrotor to evaluate the performance of the drone.
Communication between the simulated PX4 autopilot and external systems is primarily facilitated through MAVLink protocol over designated UDP ports. By default, PX4 uses UDP port 14550 for communication with ground control stations (GCS) like QGroundControl, and UDP port 14540 for off board APIs such as MAVSDK or MAVROS. The simulator itself listens on TCP port 4560, with PX4 initiating a TCP connection to this port. This setup ensures seamless data exchange between the simulated autopilot, the simulator, and any connected ground control or off-board systems.
In multi-vehicle simulations, each instance of PX4 is assigned a unique remote UDP port, starting from 14540 and incrementing sequentially (e.g., 14541, 14542, etc.), to ensure proper communication channels for each simulated vehicle.
Additionally, PX4 supports integration with Robot Operating System (ROS) 2, allowing ROS 2 nodes to interface directly with PX4 through uORB topics, facilitating advanced control and data handling capabilities in simulation environments.

7.1 Setup

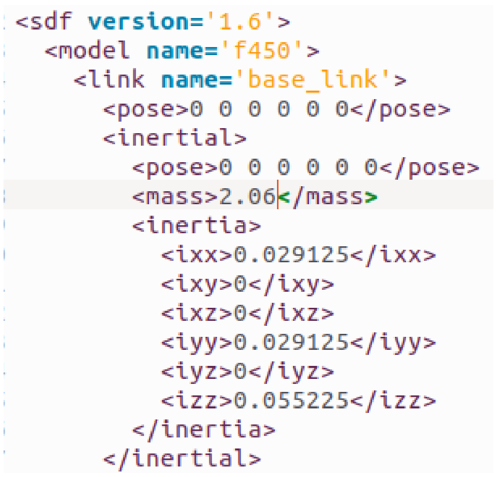
First of all a clone of the PX4 firmware is needed, this can be clone from the following GitHub[10] link through the "git clone" command in the terminal:
The PX4



```
paolo@paolo-pc:~$ git clone https://github.com/PX4/PX4-Autopilot
```

Figure 7.1: Clone command from GitHub.

Gazebo In the PX4 folder are already available different models of drone, aircraft, and rover. In this section, we discuss how the parameters evaluated in the chapter [5] are implemented in the *.sdf* file to customize the default gazebo model. The mass and the inertia as can be seen in the imagine below are implemented in the *.sdf* file in a really simple and direct way:
Meanwhile, for the thrust and torque coefficients are integrated following the following file that describe the mathematical model that gazebo uses to describe the motor[9], the other values will be briefly discussed but are the same of the default $IRIS$ drone. The $jointName$ and $linkName$ are used to describe while rotor is used, the $turningDirection$ is used to describe the cw clockwise or ccw counter-clockwise direction. The $timeConstantUp$ and $timeConstantDown$ are the two time constants of the motor when receiving a command to increase or decrease the speed of the propellers, $maxRotVelocity$ is the maximum velocity



```
<sdf version='1.6'>
  <model name='f450'>
    <link name='base_link'>
      <pose>0 0 0 0 0 0</pose>
      <inertial>
        <pose>0 0 0 0 0 0</pose>
        <mass>2.06</mass>
        <inertia>
          <ixx>0.029125</ixx>
          <ixy>0</ixy>
          <ixz>0</ixz>
          <iyy>0.029125</iyy>
          <iyz>0</iyz>
          <izz>0.055225</izz>
        </inertia>
      </inertial>
```

Figure 7.2: Mass and Inertia values in the .sdf file.

of the motor to ensure the saturation of the command for safety. The thrust coefficient is describes as:

$$T = \omega^2 C_T \tag{7.1}$$

Where ω is in $[rad/s]$ is the same evaluated in the file MATLAB used in $[5.1]$, was already evaluated in rad/s knowing the $.sdf$ file. The torque moment instead is defined as

$$moment\ constant = \frac{Q}{T} \tag{7.2}$$

or

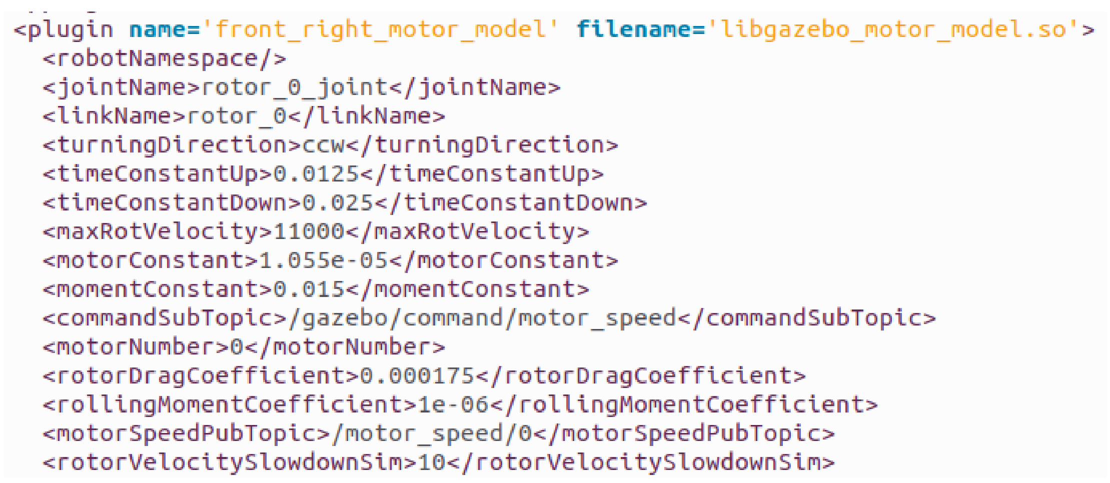
$$moment\ constant = \frac{C_{q0}}{C_{T0}} D \tag{7.3}$$

where D is the diameter in $[m]$ of the propeller.
The first formula is been used to evaluate the coefficient as it provides the more direct way. The value is :

$$moment\ constant = 0.015 \tag{7.4}$$

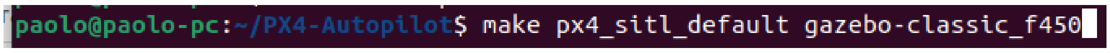
The least to values are $rotorDragCoefficient$ that is the drag itself of the motor and $rollingMomentCoefficient$ that is the the roll or pitch moment that the propellers can induced through non-alignments.

QGroundControl QGroundControl has been chose as a ground station due to the high connection that's have with the PX4's firmware.
QCG can also be used to setup the sensors, RC or test the correct configuration of the motor and their direction, but this will discuss in the further chapter.
At the moment QCG is used as ground control station to upload a mission plan and through the $PID\ tununing$ section evaluated a first guess of the response of the drone



```
<plugin name='front_right_motor_model' filename='libgazebo_motor_model.so'>
  <robotNamespace/>
  <jointName>rotor_0_joint</jointName>
  <linkName>rotor_0</linkName>
  <turningDirection>ccw</turningDirection>
  <timeConstantUp>0.0125</timeConstantUp>
  <timeConstantDown>0.025</timeConstantDown>
  <maxRotVelocity>11000</maxRotVelocity>
  <motorConstant>1.055e-05</motorConstant>
  <momentConstant>0.015</momentConstant>
  <commandSubTopic>/gazebo/command/motor_speed</commandSubTopic>
  <motorNumber>0</motorNumber>
  <rotorDragCoefficient>0.000175</rotorDragCoefficient>
  <rollingMomentCoefficient>1e-06</rollingMomentCoefficient>
  <motorSpeedPubTopic>/motor_speed/0</motorSpeedPubTopic>
  <rotorVelocitySlowdownSim>10</rotorVelocitySlowdownSim>
```

Figure 7.3: Motor model values in the .sdf file.



```
paolo@paolo-pc:~/PX4-Autopilot$ make px4_sitl_default gazebo-classic_f450
```

Figure 7.4: Motor model values in the .sdf file.

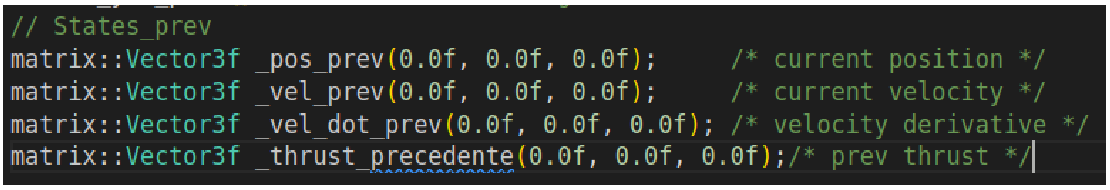
SITL with default controller At this point a SIL can be launch in the terminal with the following command: The command has been customize with the $f450$ ending to launch a simulation with the customize drone with values describe above. Ones all the file will be built an empty Gazebo environmental will be open and the SIL simulation can start using the commander in the terminal or opening QCG it will auto-connect to the simulation and a mission plan can be upload and launched.

7.2 INDI algorithms implementation in C++ code

In this chapter would be explained how the INDI has been implemented from the default P-PID controller in position and rate both. The first loop to be modified has been the position loop to ensure to not destabilize the inner dynamic and be sure that if a problem would be arise the problem would be in the outer loop. One the controller is tuned for the position loop also in the rate loop the INDI is implemented.

position control As explained in chapter [3.3.1] the default controller in the PX4's firmware is a P-PID cascade controller. The implementation lays on three main part: adding the variable for the previous thrust, adding the previous value of the position and finally filtering the output of the control law to synchronize the delays due to the EKF(Extended Kalman Filter). A briefly introduction has to be done since the feedback in the MIL and in the SIL simulations are widely different, as see in the MIL the $previous\ thrust$ derives from the RPM times the G^{-1} motor mixer matrix, in the SIL this was not possible since in the drone there are no RPM sensors at the moment. The value of the previous simple time has been saved through the cycle directly from the previous evalutation of the thrust. In these scheme the ESC, motors and propellers are out of the feedback loop so this architecture is less robust against changes in the ESC, motor or propellers dynamic.
The $previousthrust$ it has been to initialize as a global variable, this is not the best option using C++ as coding program but this enable to, in further research, to evaluate the $previous thrust$ from the RPM value as in the MIL. Also for the position and velocity previous state the same

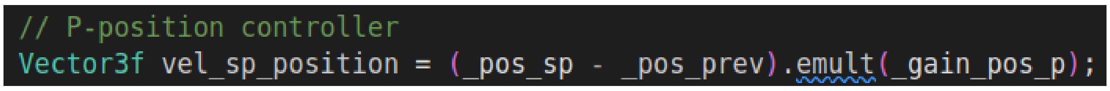
methodology has been used.
Then the PD linear controller has been introduced using the P-PID cascade controller, deleting



```
// States_prev
matrix::Vector3f _pos_prev(0.0f, 0.0f, 0.0f);     /* current position */
matrix::Vector3f _vel_prev(0.0f, 0.0f, 0.0f);     /* current velocity */
matrix::Vector3f _vel_dot_prev(0.0f, 0.0f, 0.0f); /* velocity derivative */
matrix::Vector3f _thrust_precedente(0.0f, 0.0f, 0.0f);/* prev thrust */
```

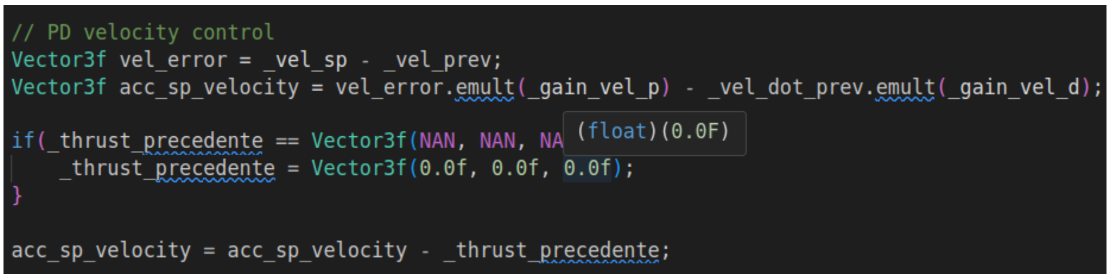
Figure 7.5: Previous state initialization.

the integrator term. In the imagine below the desired values evaluated by the control law are called *setpoint* and usually end with sp, as in the default PX4 firmware.
As describe before after the control law a filter action is needed to synchronize the delays,



```
// P-position controller
Vector3f vel_sp_position = (_pos_sp - _pos_prev).emult(_gain_pos_p);
```

Figure 7.6: Position P-controller in PX4's code.

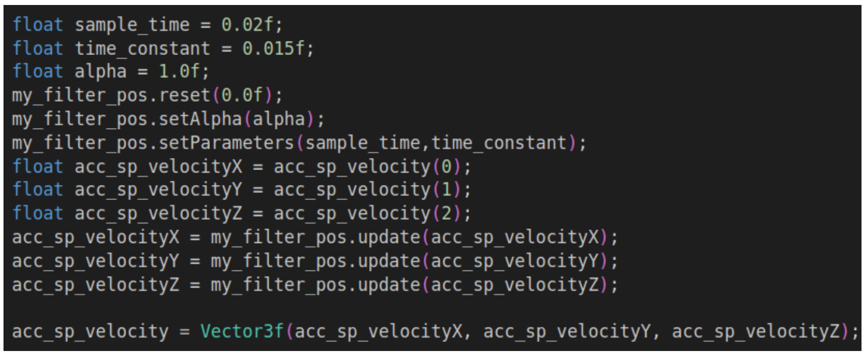


```
// PD velocity control
Vector3f vel_error = _vel_sp - _vel_prev;
Vector3f acc_sp_velocity = vel_error.emult(_gain_vel_p) - _vel_dot_prev.emult(_gain_vel_d);

if(_thrust_precedente == Vector3f(NAN, NAN, NA (float)(0.0F)
    _thrust_precedente = Vector3f(0.0f, 0.0f, 0.0f);
}

acc_sp_velocity = acc_sp_velocity - _thrust_precedente;
```

Figure 7.7: Velocity PD-controller in PX4's code.

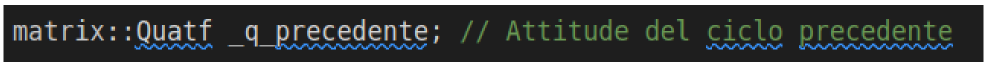
without this action a strong chattering response would be seen in the attitude desired for ϕ and θ and in the Z position.
A first order filter function it was already ready in the PX4's folder called $Alpha\ filter$ a time constant has been defined for the filter action and it is defined as the equal at the delays from the sensor block, a delay of $\simeq 8\,ms$ from the fusion data is shown in the parameter section of the PX4 main guide[6], other delays from EKF are taken in account but are not well defined in the guide so the time constant T_f has been defined as $= 0.015$.

Attitude control Ones the position loop was properly tuned also the rate loop has been changed from P-PID controller to PD-INDI controller. The same scheme has been used to implement the controller, using global variable to save values through the cycle and a filter to synchronize the delays.
As explained in chapter[3.3.1] the attitude and the rate controller are in different modules, so the output of the attitude module would be $rate\ setpoint$.
As for the position the simple proportional P gain is used in the quaternion's control law. The quaternion error is already evaluted in the PX4's firmware, called as can be seen in the imagine below eq, is multiply by the proportional gain.



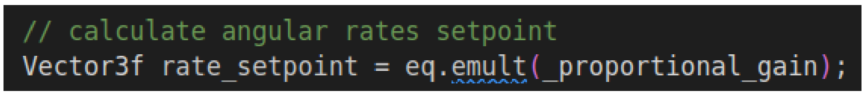
```
float sample_time = 0.02f;
float time_constant = 0.015f;
float alpha = 1.0f;
my_filter_pos.reset(0.0f);
my_filter_pos.setAlpha(alpha);
my_filter_pos.setParameters(sample_time,time_constant);
float acc_sp_velocityX = acc_sp_velocity(0);
float acc_sp_velocityY = acc_sp_velocity(1);
float acc_sp_velocityZ = acc_sp_velocity(2);
acc_sp_velocityX = my_filter_pos.update(acc_sp_velocityX);
acc_sp_velocityY = my_filter_pos.update(acc_sp_velocityY);
acc_sp_velocityZ = my_filter_pos.update(acc_sp_velocityZ);

acc_sp_velocity = Vector3f(acc_sp_velocityX, acc_sp_velocityY, acc_sp_velocityZ);
```

Figure 7.8: Filtering action in the PX4's code.



```
matrix::Quatf _q_precedente; // Attitude del ciclo precedente
```

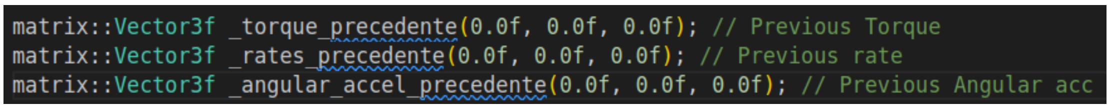
Figure 7.9: Previous quaternion initialization in the PX4's code.



```
// calculate angular rates setpoint
Vector3f rate_setpoint = eq.emult(_proportional_gain);
```

Figure 7.10: Quaternion's control law in the PX4's code.

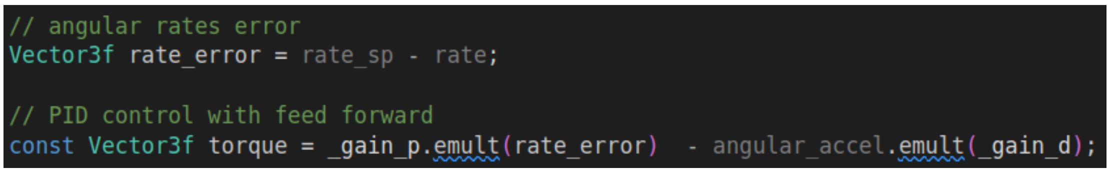
Rate control As in the position control module the values are initialized. In particular in this module to complete the INDI controller law a *previous torque*, *previous velocity* and a *previous angular acceleration* values are needed.
Then the PID control law has reduced to a PD controller:



```
matrix::Vector3f _torque_precedente(0.0f, 0.0f, 0.0f); // Previous Torque
matrix::Vector3f _rates_precedente(0.0f, 0.0f, 0.0f); // Previous rate
matrix::Vector3f _angular_accel_precedente(0.0f, 0.0f, 0.0f); // Previous Angular acc
```

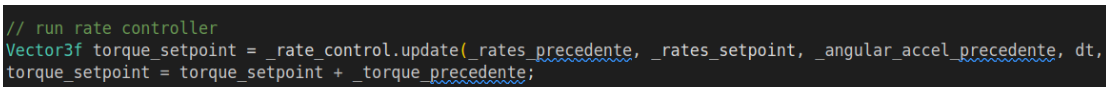
Figure 7.11: Previous state in rate control in the PX4's code.

As for the thrust evaluation also in this loop a filter is needed but since the delays are smaller



```
// angular rates error
Vector3f rate_error = rate_sp - rate;

// PID control with feed forward
const Vector3f torque = _gain_p.emult(rate_error)  - angular_accel.emult(_gain_d);
```

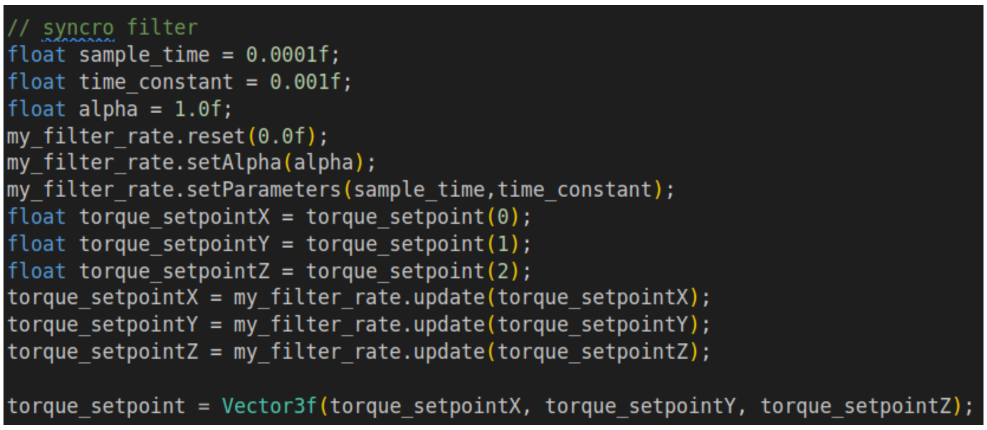
Figure 7.12: Rate's control law in the PX4's code.



```
// run rate controller
Vector3f torque_setpoint = _rate_control.update(_rates_precedente, _rates_setpoint, _angular_accel_precedente, dt,
torque_setpoint = torque_setpoint + _torque_precedente;
```

Figure 7.13: INDI's control law in rate loop in the PX4's code.

also the time constant of the filter could be smaller, in this case $T_f = 0.001\ sec$:

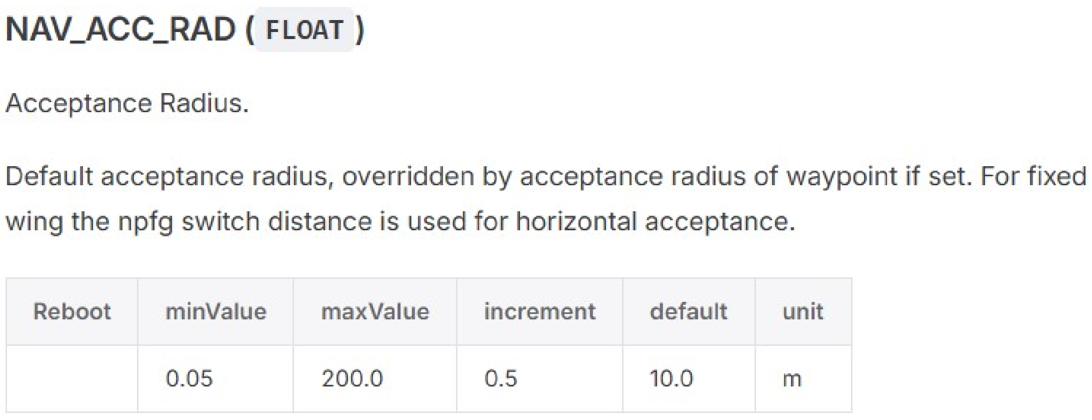


```
// syncro filter
float sample_time = 0.0001f;
float time_constant = 0.001f;
float alpha = 1.0f;
my_filter_rate.reset(0.0f);
my_filter_rate.setAlpha(alpha);
my_filter_rate.setParameters(sample_time,time_constant);
float torque_setpointX = torque_setpoint(0);
float torque_setpointY = torque_setpoint(1);
float torque_setpointZ = torque_setpoint(2);
torque_setpointX = my_filter_rate.update(torque_setpointX);
torque_setpointY = my_filter_rate.update(torque_setpointY);
torque_setpointZ = my_filter_rate.update(torque_setpointZ);

torque_setpoint = Vector3f(torque_setpointX, torque_setpointY, torque_setpointZ);
```

Figure 7.14: Filter in rate loop in the PX4's code.

7.3 Simulation

In this chapter, a waypoint mission has been simulated. The aim of this mission is to validate the generated C++ code and refine the control gain if needed.
The mission scheme consist of a take off to 10 meters, reaching different waypoints, and finally testing the "Return to home" command. This command directs the drone to ascend to 30 meters before returning to the saved home point. This test was chosen because it serves as a safety measure in case of RC signal loss.
In the navigation block, a minimum radius around the waypoint has been set as the accepted trajectory performance criterion. Even with this setting, it can be observed that after every



NAV_ACC_RAD (FLOAT)

Acceptance Radius.

Default acceptance radius, overridden by acceptance radius of waypoint if set. For fixed wing the npfg switch distance is used for horizontal acceptance.

Reboot	minValue	maxValue	increment	default	unit
	0.05	200.0	0.5	10.0	m

Figure 7.15: Acceptance radius parameter on PX4's firmware.

turn the yaw response exhibits an overshoot. This issue will be discussed in more detail in the following section, but it is primarily caused by the navigation block not by an incorrect tuning of the yaw controls gains. This overshoot would not occur if the waypoint were manually inputted one-by-one using the "Go To" mode, rather than relying on the mission mode.

	roll axis	pitch axis	yaw axis
P-angle			
P-rate			
D-rate			

Table 5: Control gains values for attitude control loop.

	X axis	Y axis	Z axis
P-position			
P-velocity			
D-velocity			

Table 6: Control gains values for position control loop.

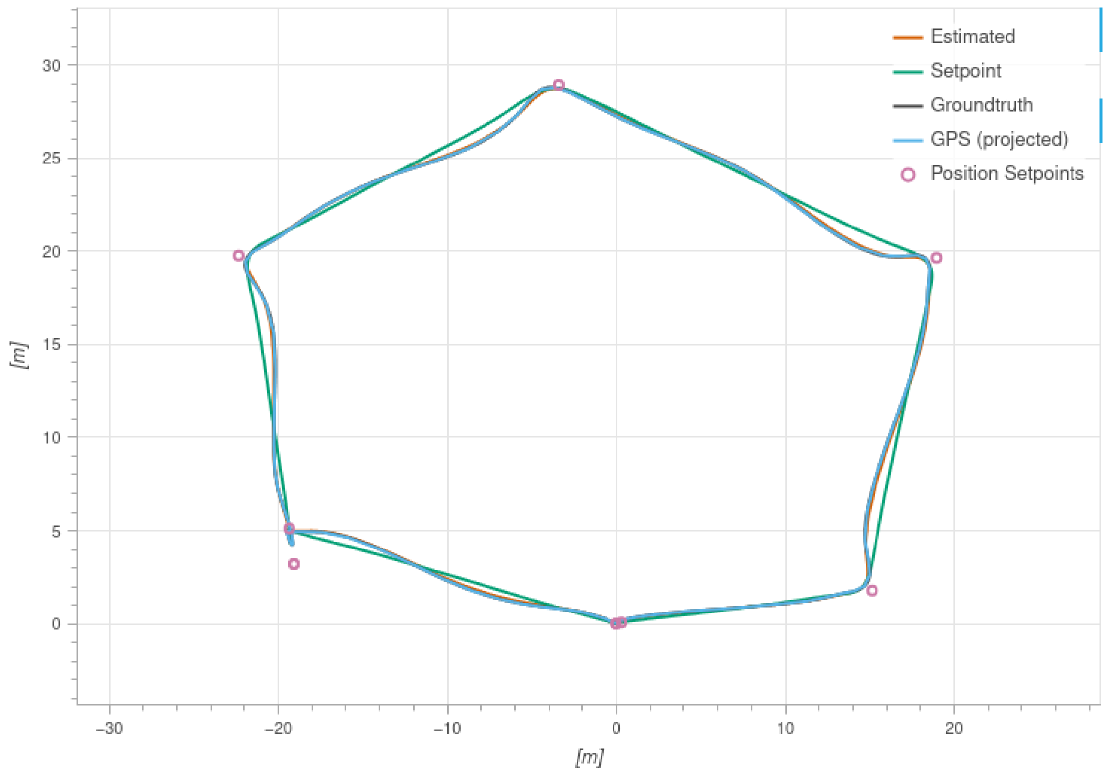


Figure 7.16: SIL trajectory[12].

RMSE position [m]	RMSE angle [deg]
0.29	1.69

Table 7: Comparison RMSE

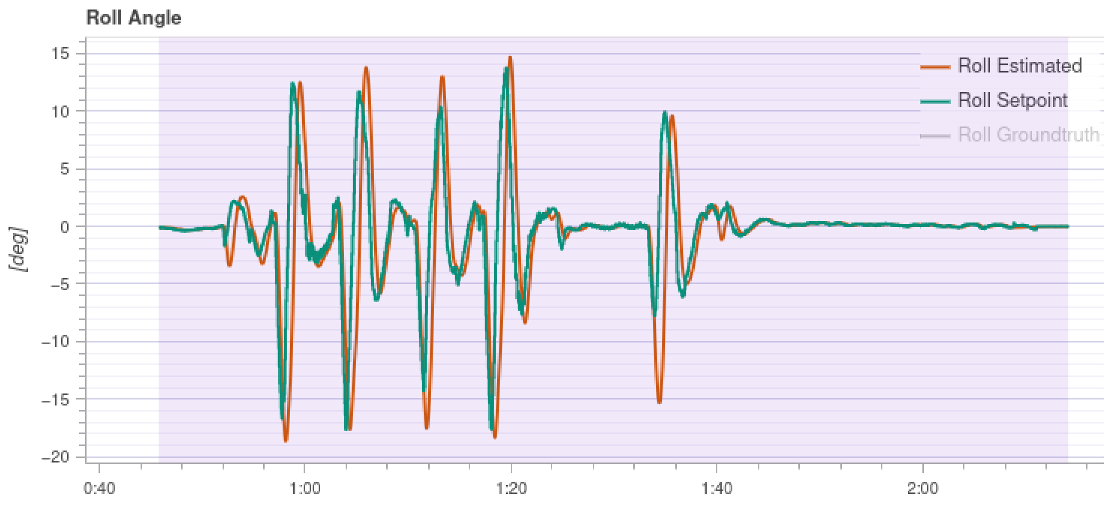


Figure 7.17: SIL roll angle[12].

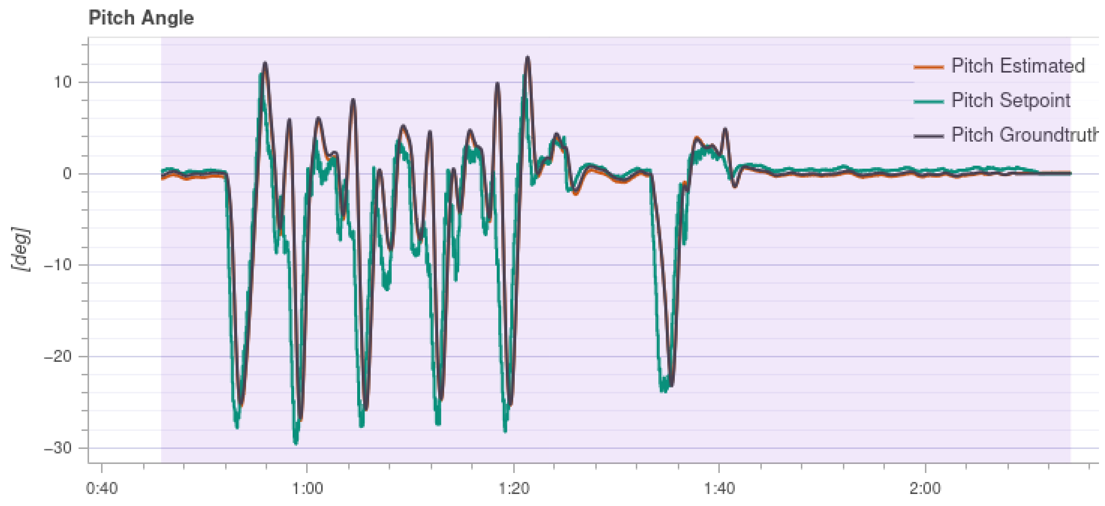


Figure 7.18: SIL pitch angle[12].

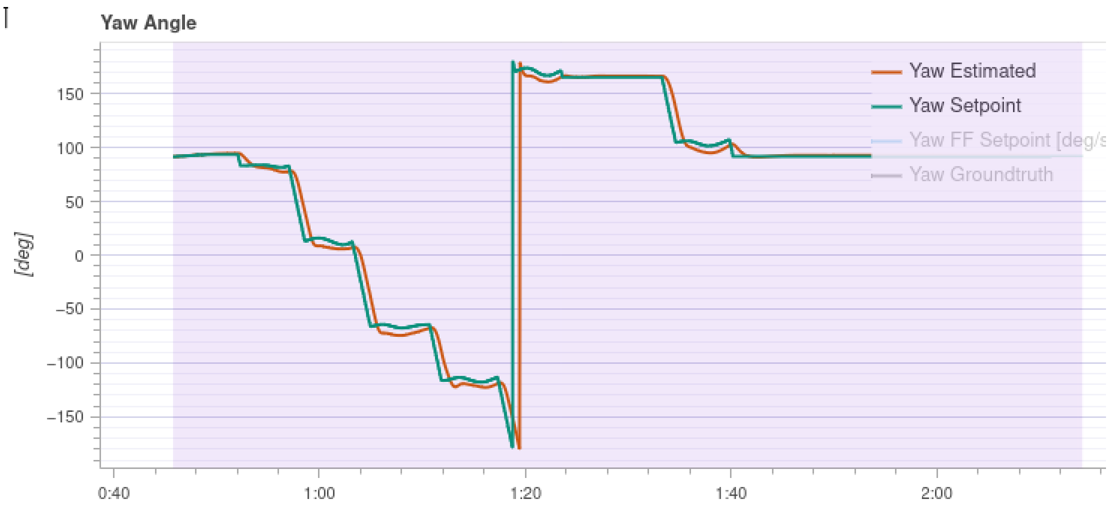


Figure 7.19: SIL yaw angle[12].

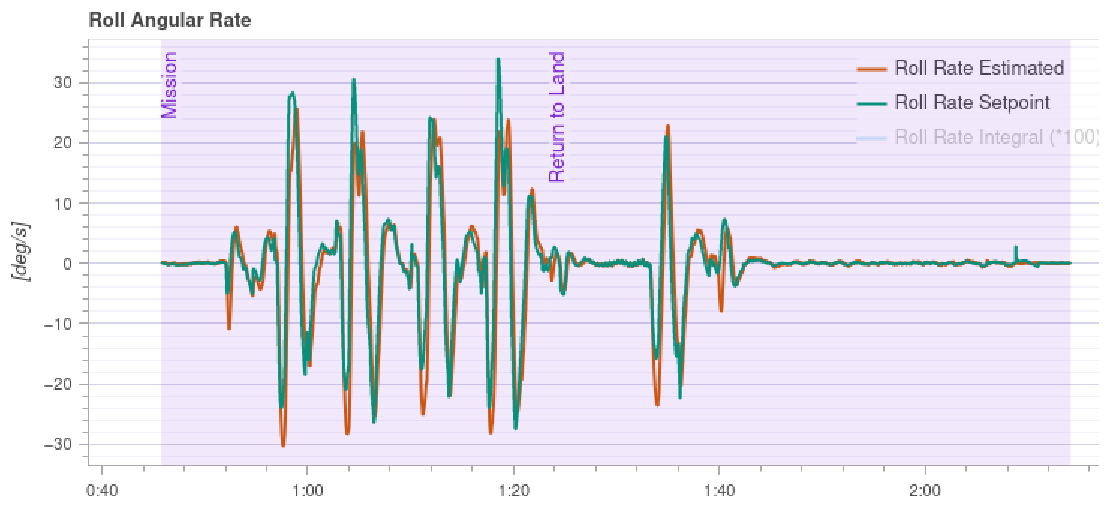


Figure 7.20: SIL roll rate[12].

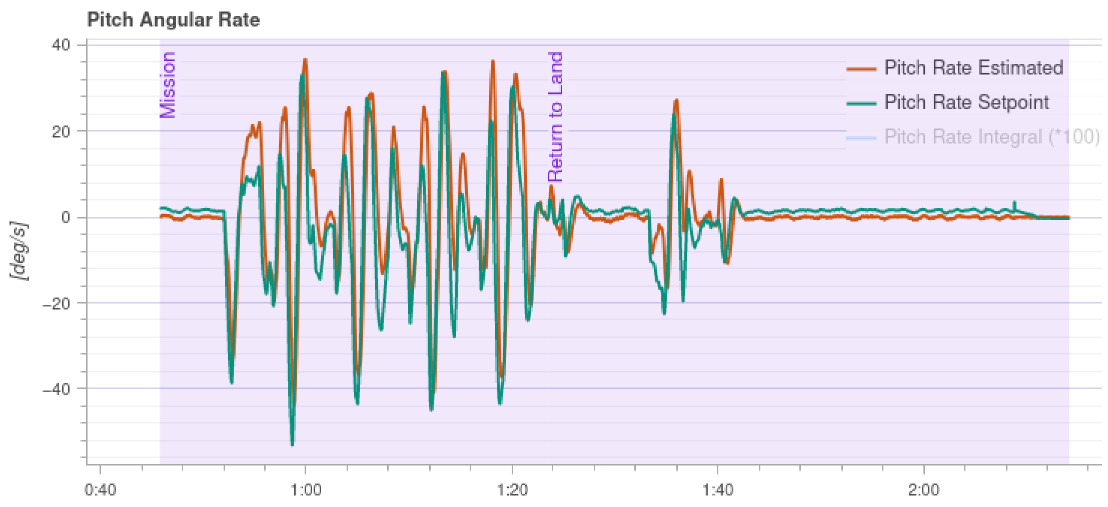


Figure 7.21: SIL pitch rate[12].

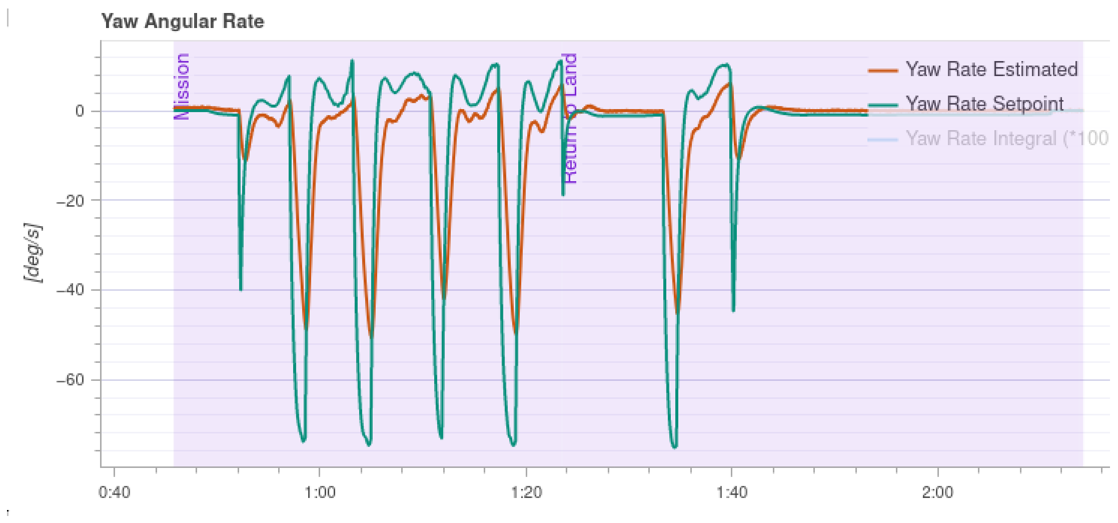


Figure 7.22: SIL yaw rate[12].

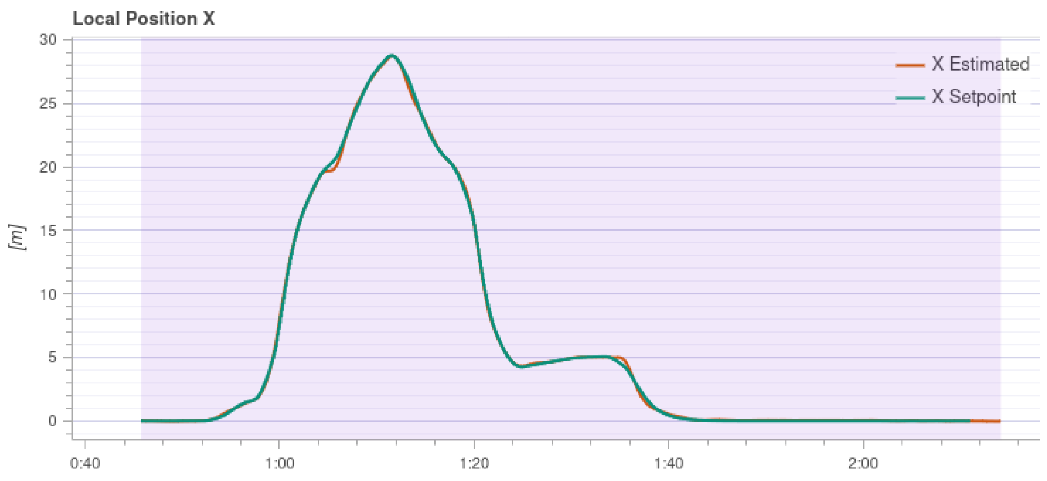


Figure 7.23: SIL X position[12].

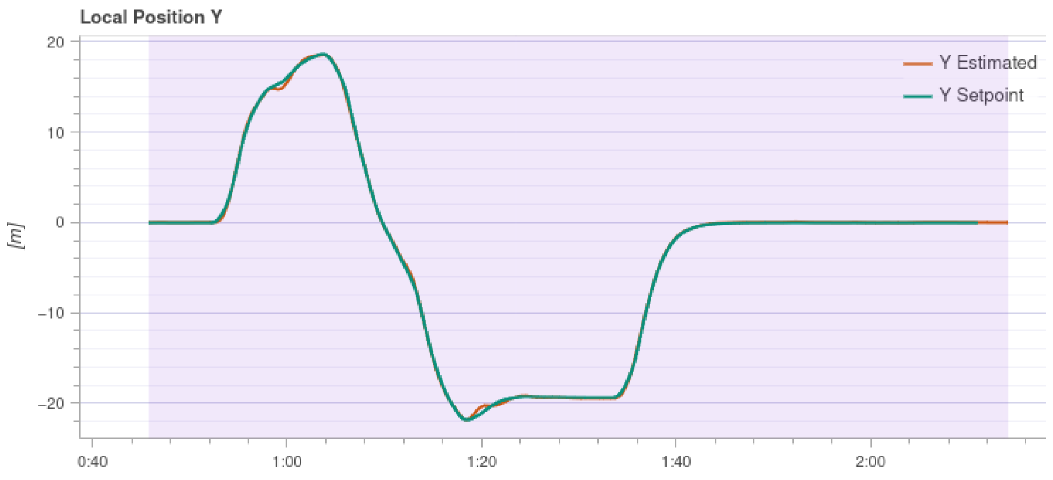


Figure 7.24: SIL Y position[12].

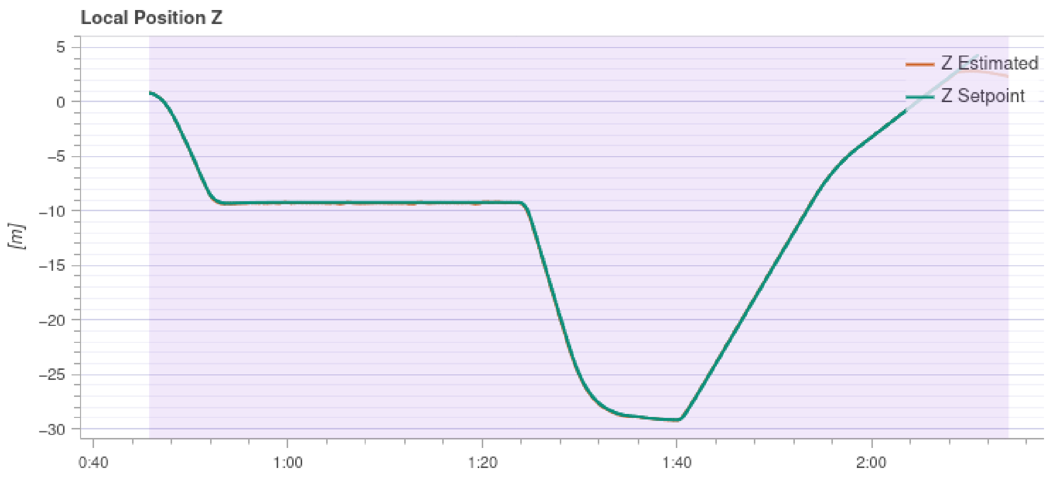


Figure 7.25: SIL Z position[12].

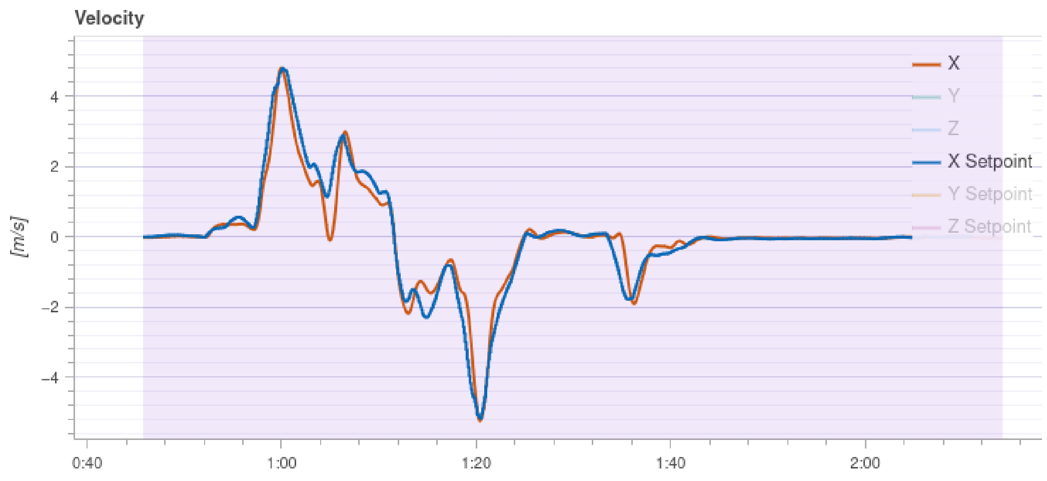


Figure 7.26: SIL X velocity[12].

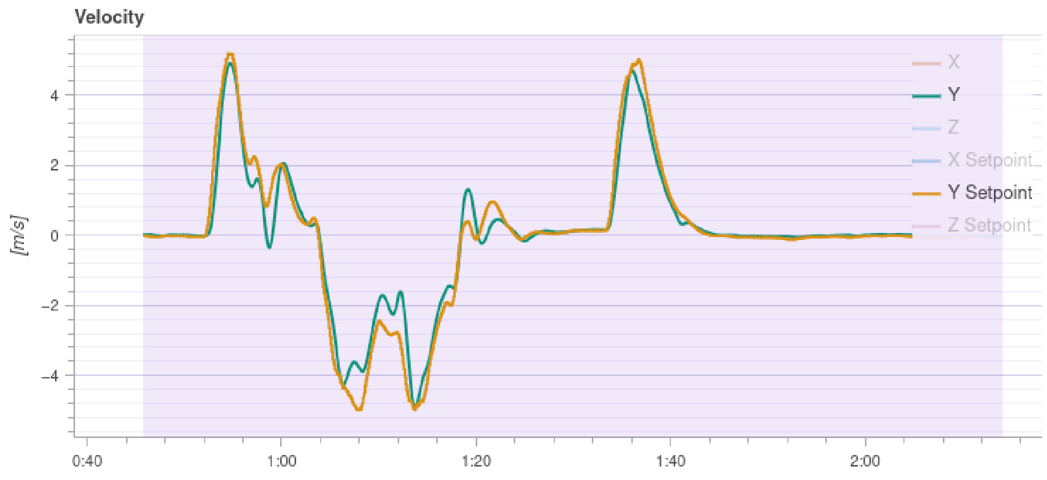


Figure 7.27: SIL Y velocity[12].

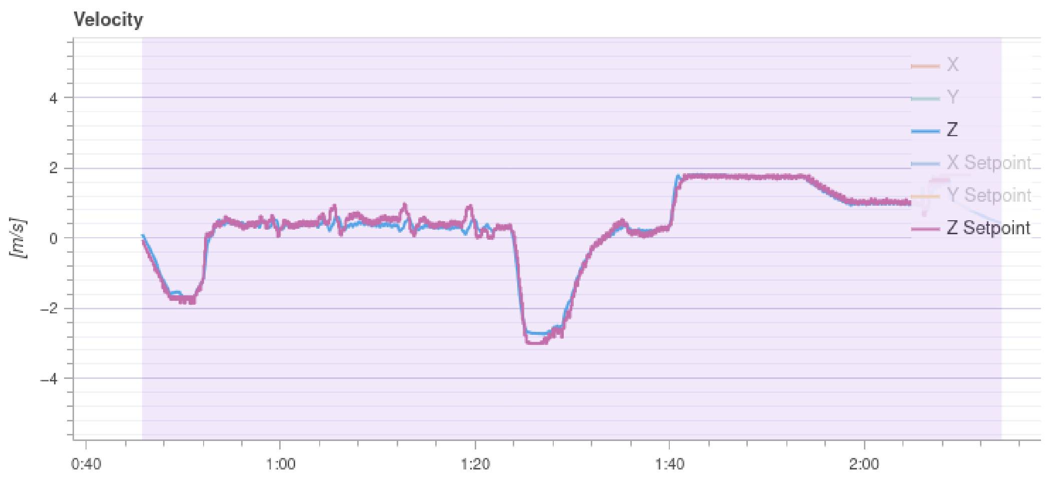


Figure 7.28: SIL Z velocity[12].

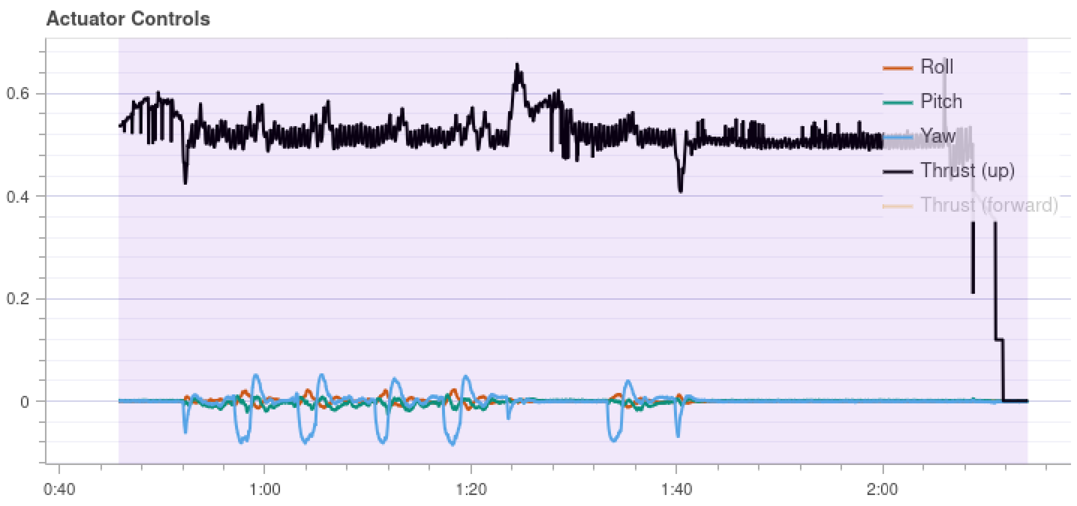


Figure 7.29: SIL normalized control input[12].

7.4 Simulation with uncertainties

As in the previous chapter, several simulations with uncertainties have also been tested in the SIL. In particular, variations in mass, inertia, and motor's time constant have been introduced to evaluate the robustness of the controller.
At the end of the section has been added a table comparing the RMSE of the different simulations.

Mass and inertia changes The evaluated change in mass is the only case where the mass is more, case of a drone with a complain computer, the change is $+25\%$ in mass. The inertia has been evaluated in a range between $[-25\%, +25\%]$.

	RMSE position [m]	RMSE angle [deg]
nominal	0.29	1.69
+25% mass	0.09	2.218
+25% inertia	0.35	1.40
-25% inertia	0.19	1.49
+25% time constant	0.37	2.29
-25% time constant	0.12	1.418

Table 8: Comparison RMSE with uncertainties

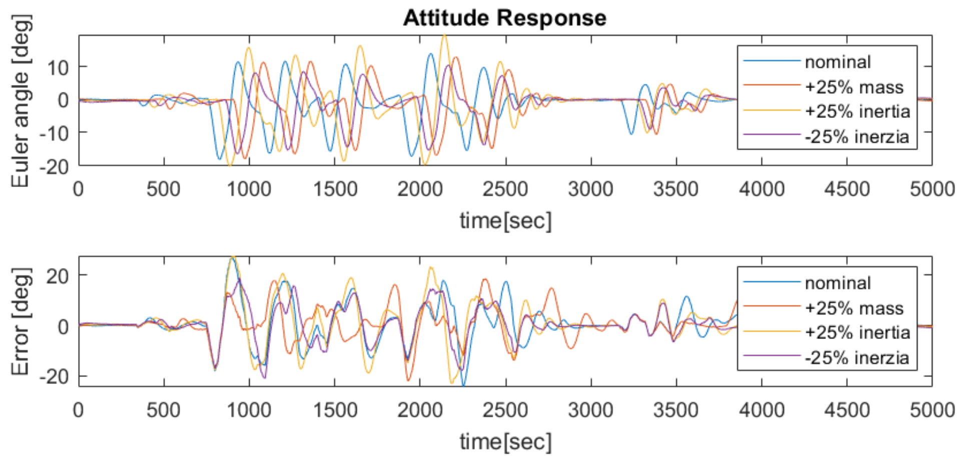


Figure 7.30: SIL roll angle response with inertia uncertainties.

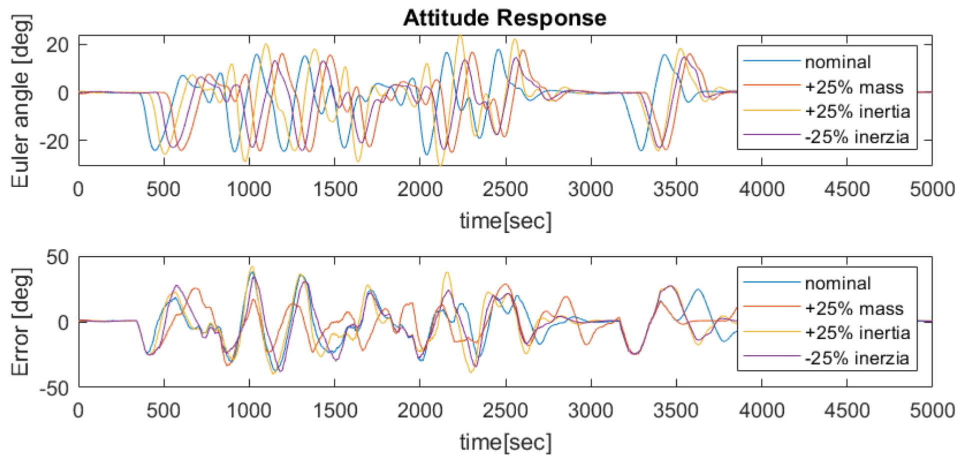


Figure 7.31: SIL pitch angle response with inertia uncertainties.

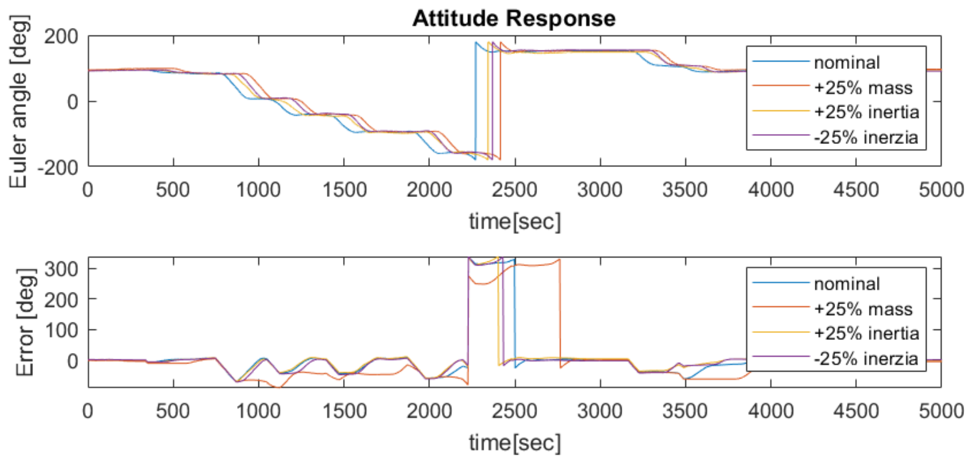


Figure 7.32: SIL yaw angle response with inertia uncertainties.

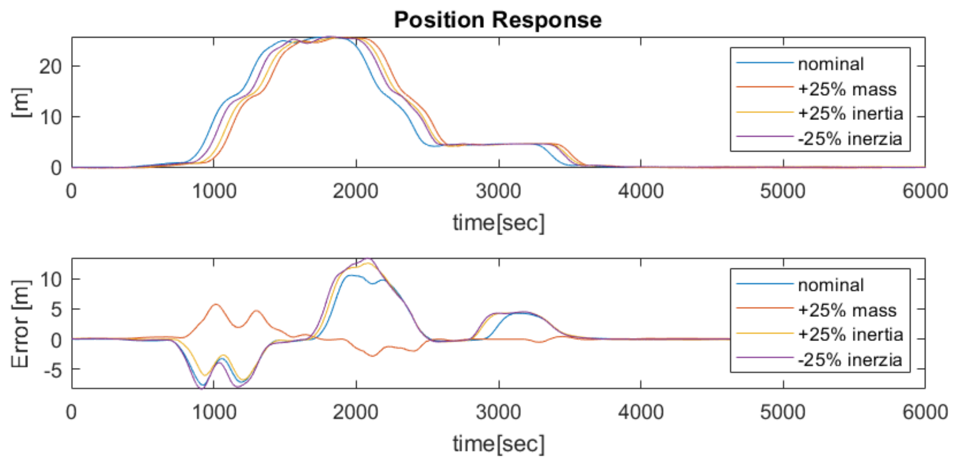


Figure 7.33: SIL x position response with inertia uncertainties.

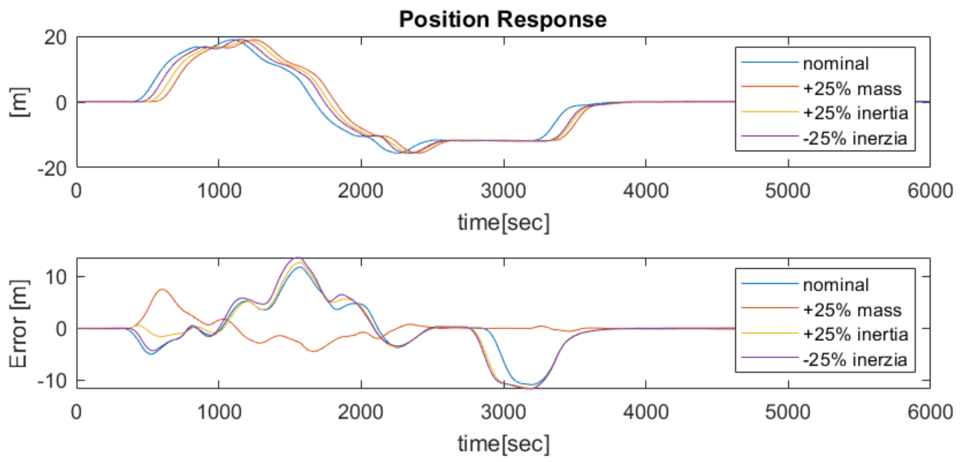


Figure 7.34: SIL y position response with inertia uncertainties.

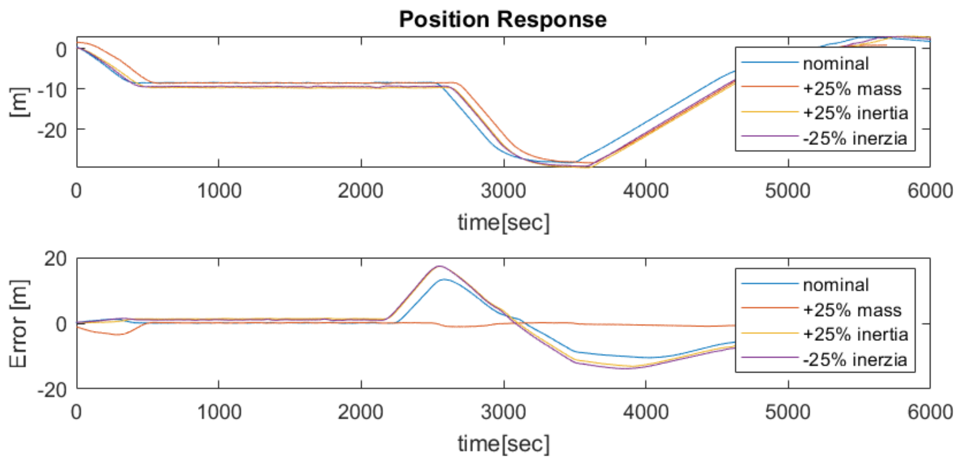


Figure 7.35: SIL z position response with inertia uncertainties.

Time constant For the time constant a maximum range of $[-50\%, +50\%]$ has been evaluated.

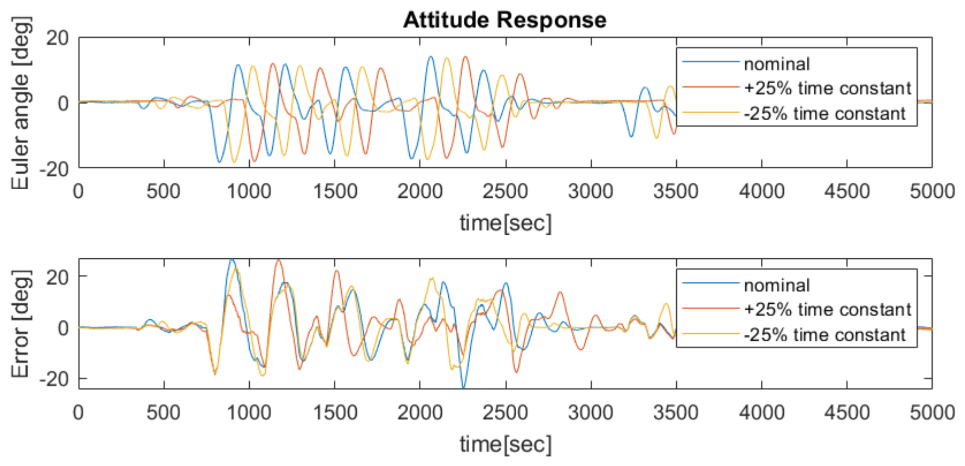


Figure 7.36: SIL roll angle response with time constant uncertainties.

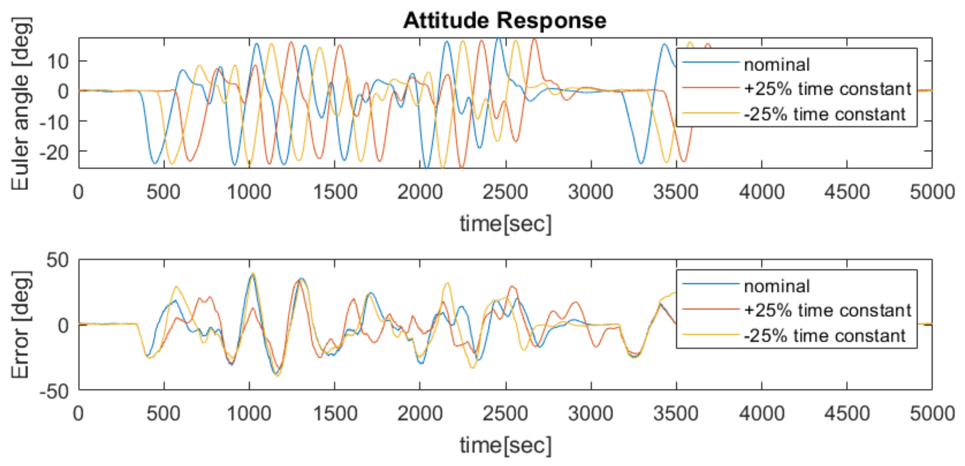


Figure 7.37: SIL pitch angle response with time constant uncertainties.

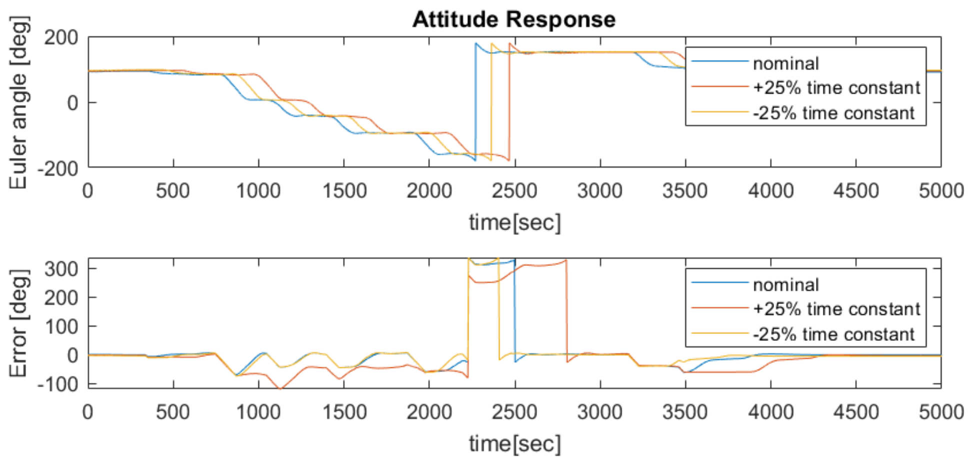


Figure 7.38: SIL yaw angle response with time constant uncertainties.

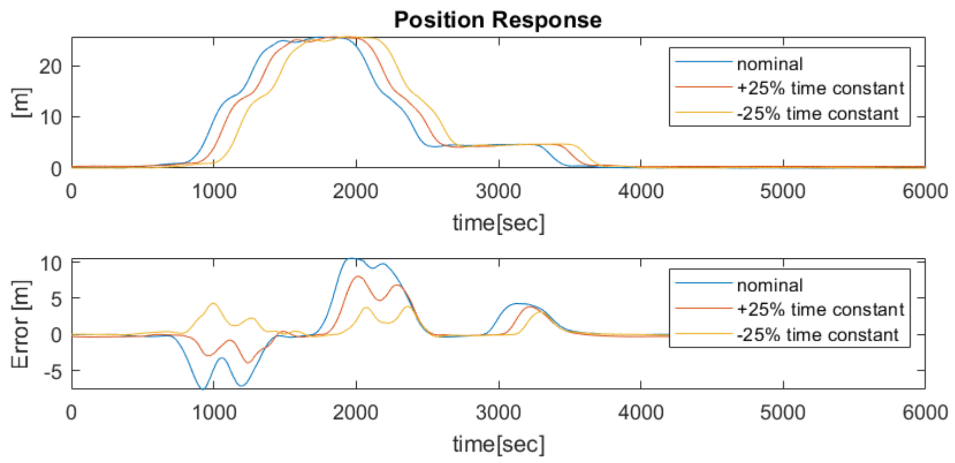


Figure 7.39: SIL x position response with time constant uncertainties.

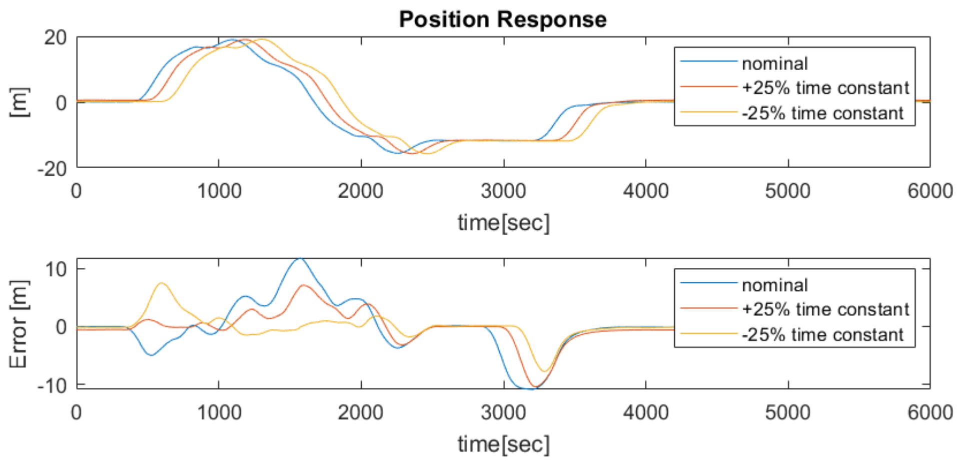


Figure 7.40: SIL y position response with time constant uncertainties.

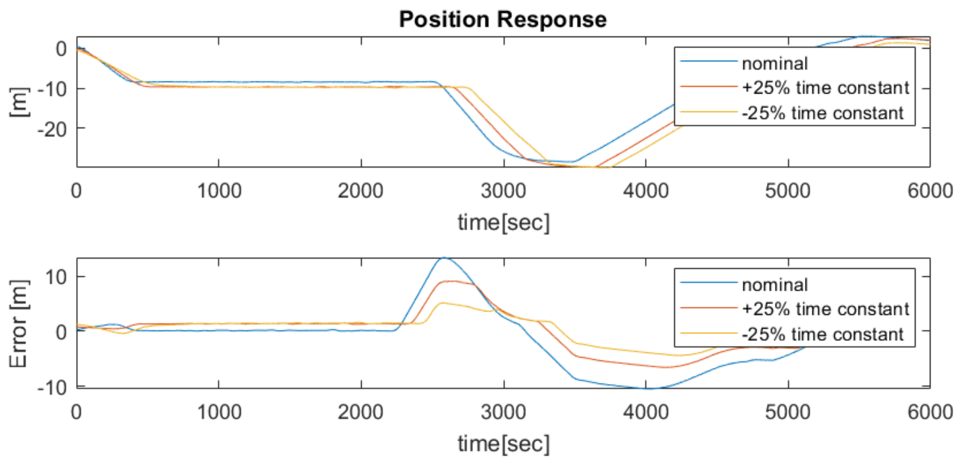


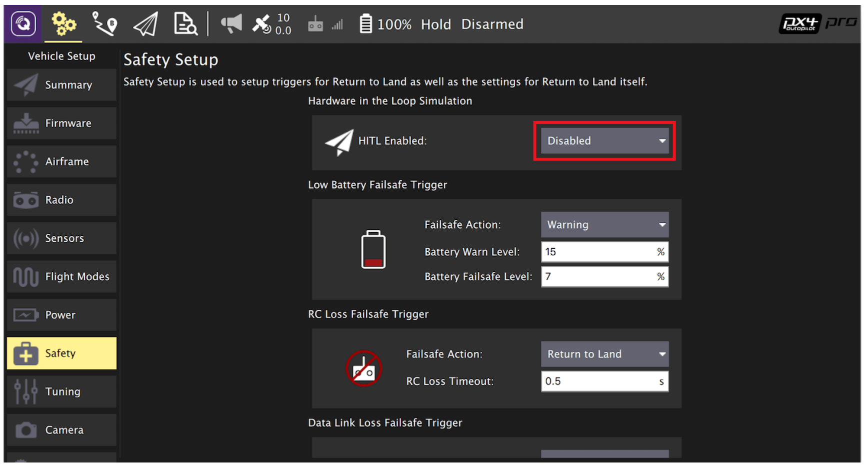
Figure 7.41: SIL z position response with time constant uncertainties.

8 Processor in the loop

In a Processor-In-The-Loop simulation (PIL), the PX4 firmware runs on real hardware. Gazebo-Classic is connected to the flight controller hardware via USB. The simulator acts as a gateway to share MAVLink data between PX4 and QGroundControl[6].
In a PIL simulation no real sensors will start, they still simulate as in a SIL, this is also because the definition of HIL in PX4'guide is a bit loose, in a more technical definition this is a PIL, that could be confusing since every command from PX4 either QGC would refer as Hardware in the loop(HIL).

8.1 Setup

To launch a PIL simulation, a different setup has to be used, in this section the one used following the PX4's guide[6]. In QGC the HIL option must be set; this allows us to use the processor of the board itself and, using the standard firmware, the SIL executes more modules of the system code. Uncheck all *Autoconnect* boxes except for UDP to leave the port free to



Safety Setup

Safety Setup is used to setup triggers for Return to Land as well as the settings for Return to Land itself.

Hardware in the Loop Simulation

HITL Enabled: Disabled

Low Battery Failsafe Trigger

Failsafe Action: Warning
Battery Warn Level: 15 %
Battery Failsafe Level: 7 %

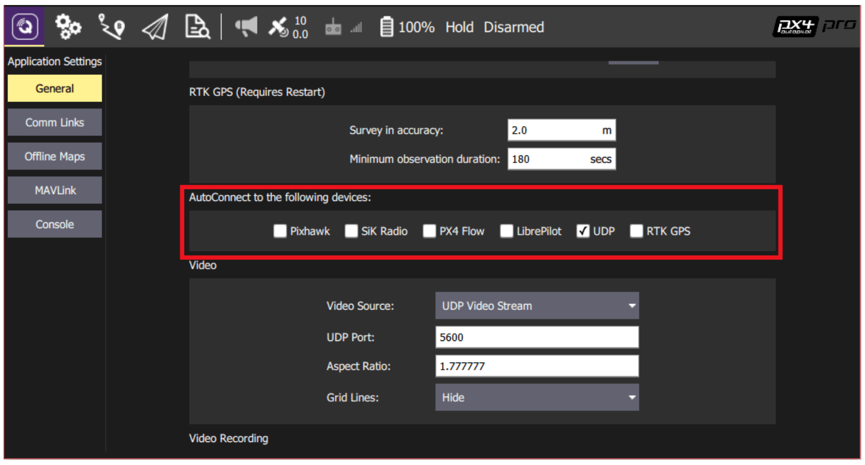
RC Loss Failsafe Trigger

Failsafe Action: Return to Land
RC Loss Timeout: 0.5 s

Data Link Loss Failsafe Trigger

Figure 8.1: QCG HIL setup[6].

connect only when the HIL simulation command is launched.



RTK GPS (Requires Restart)

Survey in accuracy: 2.0 m
Minimum observation duration: 180 secs

AutoConnect to the following devices:

☐ Pixhawk ☐ SiK Radio ☐ PX4 Flow ☐ LibrePilot ☑ UDP ☐ RTK GPS

Video

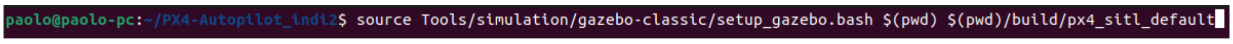
Video Source: UDP Video Stream
UDP Port: 5600
Aspect Ratio: 1.777777
Grid Lines: Hide

Video Recording

Figure 8.2: QCG Autoconnect setup[6].

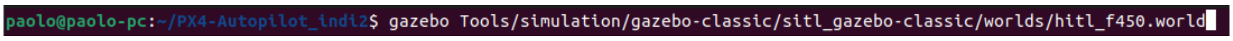
HIL compatible airframe To properly launch the PIL simulation, a different airframe must be used, in particular the $HIL\,Generic\,X$ with the $1001\,sys\,autostart$ that is compatible with the HIL word that is launched in the simulation.

Command to launch a HIL simulation Before launch the command, an important step must be checked, the board must be connected to the $/dev/ttyACM0$ serial port; This setting can be changed in the $HIL\,word$ file in the world folder. The commands that will follow were modified to properly launch a simulation with the characteristic of the $F450$ drone.
The first command is used to configure the environment variables:
The second command launches the HIL simulation:



```
paolo@paolo-pc:~/PX4-Autopilot_indi2$ source Tools/simulation/gazebo-classic/setup_gazebo.bash $(pwd) $(pwd)/build/px4_sitl_default
```

Figure 8.3: Variables setup for HIL simulation[6].



```
paolo@paolo-pc:~/PX4-Autopilot_indi2$ gazebo Tools/simulation/gazebo-classic/sitl_gazebo-classic/worlds/hitl_f450.world
```

Figure 8.4: HIL simulation command[6].

8.2 Simulation

In this section, a waypoint mission has been simulated. The aim of this mission is to test and validate that the processor used can handle the computational requirement of the controller and to increase the fidelity of the delay due to the MAVLink protocol messages. The mission scheme consists of taking off to 10 meters, reaching different waypoints that are manually input one by one, this methodology has been used to test also if the yaw angle overshoot are due to the navigator or the controller. As can be seen the issue relay on the navigator side, that part has not been studied in this thesis.

	RMSE position [m]	RMSE angle [deg]
nominal	0.0046	0.4361

Table 9: Comparison RMSE with uncertainties

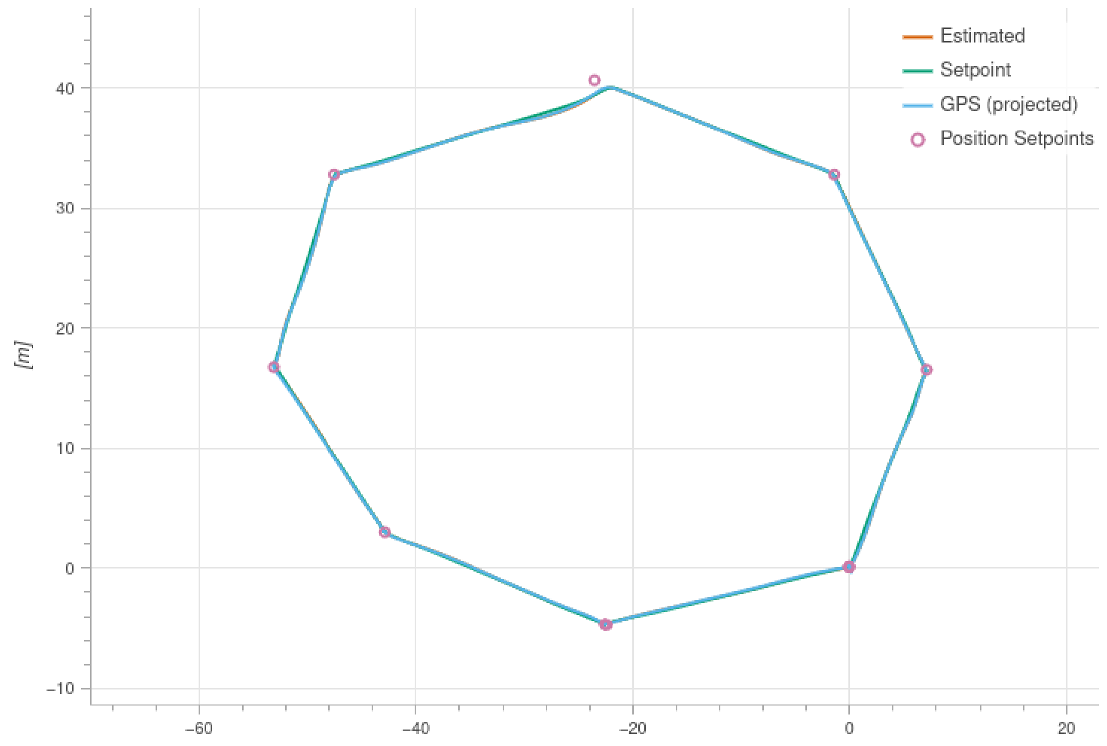


Figure 8.5: HIL trajectory[12].

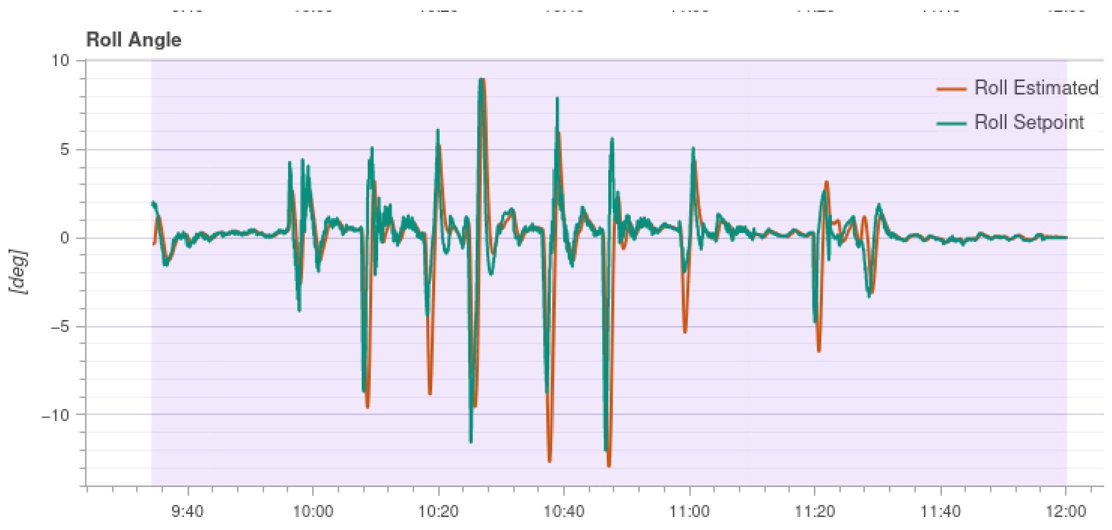


Figure 8.6: HIL roll angle[12].

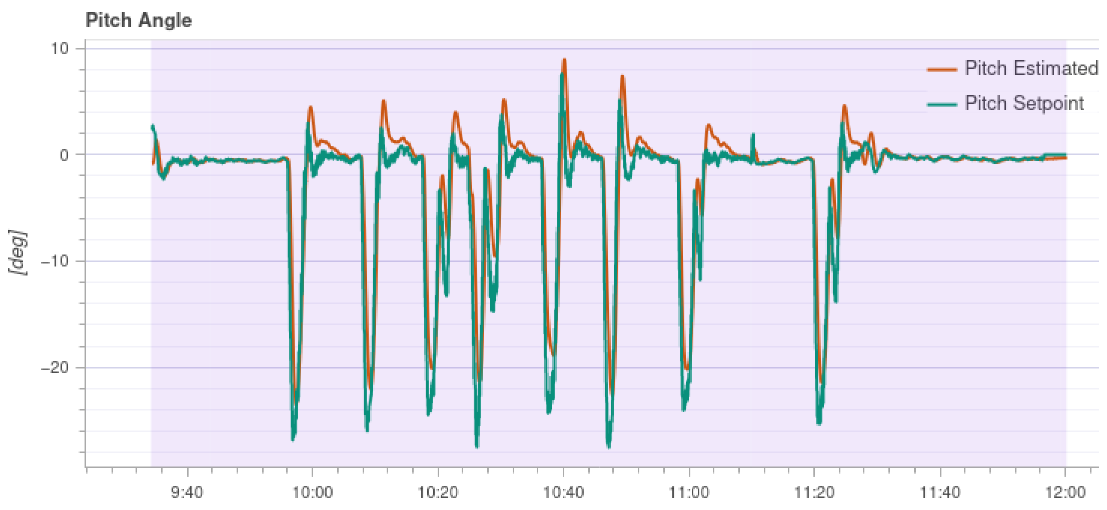


Figure 8.7: HIL pitch angle[12].

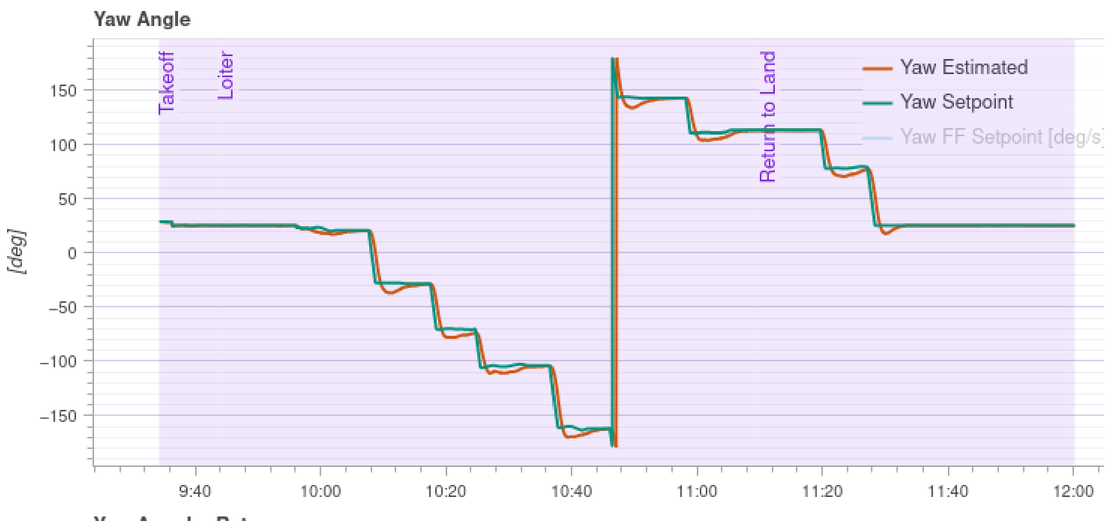


Figure 8.8: HIL yaw angle[12].

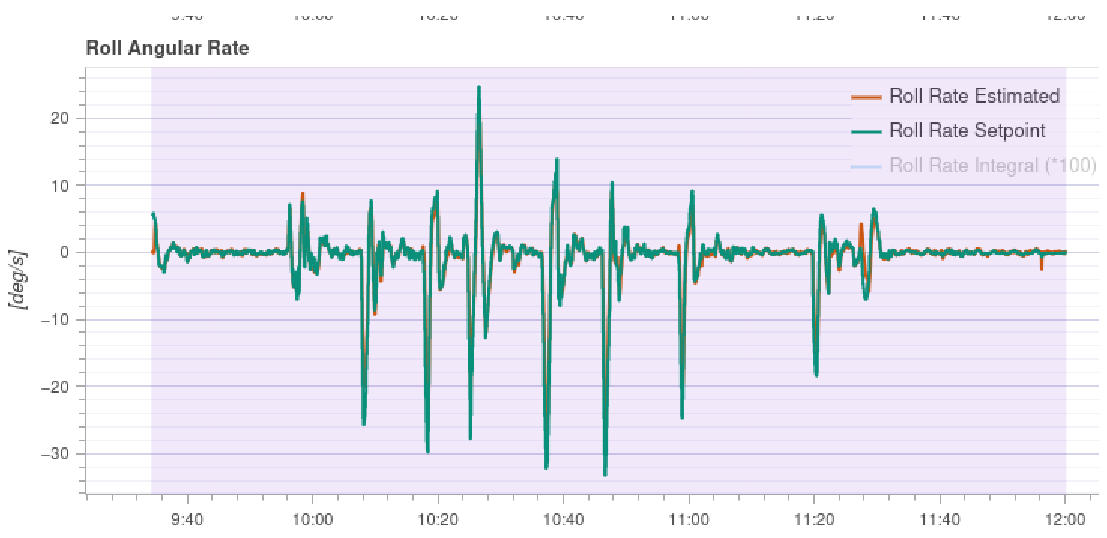


Figure 8.9: HIL roll rate[12].

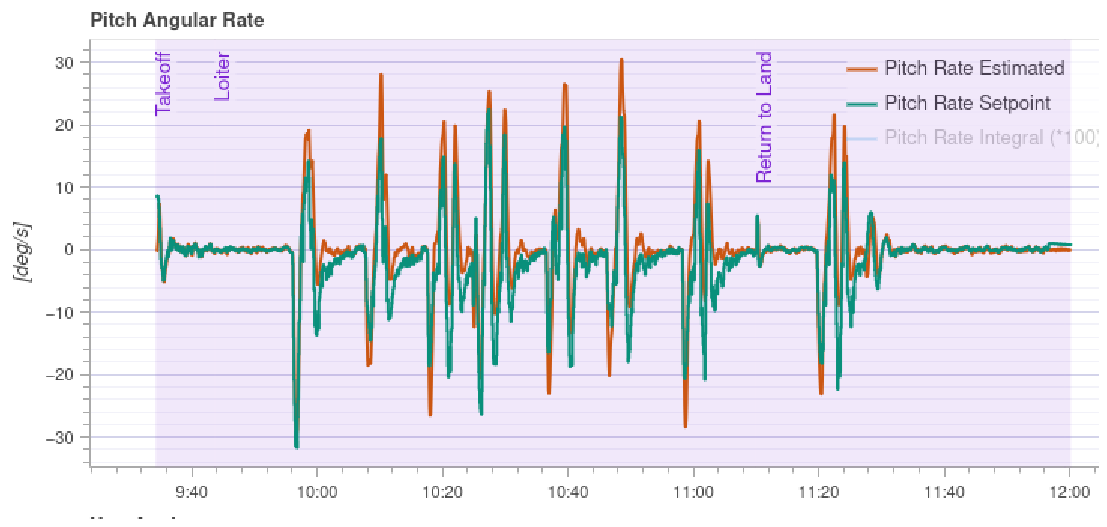


Figure 8.10: HIL pitch rate[12].

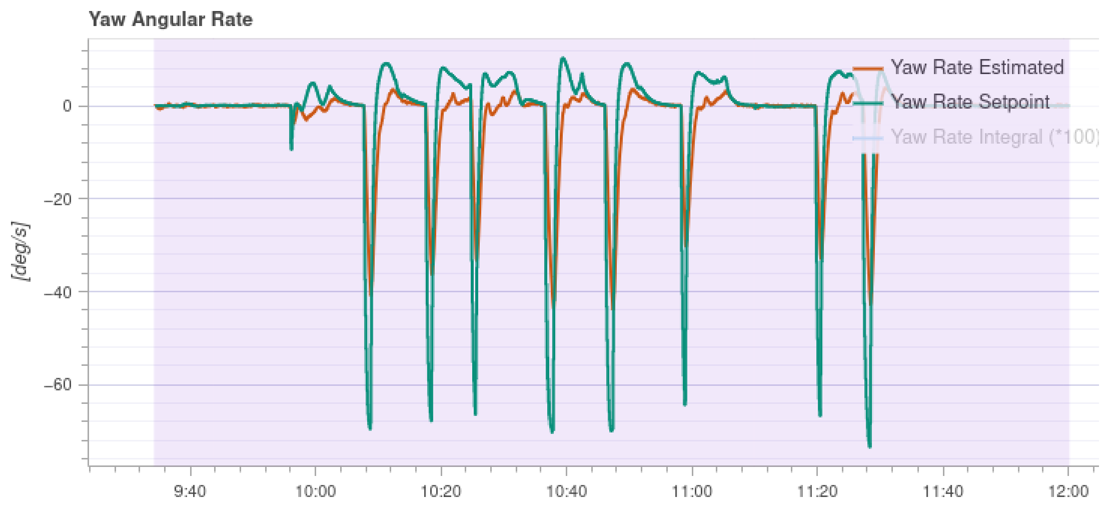


Figure 8.11: HIL yaw rate[12].

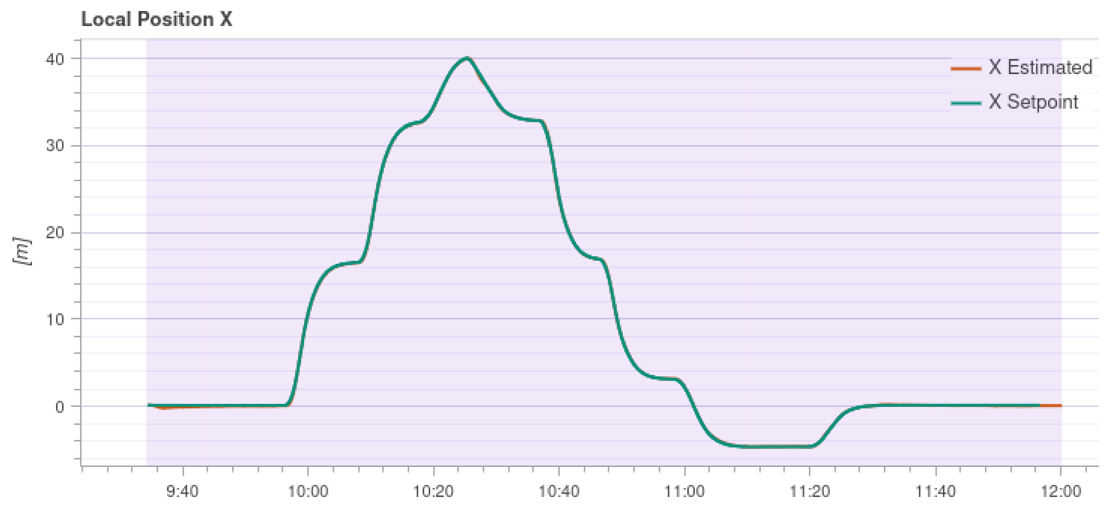


Figure 8.12: HIL X position[12].

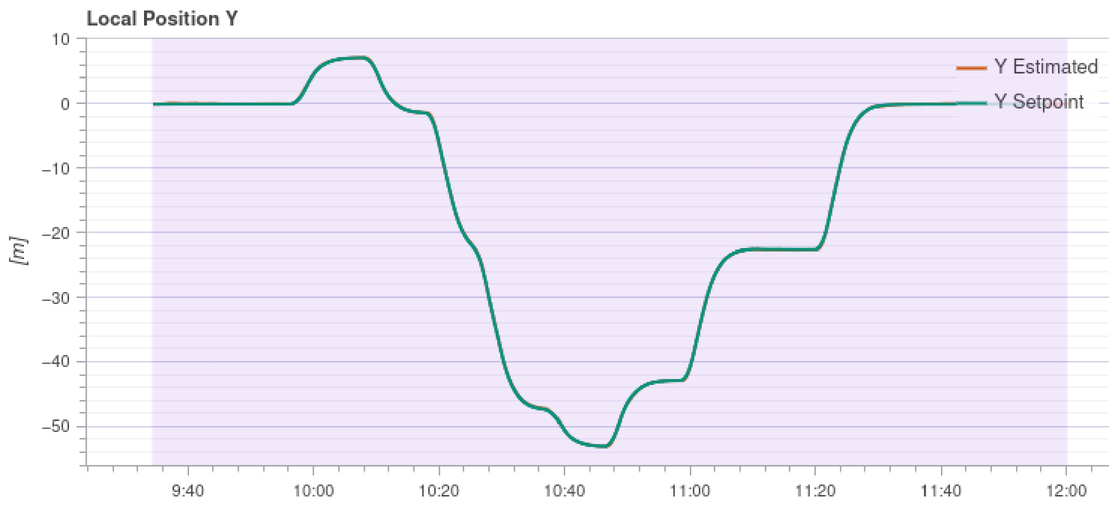


Figure 8.13: HIL Y position[12].

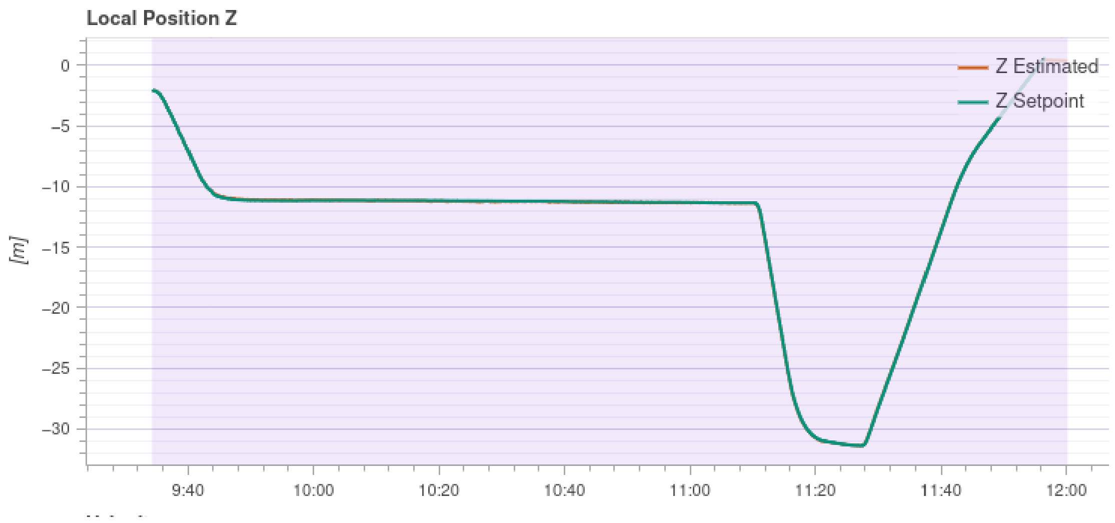


Figure 8.14: HIL Z position[12].

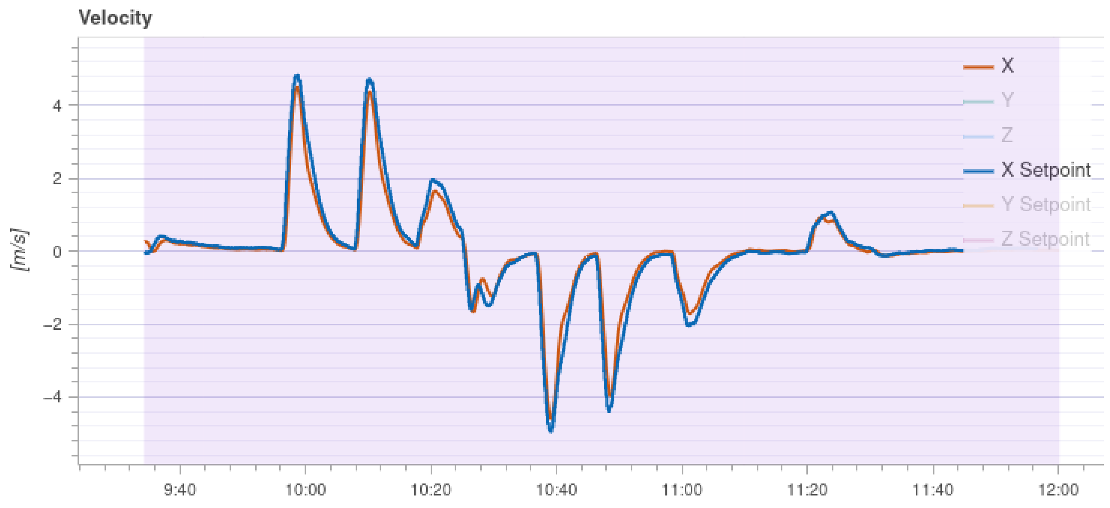


Figure 8.15: HIL X velocity[12].

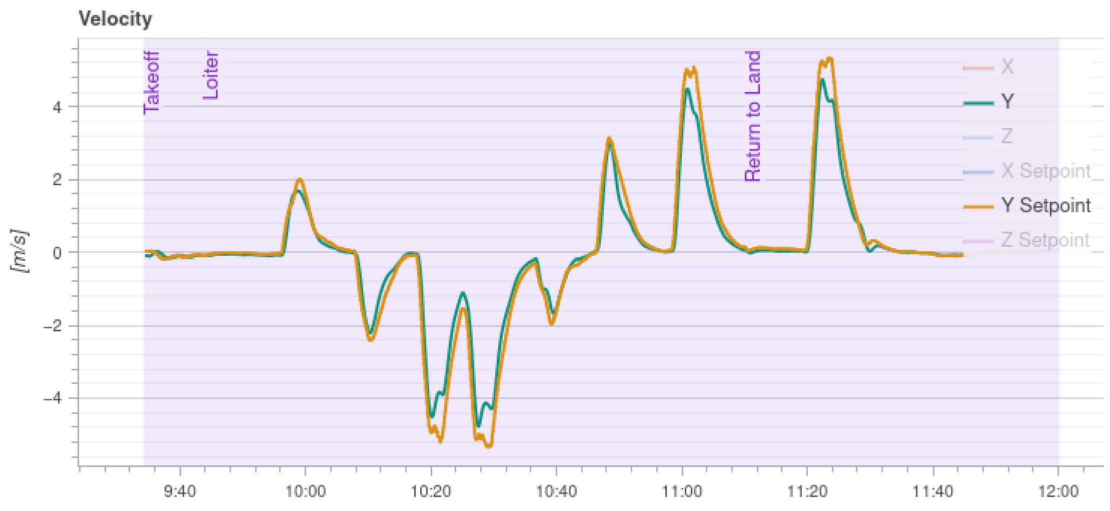


Figure 8.16: HIL Y velocity[12].

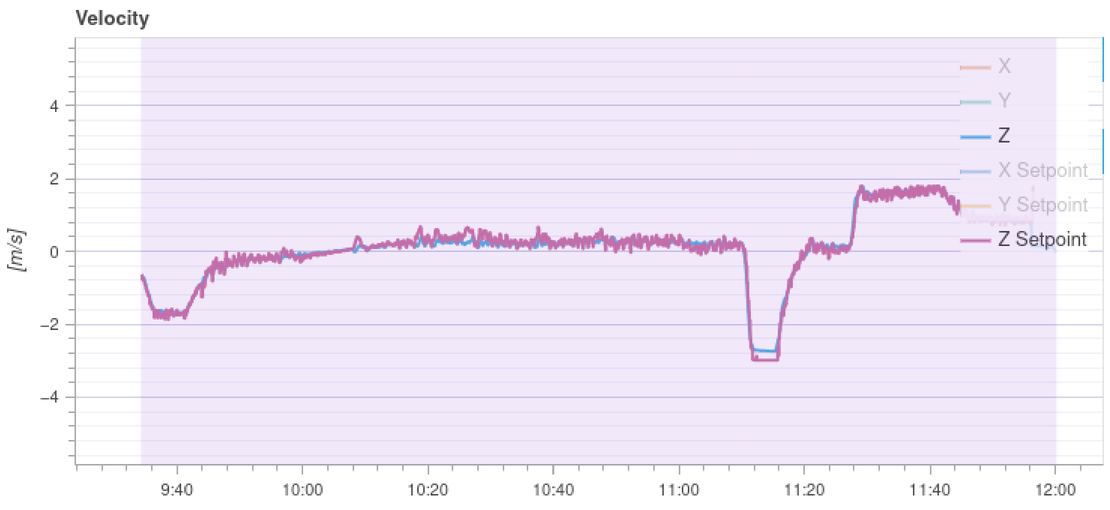


Figure 8.17: HIL Z velocity[12].

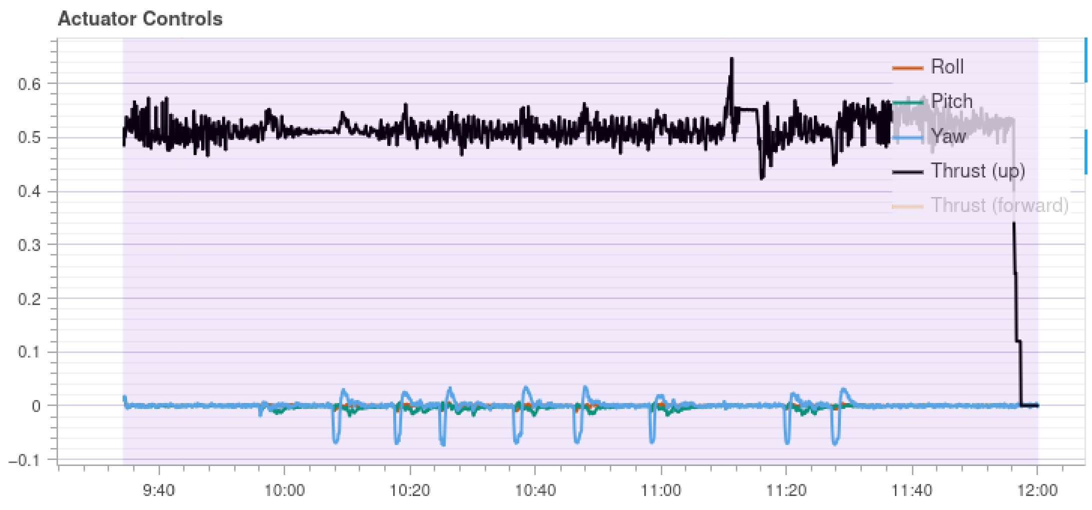


Figure 8.18: HIL normalized control input[12].

9 Manual Flight

The first step in testing the INDI controller through experimental flight is to evaluate the inner loop to ensure that any issues originate from the attitude controller. Once the inner loop is properly tuned, the position controller can be introduced and tested.
To assess only the inner loop, the stabilized mode has been selected. In this mode, the inner loop stabilizes the drone's attitude while a pilot controls it using an RC. For autonomous missions, a MOCAP system is required to provide global position data when testing flights in an indoor arena. Otherwise, a GPS sensor is necessary for outdoor tests.

9.1 Manual mode

The Stabilized manual mode stabilizes and levels the multicopter when the RC control sticks are centred. To move/fly the vehicle you move the sticks outside of the centre.
When sticks are outside the centre, the roll and pitch sticks control the angle of the vehicle (attitude) around the respective axes, the yaw stick controls the rate of rotation above the horizontal plane, and the throttle controls altitude/speed[6]. The pilot input are sent as a roll and pitch commands angle and a yaw rate command, otherwise if the stick is centred the drone will loiter.

9.2 Experimental flight

Due to inertia uncertainties, several flight tests have been conducted, analysing the log data from each flight and incorporating feedback from the pilot.
The PD control law for the attitude is defined as:

$$\Delta u = K(P * \Omega_{err} - D * \frac{\partial \Omega_{err}}{\partial t}) \tag{9.1}$$

For each flight, the value of K was incrementally increased to achieve a dynamic response similar to that observed in the simulation, while also increasing the absolute value of the incremental control.
In the following pages a table with the control gains used and RMSE and L1 error are shown but, the imagine of the attitude behaviour are from only two flights will be presented—the most significant ones: the flight with the best results and a flight conducted with a foam plate attached to the front of the drone. The latter was performed to alter the drone's dynamics and assess the controller's robustness against unmodeled dynamics.

Tuning rate PD gain First has been tune the rate controller as mention before scaling the PD controller by a K factor

	K-roll/pitch rate	K-yaw rate
Flight 1	1.5	1.0
Flight 2	1.8	1.5
Flight 3	2.0	1.5
Flight 4	3.0	1.5
Flight 5	4.0	1.5

Table 10: Control gains values for rate control loop.

	Flight 1	Flight 2	Flight 3	Flight 4	Flight 5
RMSE angle roll	0.106	0.2242	0.1323	0.1026	0.0622
RMSE angle pitch	0.0929	0.0962	0.1092	0.0795	0.0653
RMSE angle yaw	0.0063	0.0082	0.1913	0.0053	0.0039

Table 11: Error evaluation from each flight.

	Flight 1	Flight 2	Flight 3	Flight 4	Flight 5
RMSE rate roll	0.1704	0.3957	0.2484	0.109	0.0743
RMSE rate pitch	0.1833	0.2311	0.2096	0.169	0.1107
RMSE rate yaw	0.6324	0.6050	0.4555	0.652	0.2285

Table 12: Error evaluation from each flight.

	Flight 1	Flight 2	Flight 3	Flight 4	Flight 5
L1 angle roll	1.250e4	1.490e4	2.8879e4	1.268e4	1.511e4
L1 angle pitch	9.936e3	7.730e3	2.9450e4	1.446e4	1.686e4
L1 angle yaw	0.479e3	0.457e3	4.1390e3	0.359e3	0.523e3

Table 13: Error evaluation from each flight.

	Flight 1	Flight 2	Flight 3	Flight 4	Flight 5
L1 rate roll	1.380e3	1.559e3	2.965e3	1.141e3	1.357e3
L1 rate pitch	1.347e3	1.100e3	3.541e3	1.588e3	1.873e3
L1 rate yaw	6.116e3	4.871e3	1.162e4	8.792e3	5.368e3

Table 14: Error evaluation from each flight.

Then also the attitude proportional P gain has been tested to increase the dynamic response, using the value of the last experimental fight for the rate controller. The $Flight4$ has shown double to highlights the P-gain for the attitude used in the previous flight.

	P-roll/pitch angle	K-yaw angle
Flight 5	2.5	1.0
Flight 6	5	2.5
Flight 7	3.5	2.5

Table 15: Control gains values for attitude control loop.

	Flight 6	Flight 7
RMSE angle roll	0.0696	0.0567
RMSE angle pitch	0.0581	0.0596
RMSE angle yaw	0.0012	0.0048

Table 16: Error evaluation from each flight.

	Flight 6	Flight 7
RMSE rate roll	0.164	0.1194
RMSE rate pitch	0.212	0.1571
RMSE rate yaw	0.3503	0.292

Table 17: Error evaluation from each flight.

	Flight 6	Flight 7
L1 angle roll	4.225e3	7.176e3
L1 angle pitch	3.694e3	8.128e3
L1 angle yaw	0.041e3	0.318e3

Table 18: Error evaluation from each flight.

	Flight 6	Flight 7
L1 rate roll	0.762e3	0.993e3
L1 rate pitch	0.897e3	1.321e3
L1 rate yaw	2.735e3	3.893e3

Table 19: Error evaluation from each flight.

In the following pages is shown the graphical evolution of RMSE and L1 error.

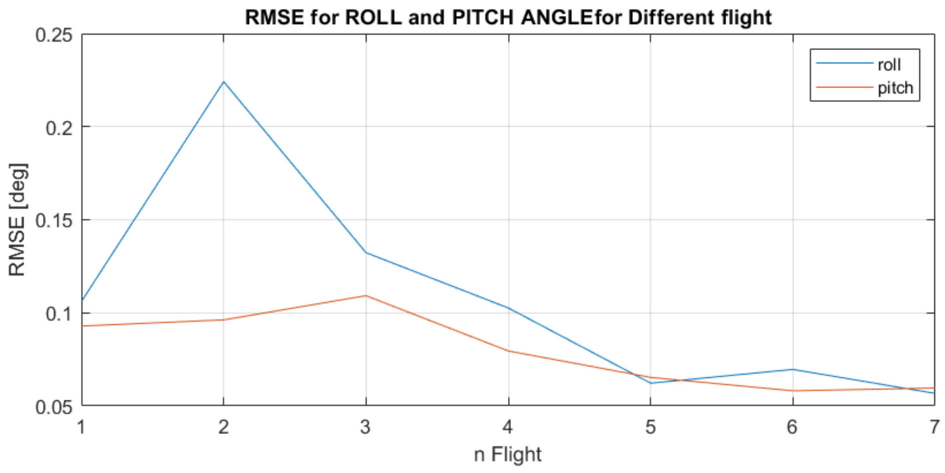


Figure 9.1: Roll and pitch angle RMSE error.

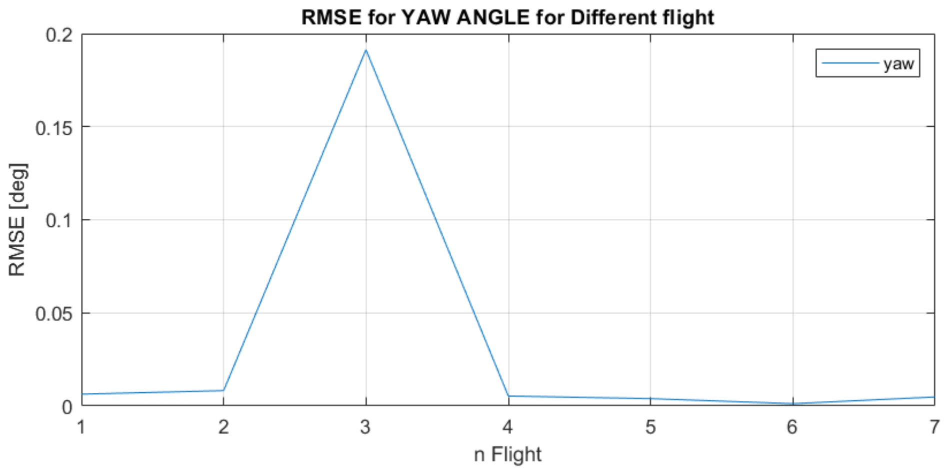


Figure 9.2: Yaw angle RMSE error.

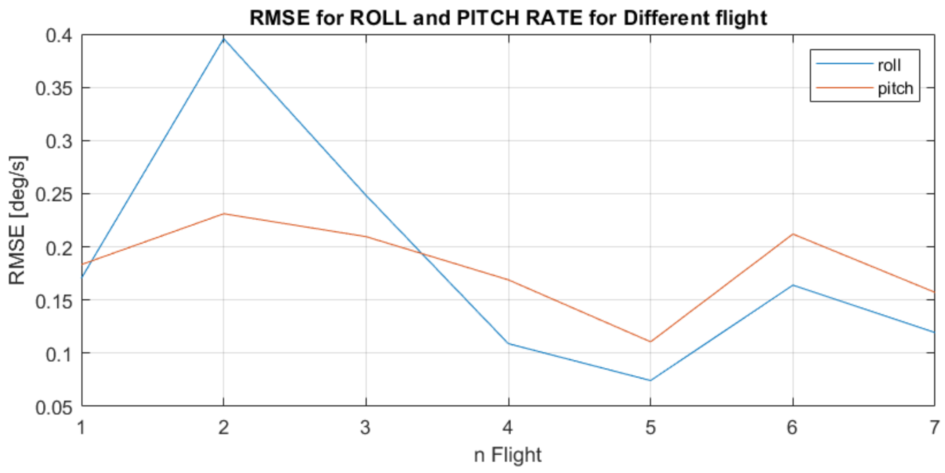


Figure 9.3: Roll and pitch rate RMSE error.

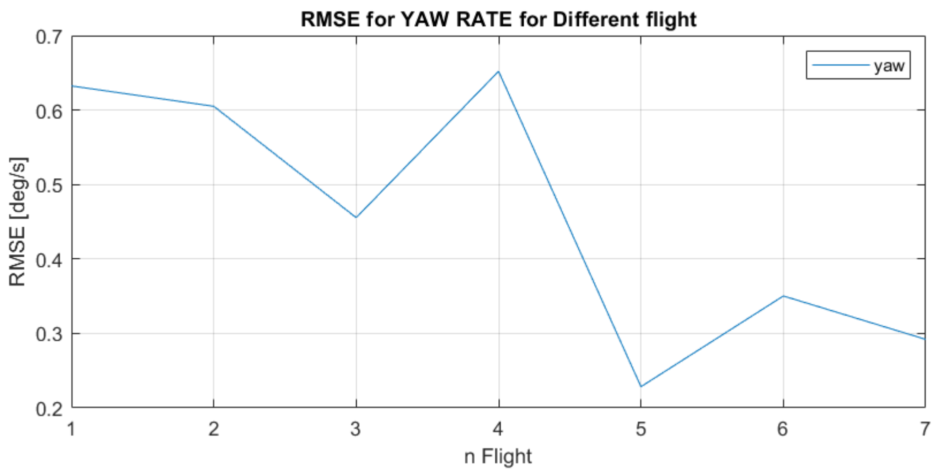


Figure 9.4: Yaw rate RMSE error.

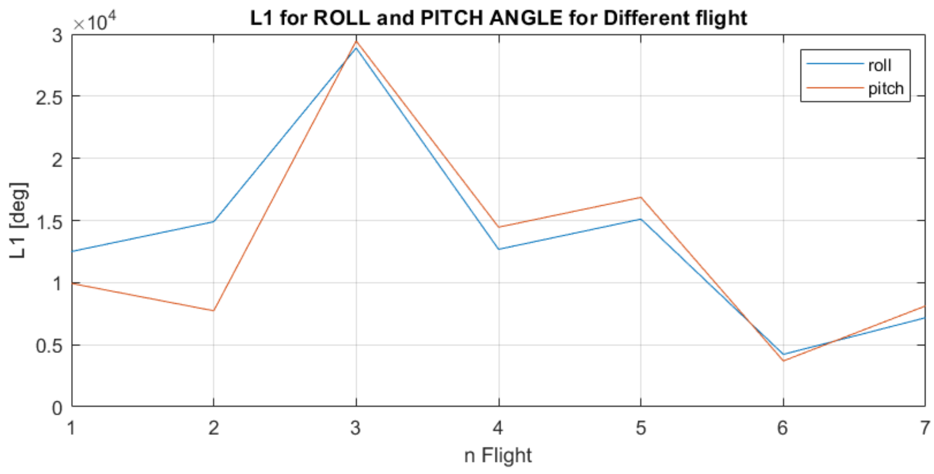


Figure 9.5: Roll and pitch angle L1 error.

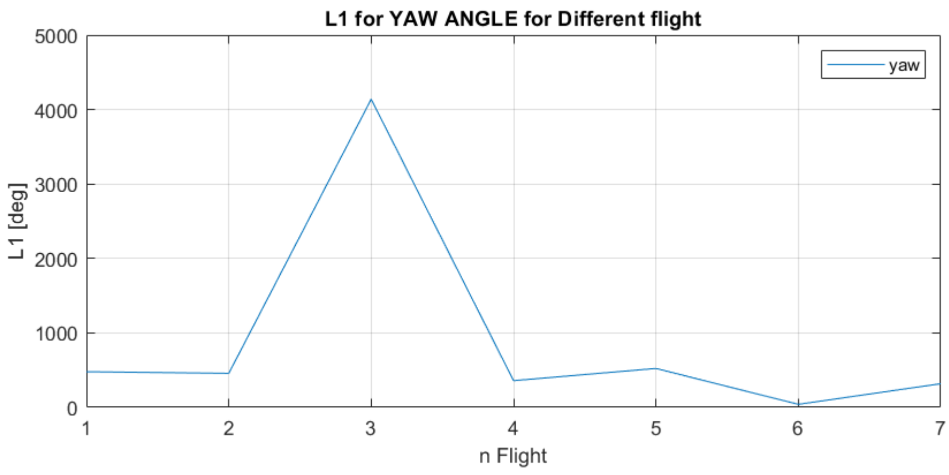


Figure 9.6: Yaw angle L1 error.

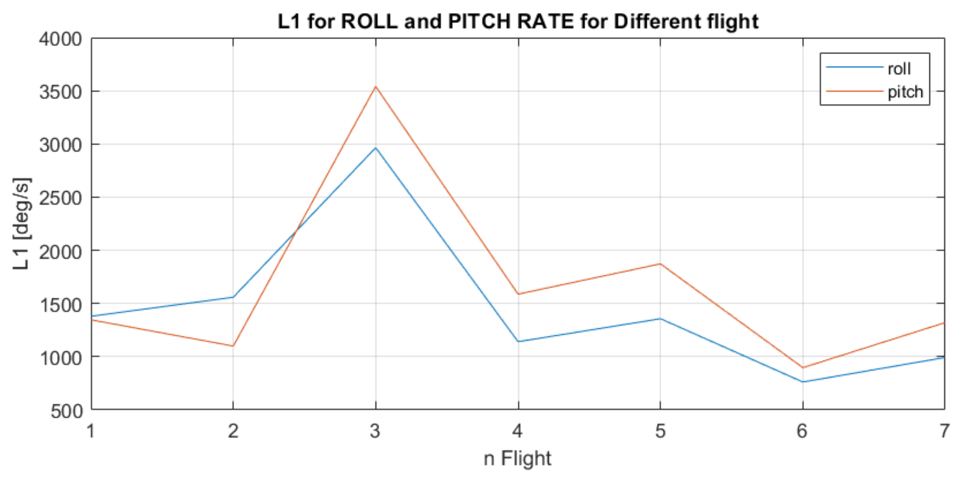


Figure 9.7: Roll and pitch rate L1 error.

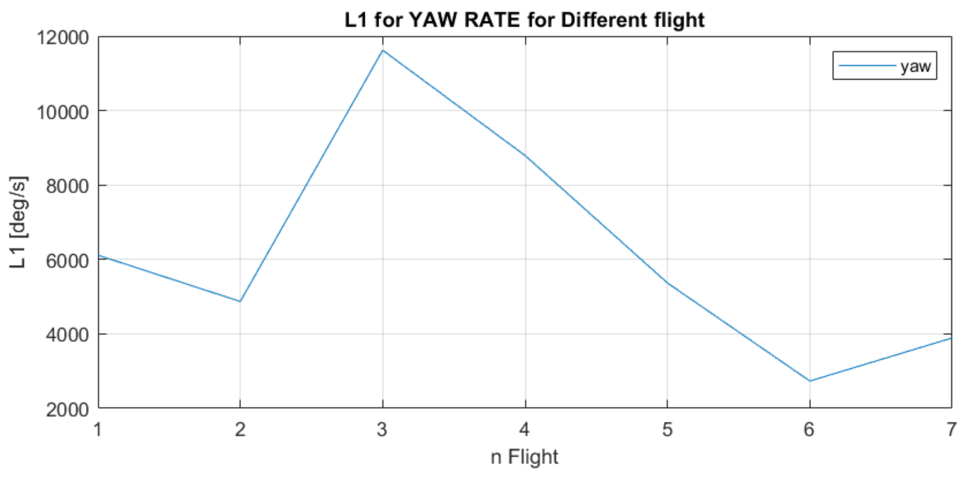


Figure 9.8: Yaw rate L1 error.

At this point for the dynamic response of the last flight is shown.

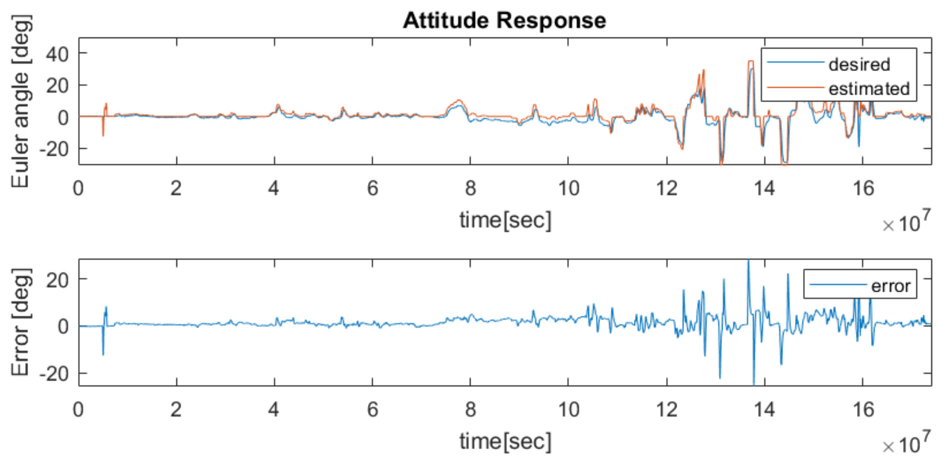


Figure 9.9: Roll angle error.

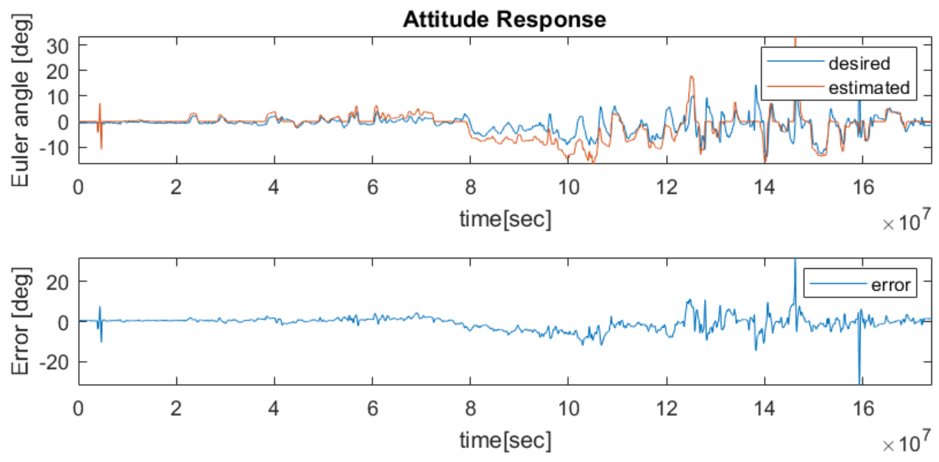


Figure 9.10: Pitch angle error.

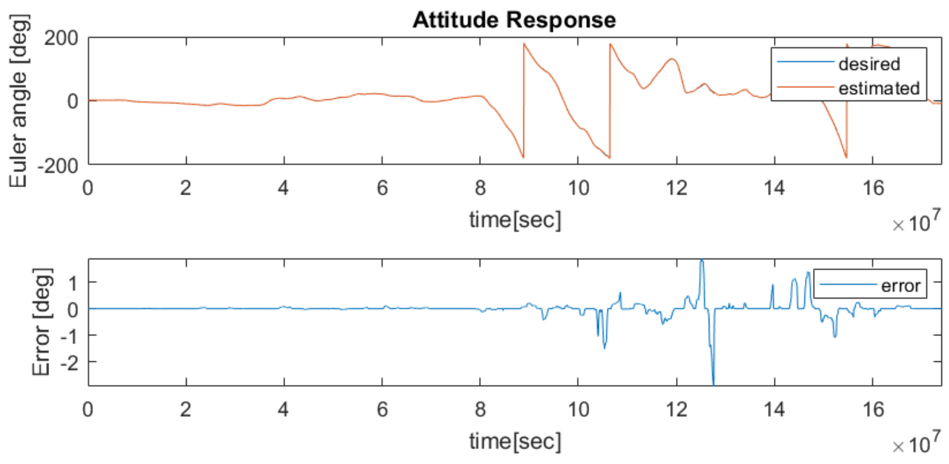


Figure 9.11: Yaw angle error.

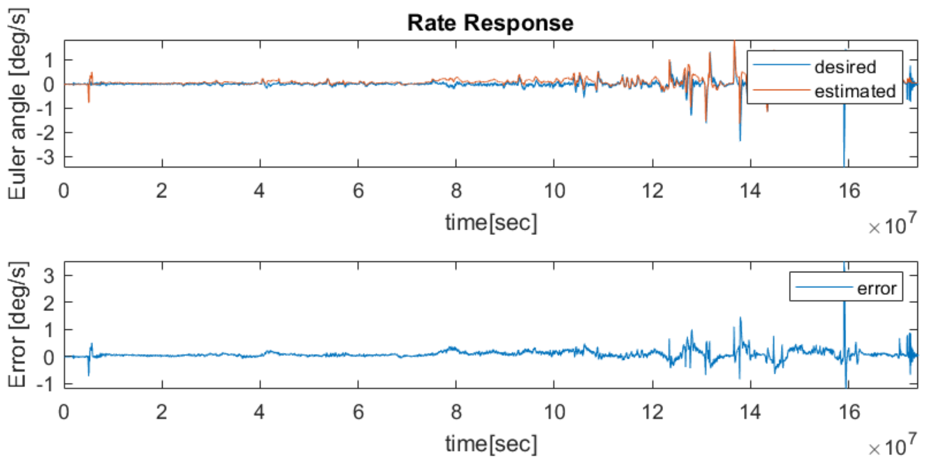


Figure 9.12: Roll rate error.

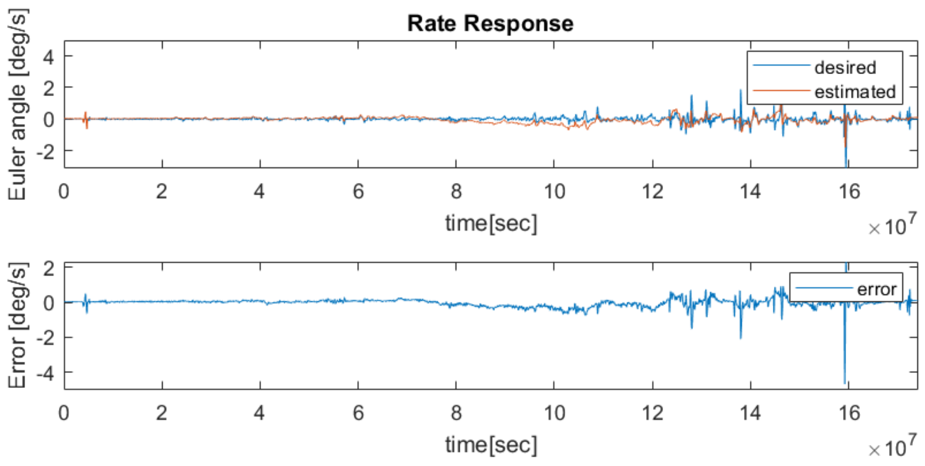


Figure 9.13: Pitch rate error.

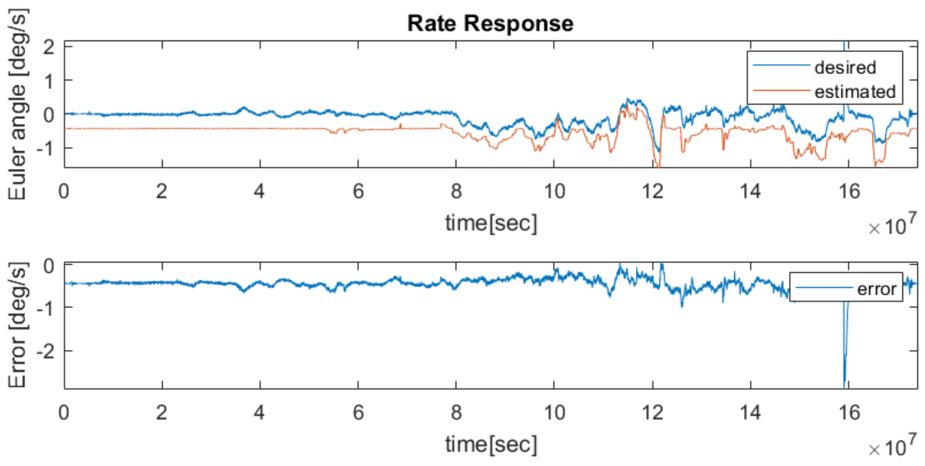


Figure 9.14: Yaw rate error.

9.3 Manual flight with disturbances

At this point a plate of foam has been added in front of the drone, increasing the speed this plate will generate a drag force opposite at the forward direction and a pitch down moment. In this flight the speed velocity has been increased to test the response, the same control gain of the last flight has been used in this test.

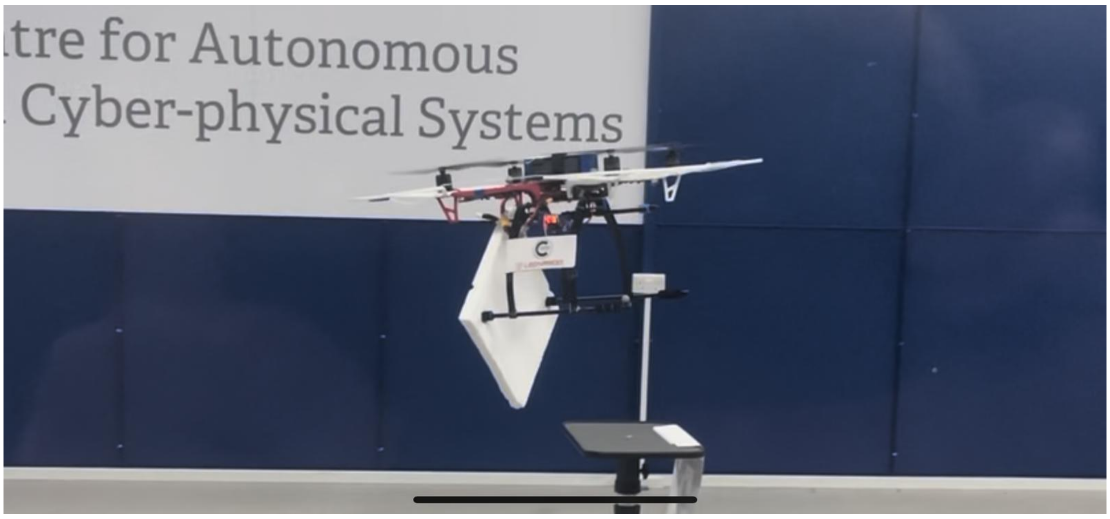


Figure 9.15: Drone with the foam plate.

	Flight 6	Flight with foam plate	Δerr[%]
RMSE angle roll	0.0567	0.0427	-24%
RMSE angle pitch	0.0596	0.0632	+6.0%
RMSE angle yaw	0.0048	0.1670	+330%

Table 20: Error evaluation from each flight.

	Flight 6	Flight with foam plate	Δerr[%]
RMSE rate roll	0.1194	0.0825	-30%
RMSE rate pitch	0.1571	0.1451	-7.6%
RMSE rate yaw	0.2920	0.2593	-11%

Table 21: Error evaluation from each flight.

	Flight 6	Flight with foam plate
L1 angle roll	7.176e3	1.00e4
L1 angle pitch	8.128e3	1.794e4
L1 angle yaw	0.318e3	2.478e3

Table 22: Error evaluation from each flight.

	Flight 6	Flight with foam plate
L1 rate roll	0.993e3	1.00e3
L1 rate pitch	1.321e3	2.730e3
L1 rate yaw	3.893e3	5.590e3

Table 23: Error evaluation from each flight.

At this point for the dynamic response of the flight is shown.

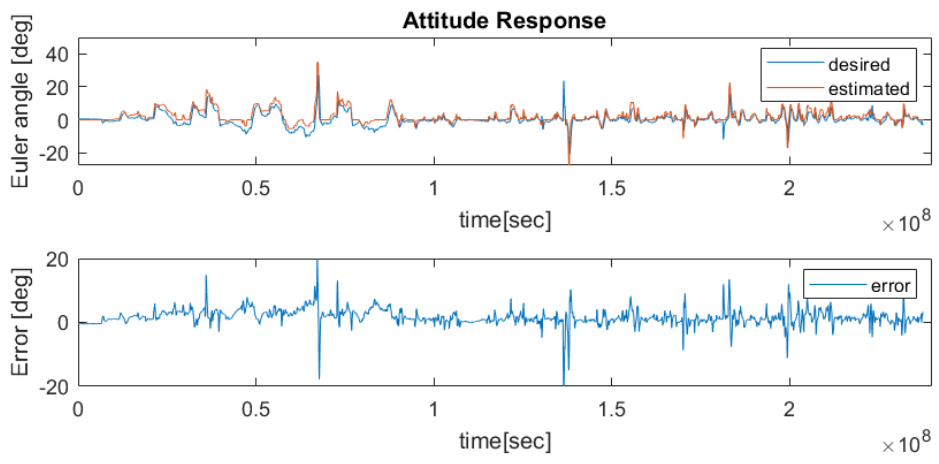


Figure 9.16: Roll angle error.

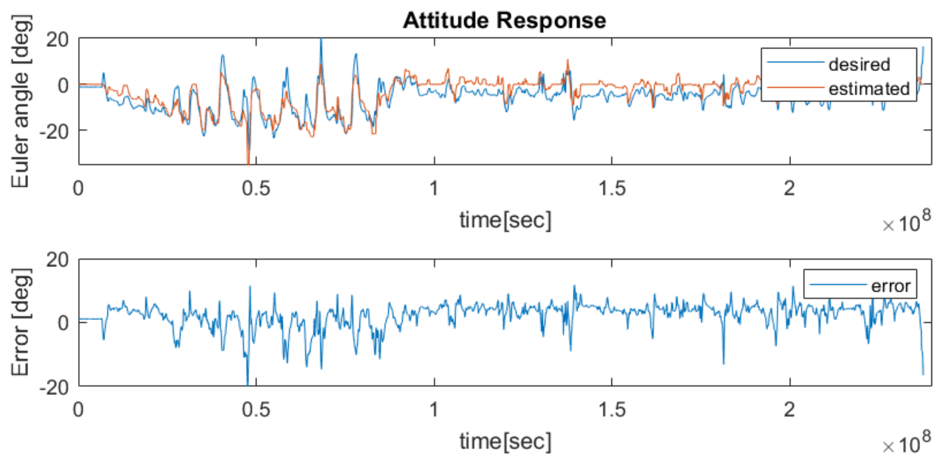


Figure 9.17: Pitch angle error.

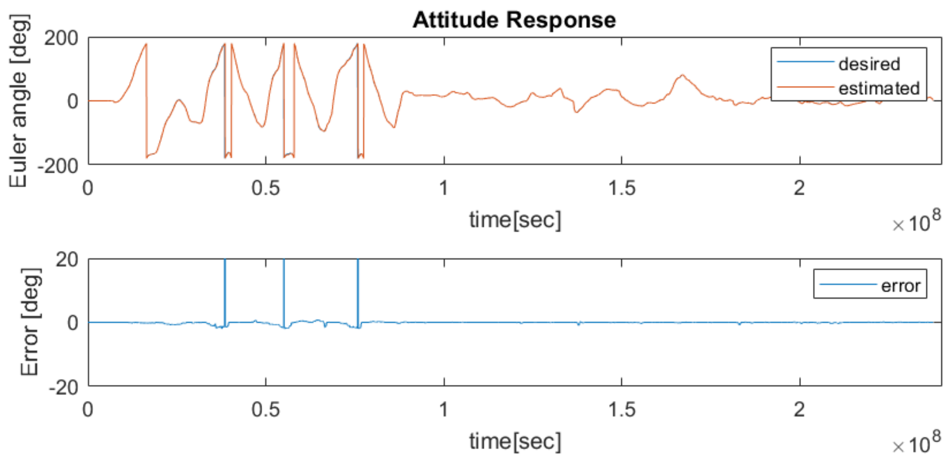


Figure 9.18: Yaw angle error.

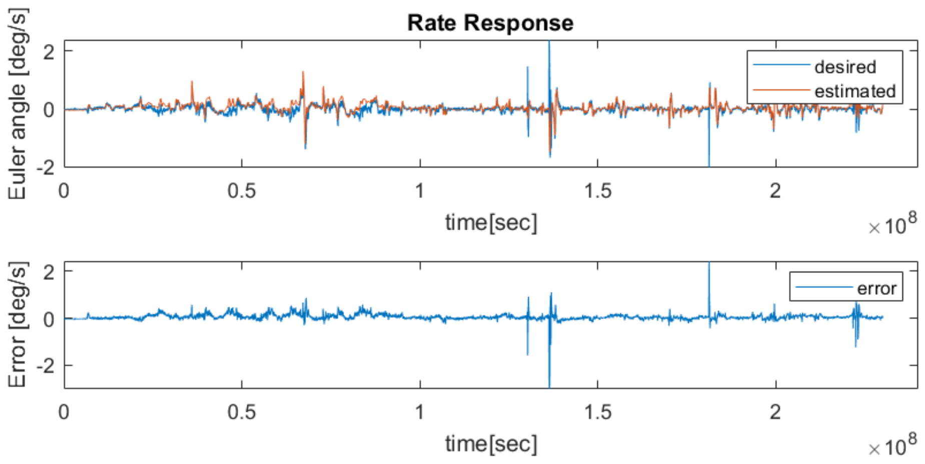


Figure 9.19: Roll rate error.

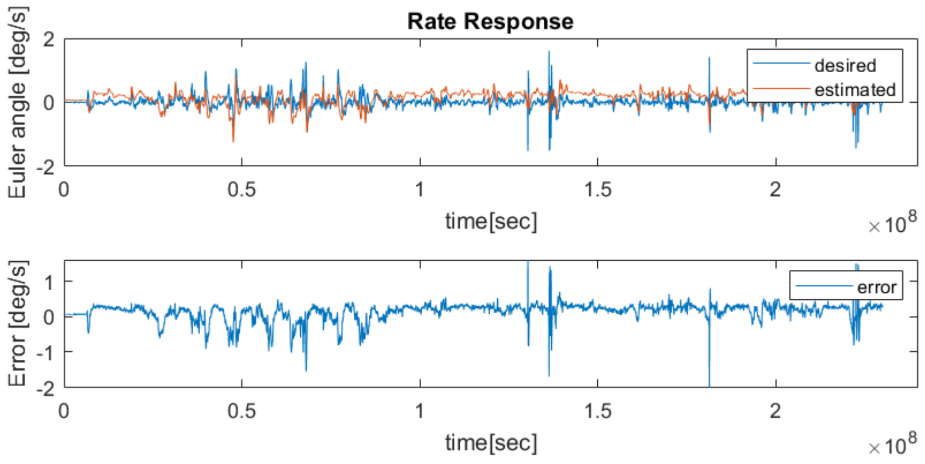


Figure 9.20: Pitch rate error.

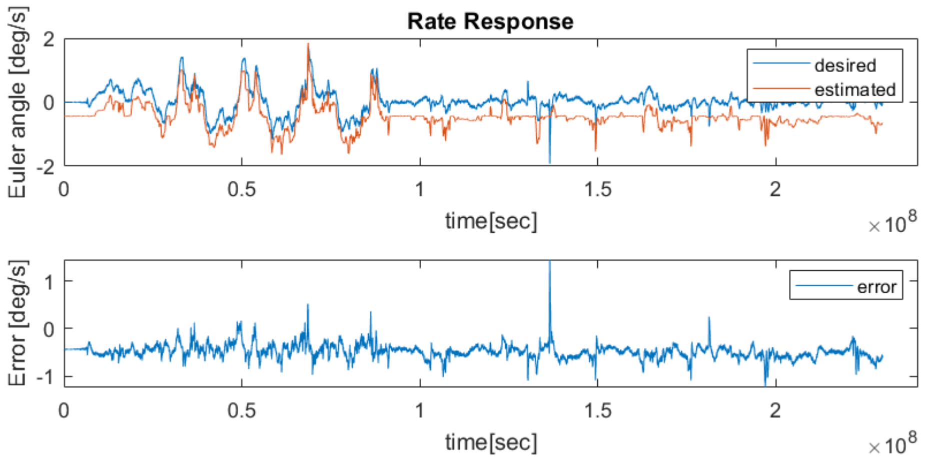


Figure 9.21: Yaw rate error.

10 Conclusion and further work

10.1 Conclusion

10.2 Further work

References

[1] Boukas F. A. “Implementazione sistema di controllo VTOL su UAV”. MA thesis. Politecnico di Torino, 2024.

[2] Dmitry Ignatyev Antonios Tsourdos and Sabyasachi Mondal. *Uncrewed Aerial System: Guidance and Control*. Elsevier Inc., 2024.

[3] Mirela Coman Bogdan-Vasilie Cioruja. “Considerations on the dynamic system study: from definition and classification to analysis and interpretation of behavior”. In: *SCIENTIFIC RESEARCH AND EDUCATION IN THE AIR FORCE* (2018).

[4] Hirce S. Capone A. “Backstepping for partially unknown nonlinear systems using gaussian processes”. In: *IEEE Control Systems Letter 3(2), 416-421* (2019).

[5] Qiping Chu Ewoud J. J. Smeur and Guido C. H. E. de Croon. “Adaptive Incremental Nonlinear Dynamic Inversion for Attitude Control of Micro Air Vehicles”. In: *Journal of Guidance, Control and Dynamics* (2016). DOI: 10.2514/1.G001490.

[6] PX4 Autopilot User Guide. *PX4 Autopilot User Guide*. https://docs.px4.io/main/en/, 2025.

[7] Gomez A. P. Camacho O. Herrera M. Chamorro W. “Sliding mode control: An approach to control a quadrotor.” In: *Proceedings of the 2015 Asia-Pacific Conference on Computer Aided System Engineering* (2015).

[8] Cao C. Hovakimyan N. *L1 adaptive control theory: guaranteed robustness with fast adaptation*. SIAM, 2010.

[9] PX4 motor model. URL: https://github.com/mvernacc/gazebo_motor_model_docs/blob/master/notes.pdf.

[10] https://github.com/PX4/PX4-Autopilot, GitHub PX4’s firmware link.

[11] https://grabcad.com/library/dji-f450-quadcopter-drone-1, CAD model.

[12] Flight Review for SIL. URL: https://review.px4.io/plot_app?log=7afca003-ae10-4cb8-b8d8-e67626fdbae2.

[13] https://www.worldronemarket.com/product/t-motor-air-gear-450, Seller website.

[14] White B. A. Kim S. Tsourdos A. *Nonlinear Flight Control Systems*. John Wiley Sons, 2010.

[15] Mohammadi K. L’Afflitto A. Anderson R.B. “An introduction to nonlinear robust control for unmanned quadrotor aircraft: How to design control algorithms for quadrotors using sliding mode control and adaptive control techniques [focus on education].” In: *IEEE Control Systems Magazine 38* (2018).

[16] Tiga Ho Yin Leung. “Development of the multirotor drone automatic landing system using optimal guidance and non linear control”. Bedford: Cranfield University, 2022.

[17] L’Afflitto A. Marshall J.A. Sub W. “A survey of guidance, navigation and control system for autonomous multi-rotor small unmanned aerial systems.” In: *Annual reviews in Control 52* (2021). URL: https://doi.org/10.1016/j.arcontrol.2021.10.013.

[18] Rupp C. Pohl K. *Requirements Engineering Fundamentals: A Study Guide for the Certified Professional for Requirements Engineering Exam*. Rocky Nook, 2015.

[19] Quan Quan. *Introducing to multicopter design and control*. Springer, 2017. DOI: 10.1007/978-981-10-3382-7.

[20] Whidborne J.F. Shaik M.K. "Robust sliding mode control of a quadrotor." In: *UKACC 11th International Conference on Control (CONTROL)* (2016).

[21] Postlethwaite I. Skogestad S. *Multivariable Feedback Control: Analysis and Design*. Jhon Wiley & Sons, 2007.

[22] Lewis F. Steven B. *Aircraft Control and Simulation*. Wiley, 2003.

[23] Surekha P. Sumathi S. Eswaran S. *Real-Time Simulation and Hardware-in-the-Loop Testing Using Typhoon HIL*. Springer, 2023.

[24] Karaman S. Tal E. "Accurate tracking of aggressive quadrotor trajectories using incremental nonlinear dynamic inversion and differential flatness". In: *IEEE Contol systems technology* (2021).

[25] Chen X. Wang S. Zhang Y. *Software-in-the-Loop Simulation for Early-Stage Testing of AUTOSAR Software Components*. IEEE, 2016.