Comparison of NMPC and GPU-Parallelized MPPI for Real-Time UAV Control on Embedded Hardware

applied sciences

MDPI

Article

Comparison of NMPC and GPU-Parallelized MPPI for Real-Time UAV Control on Embedded Hardware

Riccardo Enrico *, Mauro Mancini and Elisa Capello

Department of Mechanical and Aerospace Engineering, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy; mauro.mancini@polito.it (M.M.); elisa.capello@polito.it (E.C.)
* Correspondence: riccardo.enrico@polito.it

Abstract

Unmanned Aerial Vehicles (UAVs) operating in complex environments require advanced control strategies beyond traditional linear approaches. This work presents a comprehensive comparative analysis of Nonlinear Model Predictive Control (NMPC) and Model Predictive Path Integral (MPPI) control for UAV trajectory tracking, with an emphasis on real-time implementation feasibility on embedded hardware. A modular ROS 2 framework enables runtime controller selection using CasADi/Acados for NMPC and JAX for MPPI implementations. Processor-in-the-Loop experiments on NVIDIA Jetson Orin Nano hardware evaluate computational performance under realistic resource constraints. Results demonstrate that MPPI achieves superior tracking performance, with an 18.6% improvement in overall RMSE compared to NMPC (0.8480 m to 0.6897 m) for trajectory following. Both controllers achieve real-time performance on embedded hardware, with GPU acceleration proving critical for MPPI success, enabling a 17.63 ms median computation time versus 31.02 ms for CPU-only execution. Systematic parameter analysis reveals optimal MPPI configurations of 40 horizon steps and 800–1250 samples for balancing performance with computational constraints imposed by the 50 Hz (20 ms) control frequency inherent to PX4 hardware compliance. This study validates that mainstream computational frameworks can deliver satisfactory real-time control performance on standard robotics hardware, significantly enhancing accessibility for practical UAV deployment while providing clear guidelines for control strategy selection in resource-constrained applications.

Keywords: UAV control; Model Predictive Control; MPPI; real-time control; embedded systems; GPU acceleration; trajectory tracking; ROS 2

Academic Editor: Alessandro Gasparetto

Received: 4 July 2025
Revised: 13 August 2025
Accepted: 15 August 2025
Published: 19 August 2025

Citation: Enrico, R.; Mancini, M.; Capello, E. Comparison of NMPC and GPU-Parallelized MPPI for Real-Time UAV Control on Embedded Hardware. *Appl. Sci.* **2025**, *15*, 9114. https://doi.org/10.3390/app15169114

1. Introduction

1.1. Motivation

Unmanned Aerial Vehicles (UAVs) operating in real-world environments face significant challenges from external disturbances, including wind gusts, payload dynamics, and aerodynamic uncertainties [1,2].

Additionally, electromagnetic interference (EMI) presents a critical challenge to UAV control systems, as modern UAVs rely heavily on electronic components susceptible to electromagnetic disturbances. EMI encompasses both unintentional sources (power lines, radio frequency equipment, and on-board electronics) and intentional EMI (IEMI) involving adversarially crafted emissions designed to compromise UAV electronics and sensors [3]. EMI can severely compromise critical subsystems, including datalink communications, GPS navigation, and motor control circuits, potentially leading to complete loss of vehicle

control [4]. These perturbations particularly affect trajectory tracking during aggressive maneuvers or near performance limits, where traditional linear control approaches often prove inadequate. This vulnerability is exacerbated by EMI-induced sensor degradation and actuator malfunctions, posing significant concerns for advanced nonlinear control strategies that depend on reliable feedback and precise actuator response.

Model Predictive Control (MPC) strategies have emerged as compelling alternatives due to their ability to handle constraints, predict future system behavior, and optimize control actions over finite horizons. However, the highly nonlinear dynamics of UAVs often exceed the capabilities of linear MPC formulations [5], driving research toward Nonlinear MPC (NMPC) implementations that explicitly account for nonlinear dynamics [1,6]. Importantly, these advanced control strategies must also account for the potential degradation of control performance under EMI conditions, where sensor noise increases and actuator response becomes unreliable. Since these EMI effects are difficult to model precisely, in this paper they have been represented as general noise added to the ground truth position measurements. This approach allows us to evaluate whether the trajectory tracking performance of the predictive control algorithms deteriorates under such disturbances.

Despite these advantages, NMPC faces a fundamental computational challenge. The algorithms rely on Sequential Quadratic Programming (SQP) or SQP–Real-Time Iteration (SQP-RTI) solvers that offer limited parallelization opportunities [6,7]. This computational bottleneck becomes critical when the control strategies are implemented on resource-constrained embedded systems typical of both small and medium-sized UAVs, thus raising the following question: which predictive control algorithm can be effectively parallelized on real robotics hardware while maintaining real-time feasibility?

Model Predictive Path Integral (MPPI) control emerges as a promising candidate due to its inherently parallelizable structure [8,9]. However, existing implementations reveal significant limitations. As described in [1], the MPPI algorithm proposed in [10] requires desktop-class hardware with GPU support. Moreover, it achieves only a 23.13 ms average computation time and exhibits highly oscillatory control behavior. These computational demands and control quality issues represent the primary limitations to practical MPPI deployment on UAVs.

1.2. Problem Statement

This work addresses the critical challenge of selecting and implementing computationally efficient predictive control strategies for UAV trajectory tracking on embedded hardware. While the UAV control literature extensively documents tracking accuracy, a significant gap exists regarding comparative analysis of computational costs between deterministic optimization-based approaches (NMPC) and stochastic sampling-based methods (MPPI).

Current research suffers from three fundamental limitations:

- Implementation opacity: Published works frequently omit software packages and implementation details, imposing limitations on verification and reproducibility, as in [11,12].
- Monolithic architectures: Existing implementations usually embed the entire control logic on the flight MCU, so every time a different strategy (PID, NMPC, MPPI, ...) is tested the firmware must be re-compiled and flashed. Besides hindering fair benchmarking, this approach is limited by the scarce computational budget of typical autopilot processors (e.g., STM32H7 on Pixhawk, 1–2 MB RAM), which is insufficient for NMPC and MPPI, which require hundreds if not thousands of trajectory roll-outs per step. To address these limitations, we offload the computation of both

predictive controllers to a companion computer mounted onboard the UAV. This architecture enables

(i) the computing power needed for real-time NMPC/MPPI;
(ii) the possibility to select the controller at launch time through ROS 2 parameters instead of at compile time, thus greatly simplifying A/B comparisons.

Although the additional communication hop introduces latency overhead, this distributed architecture follows established practices in UAV control systems [13,14] and overcomes the monolithic deployments reported in [15]. The communication latency remains acceptable for the 20 ms control period used in our experiments. Moreover, when multi-node software architectures are used it is possible to use ROS node composition to overcome significant overhead [16].

- Incomplete characterization: Previous studies focus predominantly on tracking accuracy while neglecting computational requirements and software modularity [11,17].

These limitations collectively impede the development of practical guidelines for control strategy selection in resource-constrained UAV applications.

1.3. Related Works

Recent advances in predictive control for UAVs have demonstrated significant progress in achieving real-time performance on embedded platforms, with distinct trajectories for MPPI and NMPC implementations.

1.3.1. NMPC Implementations

The NMPC community has achieved broader platform compatibility, with successful deployments across diverse embedded systems. ARM-based implementations demonstrate control frequencies reaching up to 100 Hz on resource-constrained hardware [18]. The software ecosystem exhibits greater maturity if compared to the MPPI implementations, with CasADi providing significant performance improvements over MATLAB R2024b standard predictive control toolbox [19] and Acados achieving fast performance through structure-exploiting optimization [20]. The open-source nature of these tools, compared to proprietary solutions like MATLAB, R2024b provides additional advantages for deployment on real UAV systems and has been a key consideration in both the NMPC and MPPI implementations in this work.

1.3.2. MPPI Implementations

MPPI is a more recent control approach, first introduced by [8] and comprehensively described in [9]. While still requiring a workstation-class GPU in [10,17], recent advances in GPU acceleration have enabled embedded MPPI deployment, with [11] achieving an onboard control frequency of 100 Hz on quadcopters using NVIDIA Jetson Orin modules. However, their work lacks detailed documentation of the underlying software frameworks and optimization strategies employed. While existing frameworks, such as MPPI-Generic, offer comprehensive C++/Compute Unified Device Architecture (CUDA) libraries [12], they require substantial low-level programming expertise, creating accessibility barriers for researchers more familiar with high-level programming languages.

1.3.3. Research Gaps

Despite individual advances, the literature reveals a striking absence of systematic comparative studies between NMPC and MPPI for UAV applications. Most investigations prioritize control performance while providing minimal analysis of computational complexity, memory utilization, and energy consumption—critical factors for deployment on resource-constrained embedded systems. Additionally, software transparency issues and

monolithic design patterns prevent the development of fair comparative frameworks and standardized benchmarking protocols.

1.4. Contributions

This work presents a comprehensive comparative analysis of NMPC and MPPI control strategies for UAV trajectory tracking, addressing identified gaps through a systematic evaluation framework. Our key contributions include the following:

1. Modular Control Architecture: We develop a unified framework where both controllers operate as outer-loop trajectory trackers, generating standardized control commands for inner-loop attitude control, maintaining clear separation of control responsibilities.

2. Transparent Implementation: Both algorithms are implemented using widely adopted open-source frameworks within ROS 2—CasADi/Acados for NMPC and JAX for MPPI—ensuring reproducibility and accessibility with launch-time controller selection.

3. Comprehensive Evaluation: We provide detailed insights into the practical trade-offs between control performance and computational requirements through Processor-in-the-Loop (PITL) testing on embedded hardware. This includes tracking accuracy, computational overhead, and real-time feasibility. This work evaluates the performance of both control strategies under realistic disturbance conditions, representing EMI effects as general noise added to the ground truth position measurements.

4. Parameter Analysis: MPPI's oscillatory behavior and computational demands through parameter selection is systematically addressed and its performance characterized, establishing practical deployment guidelines.

1.5. Methodology Overview

The computational framework selection demonstrates that widely adopted academic and industrial tools can achieve satisfactory performance on standard robotics hardware. Rather than pursuing highly optimized implementations, this approach validates that mainstream tools and hardware like the NVIDIA Jetson can deliver real-time control performance suitable for UAV applications.

This study employs a PITL experimental framework, where control algorithms are executed on the target embedded hardware (NVIDIA Jetson Orin Nano) while UAV dynamics are simulated. This setup provides realistic computational performance metrics by running algorithms on deployment hardware.

NMPC utilizes CasADi and Acados [20] for efficient nonlinear optimization, while MPPI leverages JAX [17,21] for parallel trajectory sampling and automatic differentiation. JAX's just-in-time (JIT) compilation enables real-time MPPI performance by eliminating Python 3.12 interpreter overhead and enabling aggressive compiler optimizations through XLA, as demonstrated in [17]. This allows thousands of parallel trajectory evaluations to be achieved in real time.

1.6. Paper Structure

The remainder of this paper is organized as follows. Section 2 provides a background on UAV dynamics modeling and detailed formulations of both NMPC and MPPI control strategies. Section 3 describes the system architecture, hardware platform, and software implementation and outlines the experimental methodology, performance metrics, and test scenarios. Section 4 presents comprehensive results comparing computational performance, tracking accuracy, and real-time feasibility. Finally, Section 5 concludes with future research directions.

2. Background

2.1. Mathematical Model of the UAV

The UAV is modeled as a rigid body with mass m and inertia tensor J based on the Holybro X500 quadrotor configuration (Table 1), as seen in [1,22].

Table 1. Quadrotor (Holybro X500) physical parameters used in the model.

Parameter	Symbol	Value
Mass	m	2.0 kg
Inertia (x-axis)	J_{xx}	0.02166 kg·m^2
Inertia (y-axis)	J_{yy}	0.02166 kg·m^2
Inertia (z-axis)	J_{zz}	0.04 kg·m^2

The motion is described using a body-fixed frame $\mathcal{F}_B$ in the FLU (Forward–Left–Up) convention attached to the UAV and an inertial frame $\mathcal{F}_I$ in the ENU (East–North–Up) convention.

The UAV state comprises the position $p \in \mathbb{R}^3$ (in ENU), linear velocity $v \in \mathbb{R}^3$ (in ENU), attitude quaternion $q \in \mathbb{R}^4$ (representing rotation from ENU to FLU), and angular velocity $\omega \in \mathbb{R}^3$ (in FLU). The dynamics are governed by the Newton–Euler equations:

$$\begin{aligned} \dot{p} &= v, & \dot{q} &= \tfrac{1}{2} q \odot \begin{bmatrix} 0 \\ \omega \end{bmatrix}, \\ \dot{v} &= \tfrac{1}{m} R(q) F_t + g, & \dot{\omega} &= J^{-1}(\tau - \omega \times (J\omega)), \end{aligned} \tag{1}$$

where $\odot$ denotes standard Hamilton quaternion multiplication, $R(q)$ is the rotation matrix that transforms vectors from the body frame (FLU) to the inertial frame (ENU), and $F_t = [0, 0, T_b]^T$ is the thrust vector in the body frame, with T_b being the collective thrust magnitude produced by the propellers along the positive body z-axis (Up). The vector $g = [0, 0, -9.81]^T$ m/s² represents the gravitational acceleration in the inertial ENU frame. The vector $\tau \in \mathbb{R}^3$ represents the control torques in the body frame, and the inertia tensor J is defined with respect to the FLU body frame.

2.2. Control Algorithms

This section proposes the theoretical background behind the NMPC and MPPI controllers, which are presented and compared in this paper. The control architecture encompasses trajectory controllers operating as the outer loop of the UAV control stack, thereby sending angular rates and collective thrust commands to the inner-loop attitude controller, as illustrated in Figure 1. The following subsections provide a detailed description of the control strategies and the UAV dynamic model employed for the controller design.

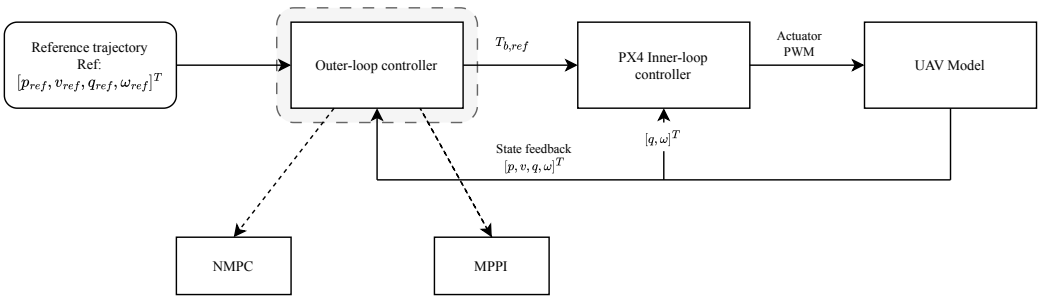


Figure 1. Visualization of the general control scheme. The NMPC and MPPI controllers can be selected at launch time through a ROS 2 launch file. The outer loop is required to run at a 50 Hz control frequency, consistent with the outer loop of the standard PX4 Autopilot [23].

2.2.1. UAV Dynamic Model Used in the Controllers

Both controllers are predictive in nature and rely on a model of the quadrotor. This model, while slightly different, is based on the one presented in Section 2.1 and is described in detail below.

Following the standard Newton–Euler formulation for rigid multi-rotor platforms, the quadrotor is modeled as follows. The state vector is

$$x = \begin{bmatrix} p^\top & v^\top & q^\top \end{bmatrix}^\top \in \mathbb{R}^{10},$$

with position $p \in \mathbb{R}^3$, velocity $v \in \mathbb{R}^3$ (both expressed in the inertial frame), and unit quaternion $q = [q_w, q_x, q_y, q_z]^\top \in \mathbb{R}^4$ describing attitude. The control input is

$$u = \begin{bmatrix} T_b & \omega^\top \end{bmatrix}^\top \in \mathbb{R}^4,$$

where T_b is the total thrust produced by the four rotors (positive along $+z_b$) and $\omega = [\omega_x, \omega_y, \omega_z]^\top$ are the body-frame angular rates commanded by an inner-loop attitude controller.

Then, the 10-state nonlinear Newton–Euler model used for the controllers is given in Equations (2)–(4):

$$\dot{p} = v, \tag{2}$$

$$\dot{v} = \frac{1}{m} R(q) \underbrace{\begin{bmatrix} 0 \\ 0 \\ T_b \end{bmatrix}}_{F_T} + g, \qquad g = \begin{bmatrix} 0 \\ 0 \\ -9.81 \end{bmatrix} \mathrm{m/s^2}, \tag{3}$$

and the quaternion time derivative is

$$\dot{q} = \tfrac{1}{2}\, \Omega(\omega)\, q, \quad \Omega(\omega) = \begin{bmatrix} 0 & -\omega^\top \\ \omega & -[\omega]_\times \end{bmatrix}, \tag{4}$$

where $[\omega]_\times$ denotes the skew-symmetric matrix of ω.

The rotor-induced moments and motor dynamics are handled by the inner-loop controller. Consequently, the outer-loop model can treat the system as a rigid body actuated by the total thrust vector T_b (expressed in the body frame) and the body angular rates ω. This separation allows the outer-loop design to focus on rigid-body motion while abstracting away fast actuator dynamics.

2.2.2. Nonlinear Model Predictive Control—NMPC

The control is formulated by adopting the state and input definitions given in Section 2.2.1, namely $x = [p^\top, v^\top, q^\top]^\top \in \mathbb{R}^{10}$ and $u = [T_b, \omega^\top]^\top \in \mathbb{R}^4$.

The finite-horizon optimal control problem is

$$\begin{aligned} \min_{x(\cdot),\, u(\cdot)} \; & \sum_{k=0}^{N-1} \left\| x_k - x_{ref,k} \right\|_Q^2 + \left\| u_k - u_{ref,k} \right\|_R^2 \\ & + \left\| x_N - x_{ref,N} \right\|_P^2 \\ \text{s.t.} \; & x_{k+1} = f(x_k, u_k), \\ & u_{\min} \le u_k \le u_{\max}, \\ & x_0 = x_{\text{current}}. \end{aligned} \tag{5}$$

Here $f(\cdot)$ is the nonlinear Newton–Euler model (3)–(4), Q, R, and P are positive semidefinite weighting matrices, and the bounds $u_{\min}$, $u_{\max}$ encode actuator limits.

2.2.3. Model Predictive Path Integral (MPPI) Control

Model Predictive Path Integral (MPPI) control was first introduced by Williams et al. [8], with its theoretical foundations extensively analyzed in [9]. MPPI is a sampling-based predictive control algorithm that employs Monte Carlo methods instead of gradient-based optimization. This approach leverages principles of information theory and importance sampling to handle general, non-convex cost functions and constraints. The theoretical foundation stems from the connection between optimal control and statistical mechanics, where the control problem is reformulated as finding the optimal probability distribution over control sequences.

MPPI is a sampling-based stochastic optimal control algorithm that solves the finite-horizon optimal control problem

$$J^* = \min_{u_{0:N-1}} \mathbb{E}\left[\sum_{k=0}^{N-1} q(x_k, u_k) + \phi(x_N)\right]. \tag{6}$$

Similarly to the NMPC formulation in Equation (5), the MPPI cost function in Equation (6) consists of a sum of stage costs $q(x_k, u_k)$ and a terminal cost $\phi(x_N)$. In the standard case, $\phi(x_N)$ can be chosen as a quadratic penalty on the final state, e.g., $\phi(x_N) = \|x_N - x_{\text{ref},N}\|_P^2$, matching the terminal cost used in NMPC. Furthermore, the cost function (6) incorporates multiple objectives for UAV control:

$$\begin{aligned} q(x_k, u_k) = {} & \|p_k - p_{\text{ref},k}\|_{Q_p}^2 + \|v_k - v_{\text{ref},k}\|_{Q_v}^2 \\ & + d_q(q_k, q_{\text{ref},k}) + \|\omega_k - \omega_{\text{ref},k}\|_{Q_\omega}^2 \\ & + \|u_k\|_R^2 + \|\Delta u_k\|_{R_\Delta}^2 \end{aligned} \tag{7}$$

where $d_q(q_1, q_2)$ represents the quaternion orientation error metric and is defined as $dq = 1 - \langle q_1, q_2 \rangle^2$, where $\langle \cdot, \cdot \rangle$ denotes the inner product between unit quaternions q_1 and q_2.

Equation (6) represents the standard MPPI cost function, consisting of a sum of stage costs $q(x_k, u_k)$ and a terminal cost term $\phi(x_N)$. In our implementation, the stage cost component was defined as shown in Equation (7). This expression includes penalties on position, velocity, orientation, angular velocity, and control input magnitude, all of which are standard in trajectory tracking tasks. The orientation error is quantified using the quaternion distance metric d_q defined above. In addition to these standard terms, we introduced a penalty on the control input rate, defined as $\|\Delta u_k\|_{\Delta R}^2$, with $\Delta u_k = u_k - u_{k-1}$. This regularization term is not part of the standard MPPI formulation but was found to significantly improve control smoothness and reduce noise-induced oscillations in practice. Such modifications are common in real-world MPPI applications, where additional smoothing is often necessary due to the stochastic nature of the algorithm. Conversely, this modification to the standard cost function has not been deemed necessary for the NMPC implementation, as the deterministic nature of the algorithm inherently provides sufficient control smoothness without requiring additional regularization terms.

The pseudo-code implementation of the MPPI controller is described in Algorithm 1. At each control step, K control sequences are sampled by adding zero-mean Gaussian noise $\epsilon_i \sim \mathcal{N}(0, \Sigma)$ to the nominal control sequence from the previous iteration (lines 2–3). The covariance matrix Σ determines the exploration amplitude in the control space. Each candidate sequence is constrained (line 4), forward-simulated using the system dynamics

(line 5), and evaluated using the cost function from Equation (6) (line 6). Here, the cost function is denoted as J_i to indicate that it corresponds to the total cost associated with the i-th sampled trajectory. Despite the change in notation, it is computed in the same manner as the optimal cost J^* defined in Equation (6), following the general formulation.

The algorithm then computes importance weights using the softmax transformation (lines 7–8), where the minimum cost ρ ensures numerical stability and the temperature parameter λ balances exploration and exploitation. The optimal control sequence is computed as a weighted average $U_{\text{opt}} = \sum_i w_i \cdot U_i$, where the weights w_i sum to unity (line 9), and their equation is detailed in Equation (8). Finally, the first control action u_0 is extracted for system application, while the remaining sequence is shifted forward to provide warm-starting for the next iteration (line 10).

$$w_i = \frac{\exp\left(-\frac{1}{\lambda}(J_i - \rho)\right)}{\sum_{j=1}^{K} \exp\left(-\frac{1}{\lambda}(J_j - \rho)\right)} \tag{8}$$

Algorithm 1: MPPI control update

Data: Current state x_0, reference trajectory x_{ref}, control sequence U
Result: Optimal control u_0, updated sequence U_{new}
1 **for** $i = 1$ **to** K **do**
2 Generate noise sequence $\epsilon_i \sim \mathcal{N}(0, \Sigma)$;
3 Compute perturbed control: $U_i = U + \epsilon_i$;
4 Clip to control bounds: $U_i = \text{clip}(U_i, u_{\min}, u_{\max})$;
5 Forward simulate trajectory: $x_i = \text{simulate}(x_0, U_i)$;
6 Evaluate cost: $J_i = \text{cost}(x_i, U_i, x_{\text{ref}})$;
7 **end**
8 Compute minimum cost: $\rho = \min_j J_j$;
9 Compute weights: $w_i = \frac{\exp(-(J_i - \rho)/\lambda)}{\sum_j \exp(-(J_j - \rho)/\lambda)}$;
10 Update control: $U_{\text{opt}} = \sum_i w_i \cdot U_i$;
11 Extract and shift: $u_0 = U_{\text{opt}}[0]$, $U_{\text{new}} = [U_{\text{opt}}[1:N], U_{\text{opt}}[N-1]]$;
12 **return** u_0, U_{new}

The MPPI controller achieves real-time performance through JAX just-in-time compilation and parallel execution of trajectory samples, with dynamics propagated using fourth-order Runge–Kutta integration.

2.2.4. Savitzky–Golay Filter Implementation in MPPI

A significant limitation of the MPPI controller is the substantial noise in the control input, which can cause oscillations during path tracking despite achieving a low overall RMSE. To address this issue, a Savitzky–Golay filter is applied as a post-processing step to the MPPI controller output within the Python 3.12 implementation. This smoothing approach represents standard practice in contemporary MPPI implementations, where post-processing filters are commonly employed to reduce control input noise and improve tracking performance [24].

Figure 2 illustrates the effectiveness of this filtering approach, demonstrating the notable reduction in control input noise achieved through the post-processing step.

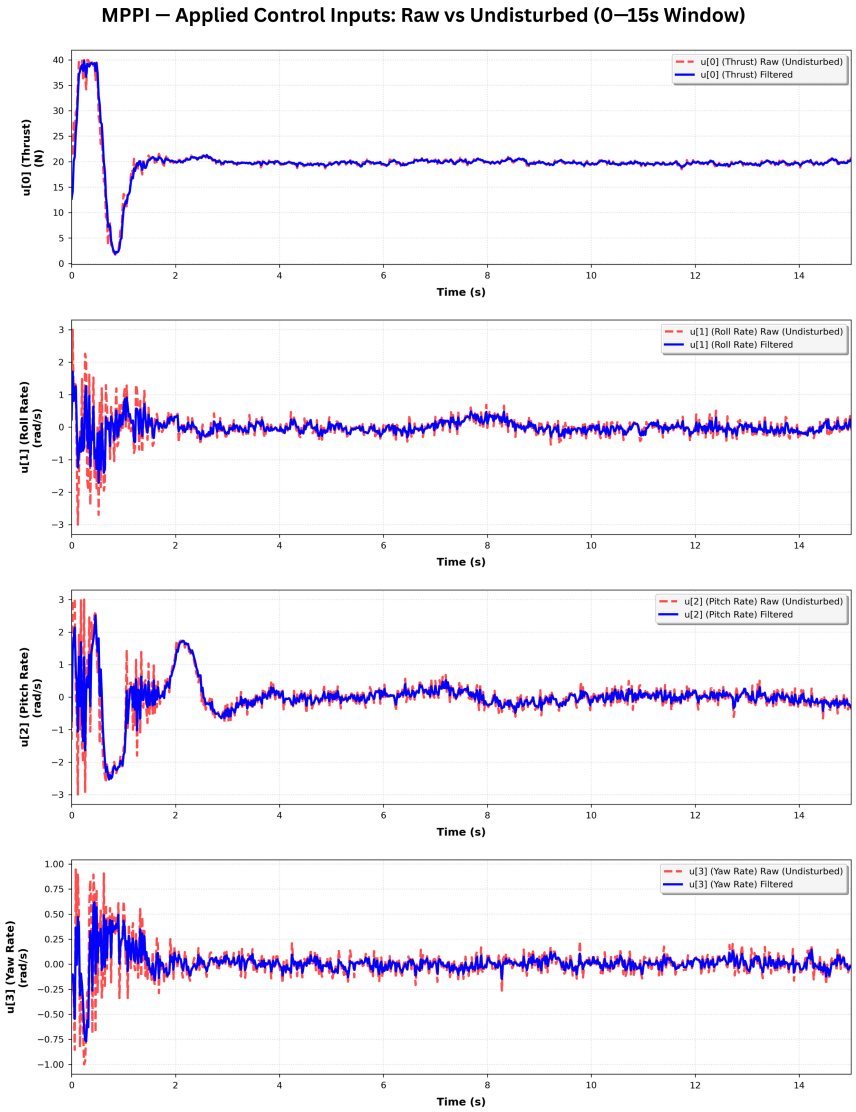


Figure 2. MPPI control inputs over 15 s trajectory execution showing the complete time series response.

Figures 2–5 demonstrate that NMPC generally produces smoother control inputs compared to MPPI. The inherent noise in MPPI control inputs, clearly visible in Figure 2, is attributed to the sampling nature of the algorithm.

2.3. Key Implementation Differences

Having established the theoretical foundations, the practical implementation of NMPC and MPPI reveals fundamental differences in their computational approaches. As pointed out in [17], NMPC employs gradient-based optimization with numerical solvers, while MPPI uses a gradient-free, sampling-based methodology leveraging GPU acceleration. Table 2 summarizes the key distinctions between these two control paradigms.

MPPI's performance is critically dependent on the number of samples K, as will be demonstrated in Section 4.2.3. While a larger sample count theoretically improves the approximation of the optimal control by exploring more candidate solutions, the computational overhead scales linearly with K. On embedded platforms with limited computational resources, unlike workstation-class GPUs, it is not feasible to arbitrarily increase the number of samples to reduce the tracking error while simultaneously meeting real-time control frequency requirements. Conversely, an insufficient number of samples leads to poor approximation quality and suboptimal control performance. This fundamental constraint

necessitates a careful trade-off between solution quality and computational feasibility in real-world deployments of sampling-based control methods.

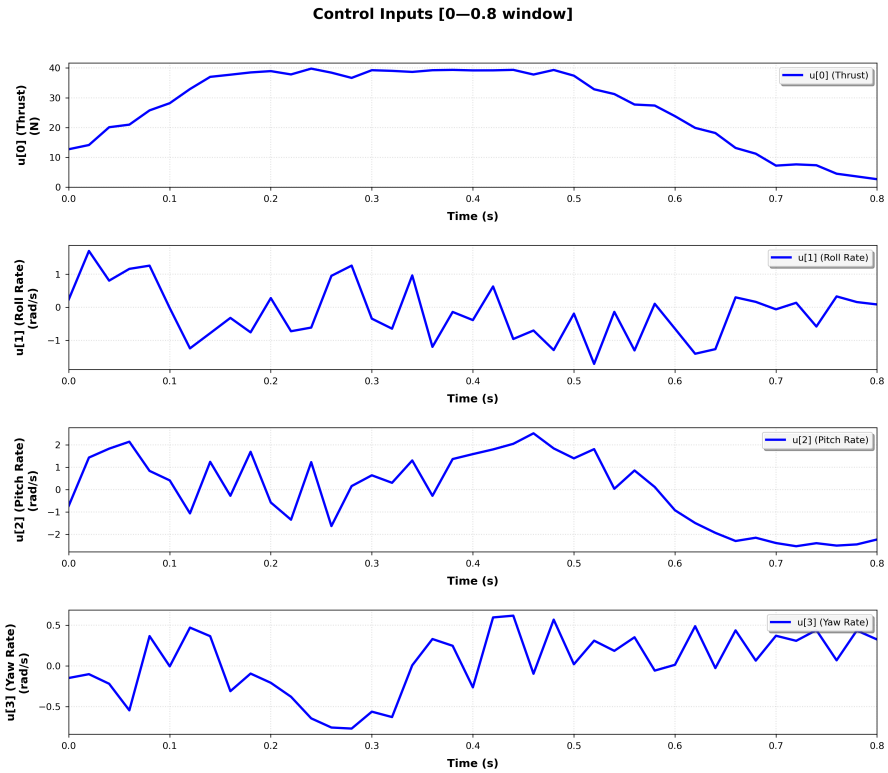


Figure 3. Zoomed view of MPPI control inputs in the time window [0–0.8] s.

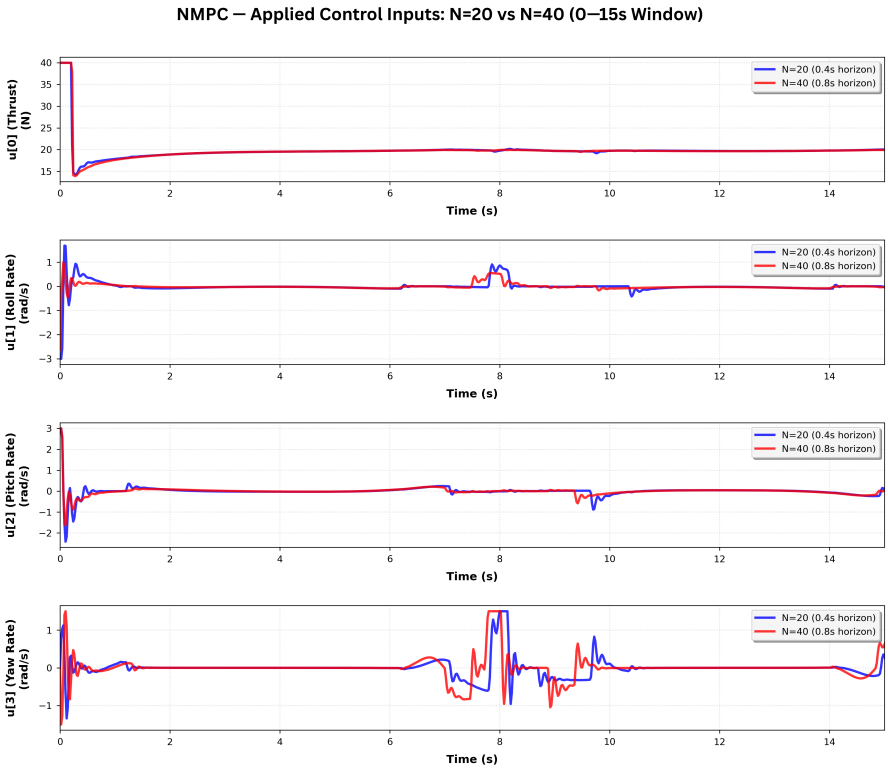


Figure 4. NMPC control inputs over 15 s trajectory execution showing the complete time series response.

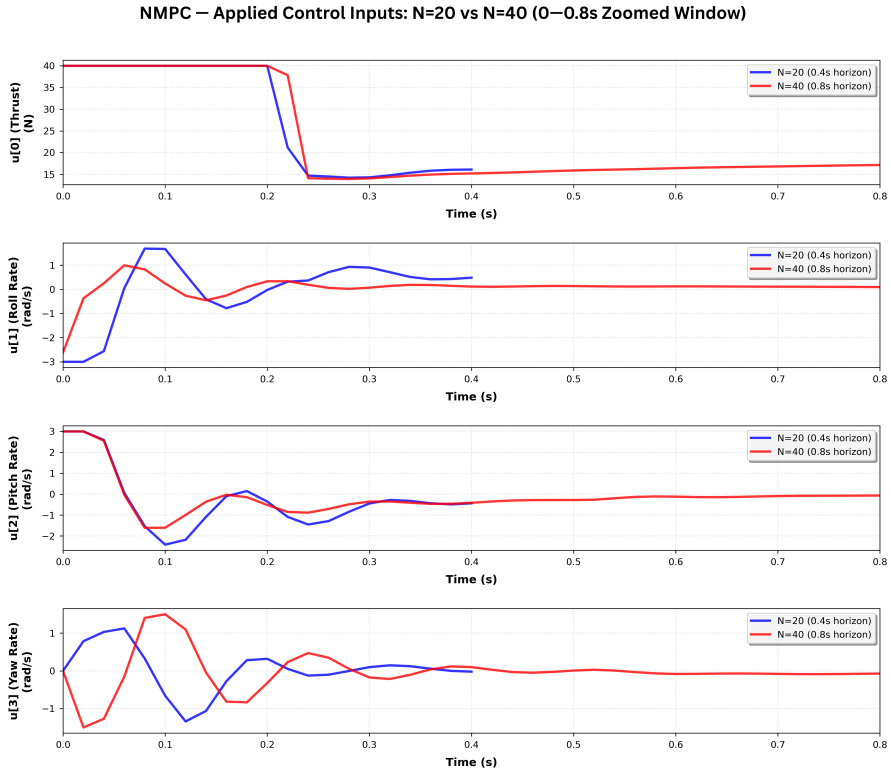


Figure 5. Zoomed view of NMPC control inputs in the time window [0–0.4] s for the NMPC with a horizon of 20 steps and [0–0.8] s for the NMPC with a horizon of 40 steps.

Table 2. Comparison of NMPC and MPPI implementation characteristics.

Aspect	NMPC	MPPI
Optimization	Deterministic optimal control via gradient-based methods	Stochastic optimal control through Monte Carlo sampling
Constraints	Explicit handling of input/ state constraints within solver framework	Input constraints via sample clipping; state constraints through cost penalties
Control Output	Smooth, locally optimal control sequence	Weighted average of noisy samples, inherently stochastic
Computation	CPU-intensive with specialized solvers (`acados`)	GPU-accelerated parallel computation (`JAX`)
Post-Processing	Direct implementation possible	Usually filtered (e.g., Savitzky–Golay) for output smoothing

These distinctions highlight a fundamental trade-off: NMPC provides structured, smooth control through mature optimization techniques. At the same time, MPPI offers computational flexibility and parallelization capabilities at the cost of output noise that requires additional processing for practical deployment.

A critical advantage of MPPI over the SQP-RTI NMPC implementation used in this work lies in its treatment of the optimization problem. Unlike gradient-based methods that rely on successive linearization of constraints and dynamics at each iteration, MPPI's sampling-based approach directly evaluates the nonlinear cost function without requiring linearization approximations. This allows MPPI to better exploit the inherent nonlinear UAV dynamics and handle non-convex cost landscapes, contributing to its superior trajectory tracking performance as evidenced by the 18.6% RMSE improvement presented in this work (this result will be discussed further in Section 4.1.1).

3. Methodology

This section presents the experimental framework used to evaluate and compare the NMPC and MPPI control strategies. The methodology covers hardware platform selection, Processor-in-the-Loop testbed architecture, controller implementation details, and performance evaluation criteria.

3.1. UAV Platform

The control algorithms are implemented on a commercial UAV, the RX2 manufactured by Mavtech Srl. While this platform shares similar characteristics with the Holybro X500 quadrotor used in the simulation, certain parameters such as mass and inertial properties differ between the two systems. However, from a mathematical standpoint, the two platforms are effectively equivalent, as the differences are purely parametric in nature. Furthermore, both systems share the same architectural design, featuring a dual computational architecture. Given the constraints on modifying the UAV's mechanical design, this work focuses exclusively on the software design of the control systems, presenting the complete design process and implementation.

Similar to many research-grade quadrotor platforms, it adopts a dual-computer architecture [14,25]. The architecture and its schematic representation are depicted in Figures 6 and 7, respectively. The Jetson Orin Nano, which serves as the target platform for implementing the algorithms described in this article, is explained in detail in Section 3.2.

Figure 6. Testbench for Processor-in-the-Loop testing on the UAV's onboard computer.

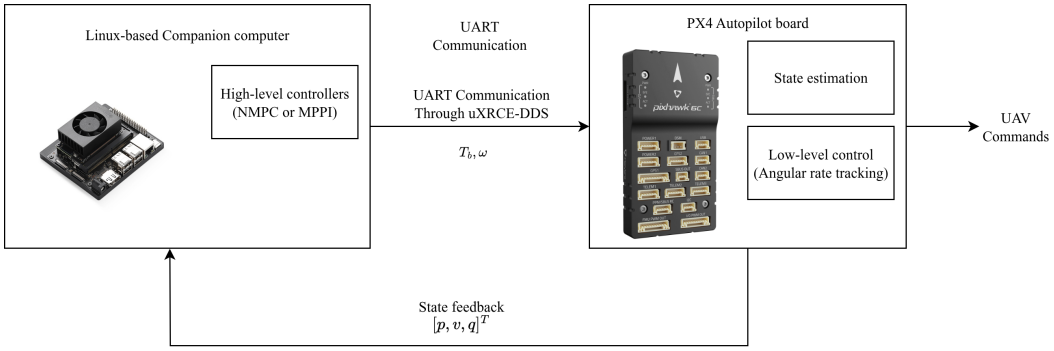


Figure 7. Schematic describing the two-computer architecture mounted on board of the UAV.

3.2. Hardware Platform: NVIDIA Jetson Orin Nano

The NVIDIA Jetson Orin Nano represents a significant advancement in embedded computing for robotics applications. The platform features an ARM Cortex-A78AE CPU and integrated Ampere GPU architecture with 1024 CUDA cores, delivering up to 40 TOPS of AI performance while maintaining power efficiency suitable for mobile platforms [26].

This system-on-module has become increasingly popular in UAV applications due to its ability to perform real-time computer vision, autonomous navigation, and control system computations onboard [1,11,27]. Thanks to its combination of GPU acceleration capabilities and relatively low power consumption (typically 5–15 W), the platform is

particularly well-suited to weight- and power-constrained UAV systems, where traditional desktop-class processors would be impractical.

Furthermore, the native CUDA support enables efficient parallel processing of computationally intensive algorithms such as MPPI controllers, which are essential for advanced UAV control. Previous work has demonstrated successful implementation of real-time MPPI controllers on this platform, validating its usage in the robotics community [11].

3.3. Processor-in-the-Loop Experimental Framework

The Processor-in-the-Loop (PITL) testbed uses an NVIDIA Jetson Orin Nano companion computer. Control algorithms developed on an external workstation using Software-in-the-Loop (SITL) are deployed and tested here.

The testbed leverages Docker containerization for seamless algorithm deployment. This approach eliminates code modifications between development and target hardware while preventing dependency conflicts. Due to architectural differences between development workstations (x86) and the Jetson platform (ARM), all packages must support both architectures. This setup validates the controller computational performance under actual UAV hardware constraints. It provides a realistic assessment of algorithm feasibility on resource-limited platforms. The PITL architecture bridges the gap between simulation-based development and real-world deployment. Controllers are tested on the same resource-constrained hardware used in actual flight operations.

Prediction Horizon Selection

A critical design consideration for both controllers is the selection of appropriate prediction horizons that balance tracking performance with computational efficiency. While both controllers operate at the same control frequency (50 Hz, corresponding to 0.02 s timesteps), they employ different horizon lengths optimized for their respective algorithmic characteristics. The horizon selection process was performed through a systematic performance analysis on workstation hardware to identify saturation points, beyond which additional prediction steps yield diminishing returns. Figure 8 demonstrates that NMPC achieves performance saturation at approximately 20 horizon steps, corresponding to a prediction time of $T_f = 0.4$ s. Beyond this point, increasing the horizon length provides no significant improvement in tracking accuracy while unnecessarily increasing computational overhead.

In contrast, MPPI requires a longer prediction horizon of 40 steps ($T_f = 0.8$ s) to achieve comparable performance saturation, as demonstrated in the parameter sensitivity analysis (Section 4.2.3). This difference stems from the fundamental algorithmic distinctions between the two approaches: NMPC's deterministic optimization efficiently leverages shorter horizons through gradient-based convergence, while MPPI's stochastic sampling benefits from extended prediction windows to adequately explore the control space and achieve robust importance sampling convergence. Therefore, the selected horizons (20 steps for NMPC and 40 steps for MPPI) represent optimal configurations that maximize tracking performance while minimizing the computational burden for each respective algorithm. This approach ensures fair comparison by allowing each controller to operate at its performance-optimized configuration, rather than imposing artificial uniformity that would disadvantage one or both methods. Nevertheless, an NMPC configuration with 40 prediction steps was also evaluated to ensure a more equitable comparison between the control algorithms.

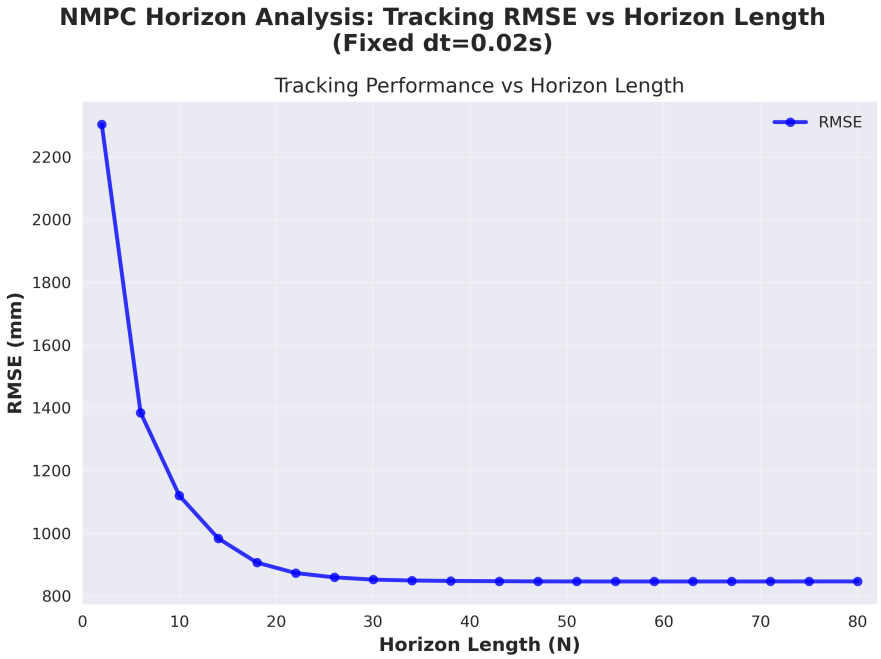


Figure 8. NMPC performance analysis showing RMSE saturation starting from 20 horizon steps, justifying the selected prediction horizon length.

3.4. Controller Implementation and Configuration

Both controllers are implemented within a modular ROS 2 framework, enabling launch-time controller selection through ROS 2 launch file parameters. This architecture ensures fair comparative evaluation while maintaining practical deployment considerations.

The following subsections detail the specific parameter configurations and implementation choices for each controller.

3.4.1. NMPC Implementation

The NMPC controller utilizes the CasADi/Acados framework for efficient nonlinear optimization. Table 3 presents the complete parameter configuration used during experimental validation.

Table 3. NMPC controller parameters and cost weights.

Parameter	Value	Unit
Prediction Horizon		
Control timestep (dt)	0.02	s
Prediction horizon (T_f)	0.4 (and 0.8)	s
Horizon steps (N)	20 (and 40)	–
Control Constraints		
Thrust limits	[0.0, 40.0]	N
Roll/pitch rate limits	$[-3.0, 3.0]$	rad s^{-1}
Yaw rate limits	$[-1.5, 1.5]$	rad s^{-1}
Solver Configuration [NMPC Specific]		
QP solver	PARTIAL_CONDENSING_HPIPM	
Hessian approximation	GAUSS_NEWTON	
Integrator type	ERK	
NLP solver	SQP_RTI	

Table 3. *Cont.*

Cost Weight	Stage (Q)	Initial (Q_0)	Terminal (Q_e)	Control (R)
Position	1×10^4	1×10^3	1	-
Velocity	1×10^3	1×10^2	1×10^{-1}	-
Quaternion	1×10^3	1×10^2	1×10^{-1}	-
Angular rate	1×10^1	1	1×10^{-3}	-
Thrust	-	-	-	2×10^{-2}
Roll/pitch rate	-	-	-	2×10^{-1}
Yaw rate	-	-	-	2×10^{-1}

The NMPC formulation employs a hierarchical cost structure with different weights for initial, intermediate, and terminal stages to ensure smooth convergence and tracking performance. The Real-Time Iteration (RTI) scheme is utilized to maintain computational efficiency suitable for embedded applications [20].

3.4.2. MPPI Implementation

The MPPI controller leverages JAX for GPU-accelerated parallel computation with just-in-time compilation. Table 4 details the complete parameter configuration.

Table 4. MPPI controller parameters.

Parameter	**Value**	**Unit**
Prediction Horizon		
Control timestep (dt)	0.02	s
Prediction horizon (T_f)	0.8	s
Horizon steps	40	–
Sampling [MPPI Specific]		
Number of samples (K)	900	–
Temperature (λ)	10^3	–
Control Constraints		
Thrust limits	[0.0, 40.0]	N
Roll/pitch rate limits	$[-3.0, 3.0]$	rad s^{-1}
Yaw rate limits	$[-1.5, 1.5]$	rad s^{-1}

Cost Weight	State (Q)	Control (R)	Control Rate (R_Δ)
Position	5×10^3	-	-
Velocity	4×10^1	-	-
Quaternion	2×10^1	-	-
Angular rate	2×10^1	-	-
Thrust	-	2×10^{-2}	1×10^{-3}
Roll/pitch rate	-	2×10^{-1}	1×10^{-2}
Yaw rate	-	2×10^{-1}	1×10^{-2}

The controller employs stochastic sampling with 900 trajectories to explore the control space effectively. As detailed in Equation (8), the temperature parameter (λ) controls the selectivity of the importance sampling, with the chosen value providing a balance between exploration and exploitation. The control rate penalties help achieve smooth actuation, while the noise scaling parameters determine the exploration magnitude for each control channel [9].

Both controllers are configured with identical physical constraints to ensure fair performance comparison and unbiased evaluation of their respective control strategies. The

parameter tuning for both controllers was performed in the SITL simulation to expedite the deployment process and reduce hardware testing time.

3.5. Performance Evaluation Criteria

The comparative evaluation encompasses both trajectory tracking accuracy and computational performance metrics. Both MPPI and NMPC are evaluated against the timing requirements established by the PX4 Autopilot [23], which specifies that outer-loop controllers should operate at 50 Hz (corresponding to a 20 ms maximum computation time, as indicated in both Tables 3 and 4) [28]. This real-time constraint serves as the primary computational feasibility threshold for embedded deployment.

3.5.1. Tracking Performance Metrics

Trajectory tracking performance is quantified using Root Mean Square Error (RMSE), computed as follows:

$$RMSE_{total} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} ||p(t_i) - p_d(t_i)||^2} \tag{9}$$

where $p(t_i)$ represents the actual position and $p_d(t_i)$ the desired position at time t_i. Component-wise RMSE values are calculated separately for each spatial dimension to provide detailed performance analysis.

3.5.2. Computational Performance Metrics

Computational performance evaluation includes the following:

- Computation Time Statistics: Mean, median, and 95th percentile execution times;
- Real-time Feasibility: Which hardware implementation meets the 20 ms constraint.

3.5.3. Test Scenarios

The experimental validation employs multiple test scenarios to comprehensively evaluate controller performance:

1. Nominal Trajectory Tracking: Three-dimensional figure-8 (lemniscate) reference trajectory under ideal conditions;
2. Robustness Evaluation: Performance under realistic sensor noise conditions;
3. Dynamic Capability: Dynamic reference trajectory switching scenarios;
4. Parameter Sensitivity: Systematic analysis of key parameter effects on performance and computational requirements.

These test scenarios offer a thorough examination of typical UAV operational conditions, allowing for the systematic evaluation of the trade-offs between tracking performance and computational efficiency.

4. Results and Analysis

This section presents a comprehensive evaluation of the NMPC and MPPI control strategies, encompassing trajectory tracking performance under various conditions and detailed computational analysis. The results provide critical insights into the practical trade-offs between control accuracy and computational requirements for MPPI deployment on embedded UAV hardware.

4.1. Trajectory Tracking Performance

The trajectory tracking performance evaluation encompasses nominal operating conditions, realistic sensor noise scenarios, and dynamic reference changes to comprehensively assess controller capabilities under typical UAV operational scenarios.

4.1.1. Nominal Conditions

To benchmark the closed-loop tracking performance under ideal conditions, both controllers are commanded to follow a three-dimensional lemniscate of Bernoulli (figure-8 trajectory), as depicted in Figure 9. The parametric representation is

$$x(t) = x_c + s\,\frac{\cos\theta}{1+\sin^2\theta},$$
$$y(t) = y_c + s\,\frac{\sin\theta\,\cos\theta}{1+\sin^2\theta}, \qquad \theta = \frac{2\pi t}{T}.$$
$$z(t) = z_c,$$

The numerical parameters are identical for both controllers:

- Centre: $(x_c, y_c, z_c) = (0.0, 0.0, 2.5)$ m;
- Scale: $s = 5.0$ m;
- Period: $T = 15.0$ s (one complete figure-8 every 15 s);
- Total simulation time: $T_{sim} = 30$ s;
- Simulation step: $\Delta t = 0.02$ s.

The reference attitude is maintained level ($\phi = \theta = \psi = 0$), encoded as unit quaternion $q = [1, 0, 0, 0]^\mathsf{T}$, with zero angular rate demands. Velocity references are derived analytically from the trajectory definition. For each controller, tracking error is evaluated using the RMSE metric defined in Equation (9).

The initial state is the same for both controllers in order to ensure a fair evaluation, with the UAV starting from position $p_0 = [2.0, 2.0, 0.0]^T$ m, with zero initial velocity, level attitude, and zero angular rates.

The comparative analysis in Table 5 highlights notable performance differences between the control methodologies. Within the NMPC framework, extending the prediction horizon from $N = 20$ to $N = 40$ yields modest gains, as previously illustrated in Figure 8, with an overall RMSE reduction of 4.4% (from 0.8866 m to 0.8480 m). As expected from MPC theory, longer prediction horizons generally improve RMSE; however, in this case, the improvement is limited and comes at the expense of increased computational time, (as detailed in the computational analysis in Section 4.2).

Despite these improvements, MPPI demonstrates substantially superior tracking capabilities across all spatial dimensions. Compared to the optimized NMPC configuration (N = 40), MPPI achieves an 18.6 % reduction in overall RMSE (0.8480 m to 0.6897 m).

Table 5. RMSE performance comparison: NMPC vs. MPPI.

Component	NMPC (N = 20) (m)	NMPC (N = 40) (m)	MPPI (m)
X	0.7682	0.7272 (−5.3%)	0.6126
Y	0.1760	0.1742 (−1.0%)	0.0901
Z	0.4063	0.4000 (−1.6%)	0.3039
Overall	0.8866	0.8480 (−4.4%)	0.6897

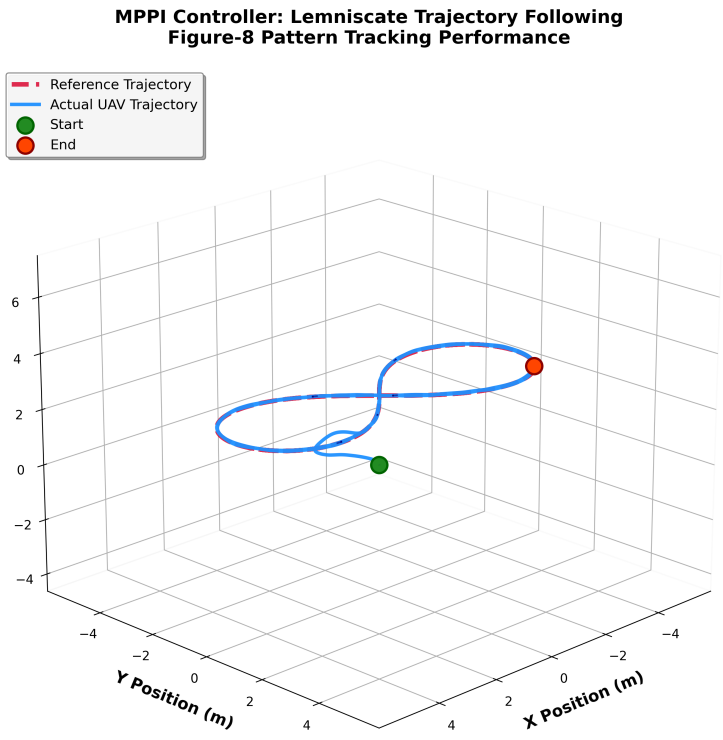


Figure 9. Three-dimensional visualization of lemniscate (figure-8) reference trajectory and corresponding MPPI tracking performance under nominal simulation conditions.

Given this comprehensive performance analysis, MPPI emerges as the optimal control architecture for this investigation. The algorithm's recent development provides an opportunity to evaluate its efficacy under realistic operational conditions, including process noise and dynamic reference variations. Consequently, MPPI has been selected as the primary control framework for subsequent trajectory tracking assessments, leveraging its demonstrated performance advantages while maintaining comparison against the established NMPC methodology.

4.1.2. Noisy Conditions

To validate controller robustness, performance is evaluated under sensor noise conditions that simulate realistic measurement uncertainties. The sensor noise parameters are detailed in Table 6, employing a comprehensive stochastic model incorporating both measurement noise and sensor bias drift, inspired by sensors usually mounted on board a quadrotor. The sensor noise implementation includes the following.

Position Noise: Noisy position measurements are computed as follows:

$$p_{\text{meas}}(t) = p_{\text{true}}(t) + n_p(t) + b_p(t) \tag{10}$$

where $n_p(t) \sim \mathcal{N}(0, \sigma_p^2 I)$ represents white Gaussian noise and $b_p(t)$ is accumulated position bias.

The value $\sigma_p = 0.1\,\text{m}$ reported in Table 6 represents the standard deviation of a continuous Gaussian noise process that corrupts all position measurements at each timestep throughout the simulation, rather than a single perturbation applied to the initial position. As such, it cannot be directly interpreted as a fixed percentage error relative to the starting position. This noise modeling approach applies consistently to all sensor measurements, which are corrupted according to the noise characteristics specified in Table 6.

Velocity Noise: Similarly, velocity measurements are corrupted by

$$v_{\text{meas}}(t) = v_{\text{true}}(t) + n_v(t) + b_v(t) \tag{11}$$

Attitude Noise: This is applied in Euler angle space to maintain valid quaternion representations:

$$[\phi, \theta, \psi]_{\text{noisy}} = [\phi, \theta, \psi]_{\text{true}} + \boldsymbol{\eta}_{\text{att}}(t) + b_{\text{att}}(t) \tag{12}$$

$$q_{\text{meas}}(t) = \text{Euler2Quat}([\phi, \theta, \psi]_{\text{noisy}}) \tag{13}$$

Bias Drift Model: Sensor biases evolve according to random walk processes:

$$b_p(t + \Delta t) = b_p(t) + w_p(t)\Delta t \tag{14}$$

where $w_p(t) \sim \mathcal{N}(0, \sigma^2_{\text{drift},p} I)$ represents the process noise driving the bias drift.

Table 6. Sensor noise configuration for UAV simulation.

Parameter	Value	Units	Description
Position noise (GPS)	0.1	m	Standard deviation of position measurements
Velocity noise	0.05	m/s	Standard deviation of velocity estimation
Attitude noise (IMU)	1.0	deg	Standard deviation of attitude measurements
Angular rate noise	2.0	deg/s	Gyroscope measurement noise
Measurement frequency	100.0	Hz	Sensor update rate
Position bias drift	0.001	m/s	Position bias drift rate
Attitude bias drift	0.1	deg/s	Attitude bias drift rate

Conversely to the previous section, steady-state performance is evaluated following the transient phase, with system convergence to the reference trajectory occurring at approximately 2.5 s. Tables 7 and 8 present position RMSE results, focusing exclusively on steady-state errors.

Moreover, the robustness evaluation involved tracking the identical reference trajectory used in the previous nominal test case (Section 4.1.1), employing both MPPI and NMPC controllers (in both horizon configurations). The key modification was the introduction of realistic sensor noise to assess controller performance under more challenging conditions.

Table 7. MPPI post-transient RMSE with percentage degradation.

Component	Undisturbed (mm)	Noisy (mm) (degr. %)
X-axis	28.92	82.72 (+186.0%)
Y-axis	19.60	65.98 (+236.6%)
Z-axis	83.09	110.93 (+33.5%)
Total	90.13	153.30 (+70.1%)

Table 8. NMPC post-transient RMSE comparison for horizon lengths N = 20 and N = 40.

	N = 20		N = 40	
Component	Clean (mm)	Noisy (mm) (degr. %)	Clean (mm)	Noisy (mm) (degr. %)
X-axis	577.74	632.13 (+9.4%)	317.37	349.95 (+10.3%)
Y-axis	189.33	296.79 (+56.8%)	117.74	136.51 (+15.9%)
Z-axis	84.42	113.11 (+34.0%)	53.92	80.85 (+49.9%)
Total	613.80	707.43 (+15.3%)	342.77	384.23 (+12.1%)

The results presented in Tables 7 and 8 suggest that MPPI offers more precise trajectory tracking than NMPC under the tested ideal and noisy conditions. Conversely, the NMPC exhibits greater relative robustness, as its performance degrades less significantly when noise is introduced. This indicates a predictable response to disturbances, though it comes at the cost of a higher baseline tracking error. Numerically, the MPPI controller's absolute error of 153.30 mm in the noisy scenario is considerably lower than the NMPC's best result of 384.23 mm. This suggests that for applications where minimizing the absolute tracking error is the primary objective, MPPI presents a favorable performance profile.

4.1.3. Dynamic Reference Changes

To evaluate adaptability to time-varying mission requirements, the controllers' performance is tested with dynamically changing reference trajectories. The UAV initially tracks a figure-8 trajectory before switching to a circular path at a different spatial location at $t = 20$ s, as shown in Figure 10.

The trajectory switch deliberately introduces a challenging scenario requiring rapid adaptation to simultaneous changes in path geometry, spatial location, and altitude. The circular path center is positioned to require UAV direction reversal, testing the controller's predictive capabilities under discontinuous reference changes.

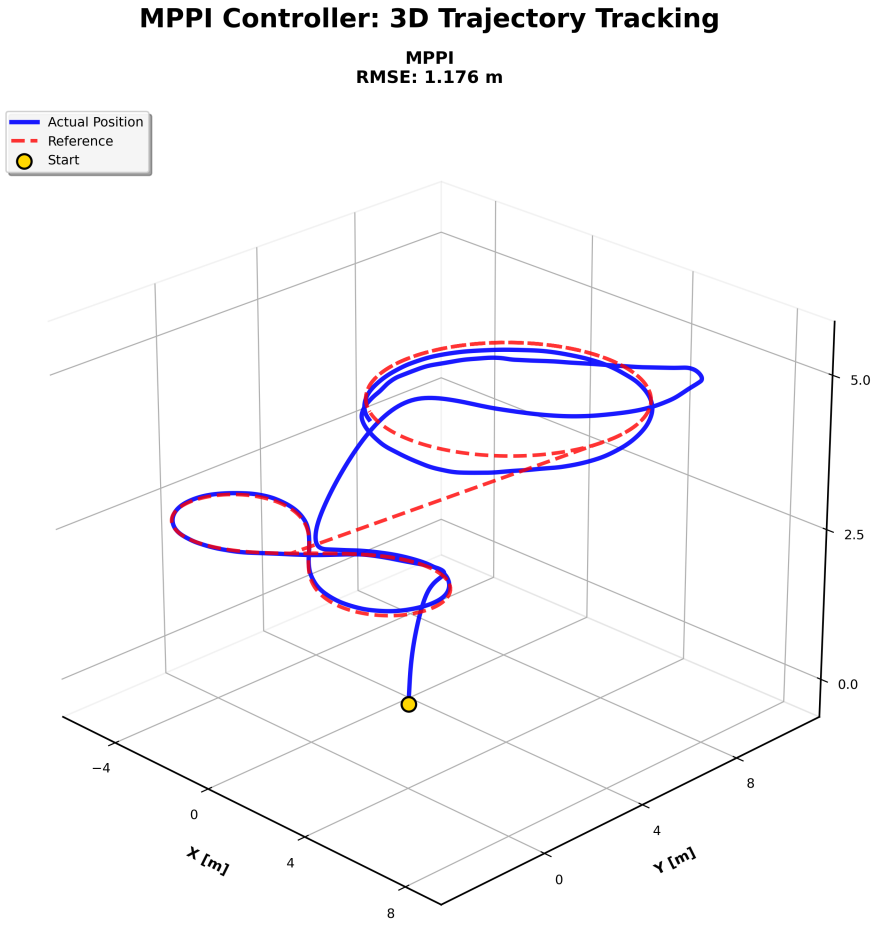


Figure 10. UAV trajectory switching from figure-8 to circular path at $t = 20$ s. The blue line shows the actual position of the quadrotor, while the dashed red line shows the reference position. For conciseness, only the MPPI controller results are shown here, but Figure 11 presents a comparison with the NMPC controller as well.

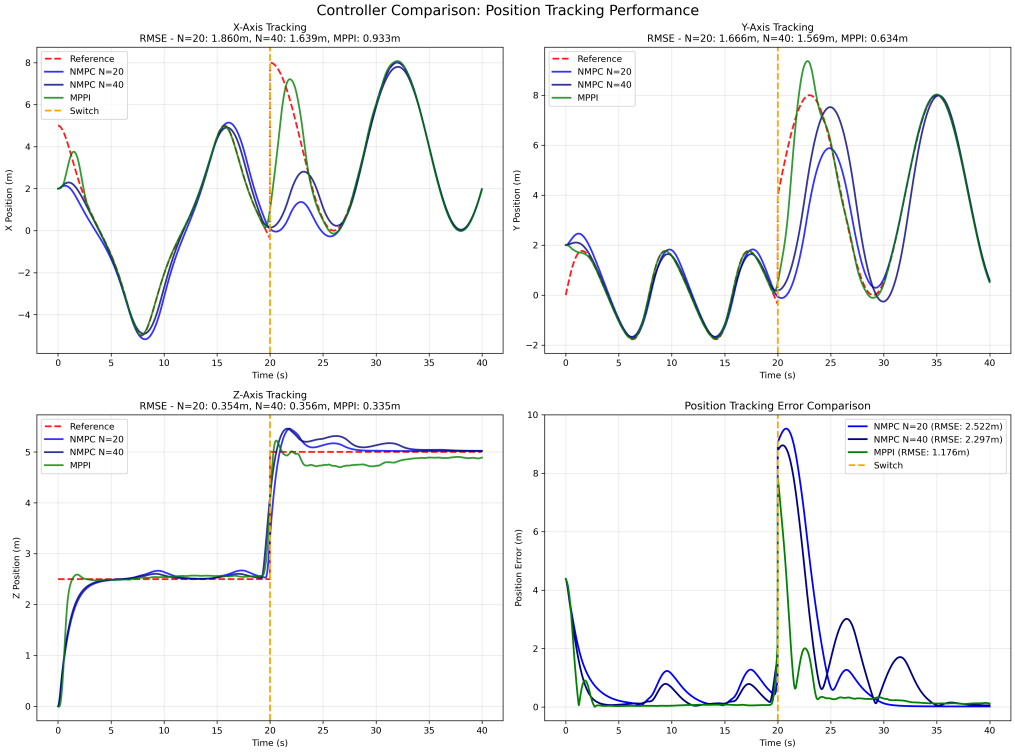


Figure 11. Time series data during trajectory switching at $t = 20$ s. Blue line shows actual quadrotor position; dashed red line shows reference position. The yellow dotted line indicates when the reference switches.

The quantitative analysis in Figure 11 reveals distinct performance characteristics across the three control strategies. During the initial figure-8 phase (0–20 s), all controllers demonstrate stable tracking performance, with MPPI achieving the lowest steady-state error. At the trajectory switch instant (marked by the vertical dashed line), position errors exhibit controller-dependent responses: NMPC with N = 20 peaks at approximately 9.5 m, NMPC with N = 40 reaches about 8.0 m, while MPPI maintains the most controlled transition, with a peak error of approximately 7.5 m.

The convergence characteristics post-switch reveal important distinctions in controller behavior. MPPI demonstrates the fastest recovery, returning to steady-state levels (below 0.25 m) within 3 s. NMPC with N = 40 shows intermediate performance, while NMPC with N = 20 exhibits the slowest convergence. The overall RMSE values confirm these observations: MPPI achieves 1.176 m, NMPC N = 40 records 2.297 m, and NMPC N = 20 reaches 2.522 m.

Examining the individual axis responses, MPPI maintains superior tracking fidelity across all three dimensions, particularly evident in the Z-axis where altitude transitions from 2.5 m to 5.0 m are executed smoothly. These results demonstrate that while NMPC benefits moderately from extended prediction horizons, MPPI's sampling-based optimization approach provides inherently better transient response and faster adaptation to reference changes, making it particularly suitable for dynamic trajectory tracking missions requiring frequent maneuver transitions.

4.2. Computational Performance Analysis

This section provides a detailed analysis of computational requirements for both control strategies, encompassing single-platform performance characteristics and parameter sensitivity effects on computational overhead.

The computational frameworks differ fundamentally: NMPC (CasADi/Acados, Table 9) uses pre-compiled routines with minimal runtime overhead, while MPPI (JAX, Tables 10 and 11) relies on JIT compilation occurring once at initialization. In practical UAV deployment, this JIT overhead is absorbed during ground-based system initialization and does not impact flight performance. Median computation times are reported to provide robust performance metrics that are less sensitive to occasional computational outliers than mean values.

4.2.1. NMPC Performance

The NMPC implementation demonstrates consistent computational characteristics across hardware platforms, being independent of CPU/GPU computational paradigms. This hardware-agnostic behavior significantly simplifies deployment on embedded systems while benefiting from mature optimization frameworks.

Table 9. Computational time statistics for NMPC implementation.

Platform	Mean (ms)	Median (ms)	95th %ile (ms)
Workstation PC (N = 20)	0.76	0.74	0.96
Workstation PC (N = 40)	1.56	1.48	1.94
Embedded Hardware (N = 20)	5.76	5.60	6.69
Embedded Hardware (N = 40)	10.98	10.38	13.14

Table 9 presents computational time statistics across both hardware platforms. The NMPC approach exhibits substantially faster computation times compared to the MPPI implementation, with excellent predictability demonstrated by low standard deviation values. The CasADi/Acados framework provides mature optimization capabilities suitable for embedded deployment.

4.2.2. MPPI Performance (CPU vs. GPU)

MPPI computational performance is critically dependent on hardware acceleration capabilities. The following analysis compares CPU and GPU implementations across workstation and embedded platforms to quantify the impact of parallel processing on real-time feasibility.

Workstation Performance (CPU: Intel Ultra 7 165H/GPU: Nvidia RTX 2000)

Table 10. MPPI algorithm performance comparison—workstation PC.

Metric	CPU	GPU
Mean (ms)	14.97	3.60
Median (ms)	14.00	2.53
95th Percentile (ms)	19.98	2.88
Meets 20 ms Target	Yes	Yes
Speedup (Median)	—	5.53×

Embedded Hardware Performance (CPU: 6-core Arm Cortex-A78AE/GPU: NVIDIA Ampere 1024 CUDA cores)

Table 11. MPPI algorithm performance comparison—Jetson Orin Nano.

Metric	CPU	GPU
Mean (ms)	31.02	26.40
Median (ms)	29.15	17.63
95th Percentile (ms)	30.71	18.35
Meets 20 ms Target	No	Yes
Speedup (Median)	—	1.65×

The results demonstrate that GPU acceleration is essential for MPPI real-time performance on embedded hardware. While workstation hardware provides sufficient computational resources for both CPU and GPU implementations, the embedded platform requires GPU acceleration to meet the 20 ms real-time constraint.

JIT Compilation Overhead

Initial computation times include JAX just-in-time compilation overhead, with GPU compilation requiring additional CUDA kernel generation. However, as indicated in Tables 10 and 11, subsequent iterations demonstrate substantial performance improvements, making the initial compilation cost acceptable for practical deployment.

The computational statistics in Tables 10 and 11 exclude the initial JIT compilation times from Table 12, as these represent one-time initialization overhead. Speedup calculations use median values for robust comparison against occasional computational outliers.

Table 12. MPPI first computation time (including JIT compilation)—workstation.

Implementation	First Computation Time (ms)
CPU	704.89
GPU	1800.29

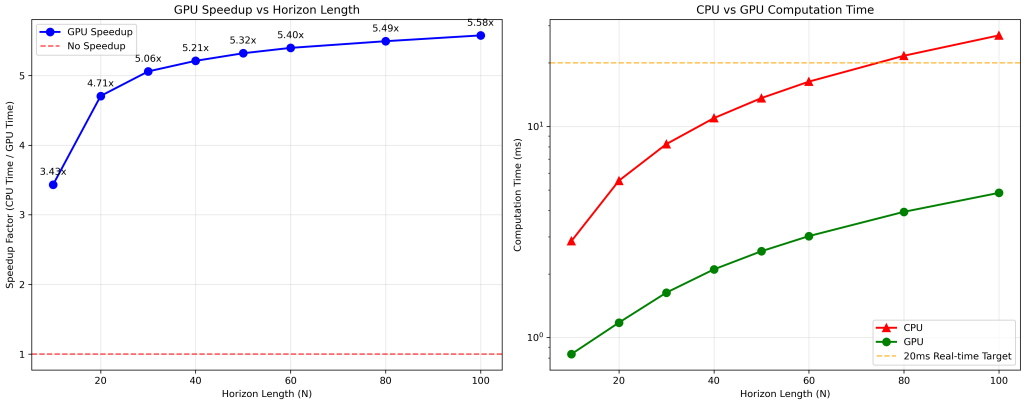
Discussion on GPU Speedup

The performance improvement gained from GPU acceleration is highly variable and not a fixed value. The actual speedup depends on numerous factors, including the underlying hardware architecture, the specific CPU and GPU models, and the nature of the computational task. For our MPPI implementation, this speedup fluctuates significantly based on algorithmic parameters. For instance, Figure 12a illustrates how the speedup changes with the prediction horizon, while Figure 12b shows its dependency on the number of samples. As demonstrated in Tables 10 and 11, the speedup varies considerably between hardware architectures, with workstation systems achieving a 5.53× improvement compared to 1.65× on embedded platforms. While performance gains may be more modest on embedded systems compared to high-end workstations, GPU implementation often remains crucial for achieving the real-time performance demanded by complex control algorithms on resource-constrained hardware.

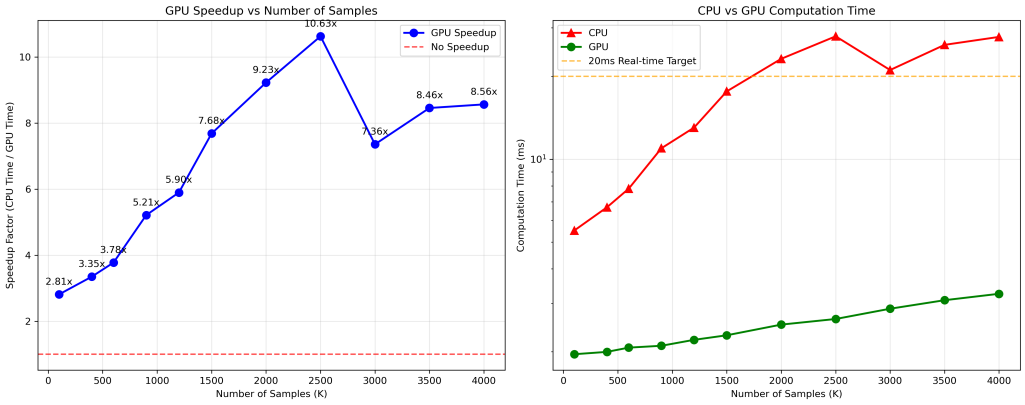
4.2.3. MPPI Parameter Sensitivity Analysis

This work presents a systematic evaluation of key MPPI parameters for real-time UAV trajectory tracking applications. The analysis examines the critical trade-offs between sample batch size and prediction horizon length in order to identify the optimal configuration that balances tracking accuracy with computational constraints. All experimental validation is conducted on embedded UAV hardware using the Jetson Orin Nano platform.

Note: RMSE values in this analysis are elevated compared to Section 4.1.1 due to shorter simulation durations used for computational efficiency during parameter sweeps, which include proportionally more transient behavior.



(**a**)



(**b**)

Figure 12. Performance comparison between GPU and CPU implementations of MPPI across different parameter configurations. (**a**) GPU speedup compared to CPU implementation of MPPI at varying values of N horizon steps. (**b**) GPU speedup compared to CPU implementation of MPPI at varying values of K samples.

Horizon Length Analysis: Figure 13 illustrates the relationship between prediction horizon length and both tracking performance and computational cost. The results demonstrate sharp RMSE improvement from 20 to 30 timesteps, followed by a performance plateau between 30 and 40 steps. This indicates diminishing returns for longer prediction windows.

Beyond horizon 50, slight performance degradation occurs, suggesting that overly long horizons introduce prediction uncertainties. Considering the 20 ms computational constraint shown in the figure, a horizon length of 40 steps emerges as optimal. This configuration achieves competitive accuracy (RMSE $\approx$ 0.92 m) while maintaining computation time around 17 ms.

Sample Size Analysis: Figure 14 reveals non-monotonic performance characteristics with respect to sample count. The RMSE initially degrades around 400 samples due to insufficient exploration. Performance then progressively improves toward 1650 samples through better importance sampling convergence. However, the 20 ms constraint shown in the figure limits the practical sample size to approximately 800–1250 samples.

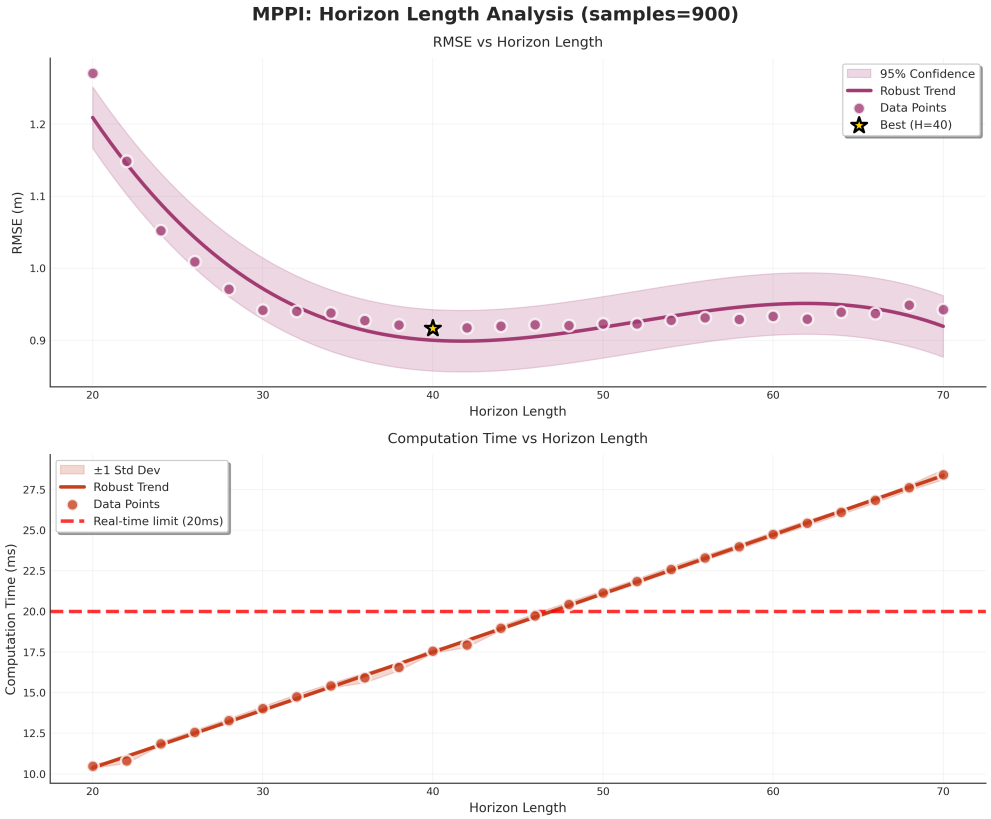


Figure 13. MPPI horizon length analysis showing RMSE and computation time versus prediction horizon.

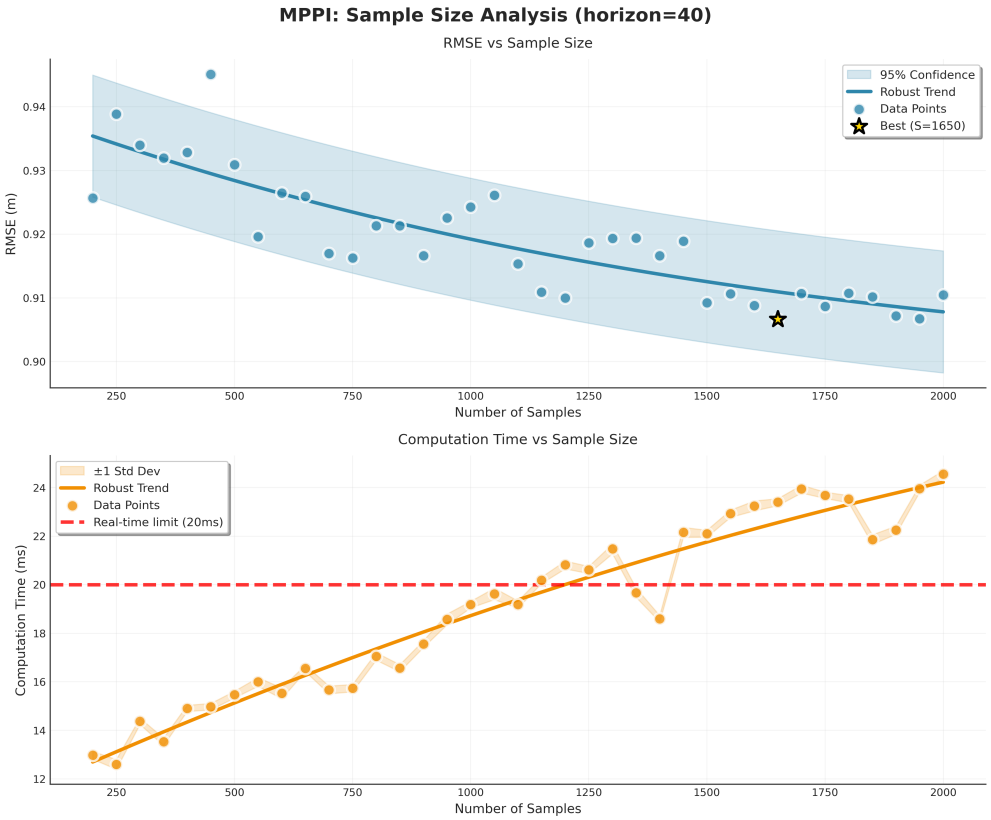


Figure 14. MPPI sample size analysis showing RMSE and computation time versus number of samples.

Temperature Parameter Effects: The temperature parameter λ controls exploration–exploitation trade-offs through the weighting function (detailed implementation in Section 2.2.3):

$$w_i = \frac{\exp(-(J_i - \rho)/\lambda)}{\sum_{j=1}^{K} \exp(-(J_j - \rho)/\lambda)} \tag{15}$$

As demonstrated in Figure 15, lower temperature values improve RMSE performance by focusing on optimal trajectories but increase control input noise. Higher values maintain exploration diversity while producing smoother control signals. The Savitzky–Golay filter (Section 2.2.4) addresses this trade-off effectively.

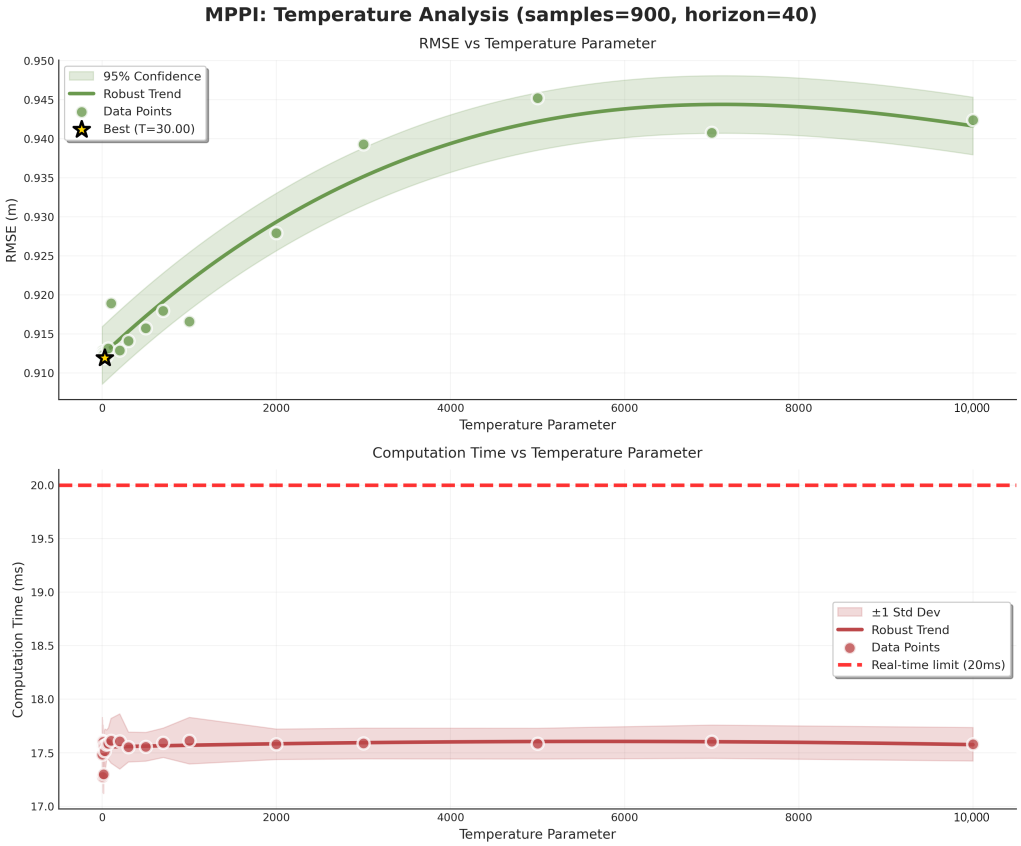


Figure 15. Effect of MPPI temperature parameter on control performance.

4.3. Real-Time Feasibility Assessment

This section summarizes the computational performance results, providing clear guidance on the feasibility of real-time deployment of both control strategies on embedded UAV hardware.

NMPC Real-time Characteristics:

- Consistent sub-15 ms performance on Jetson Orin Nano;
- CPU-only deployment, thus not dependent on GPU acceleration;
- Predictable computational overhead with low variance;
- Excellent safety margins for real-time operation (for 50 Hz control rate).

MPPI Real-time Characteristics:

- Requires GPU acceleration for embedded deployment;
- 17.63 ms median computation time (GPU) vs. 29.15 ms (CPU);
- Parameter-dependent performance requiring careful tuning;
- JAX JIT compilation provides significant optimization.

Deployment Recommendations:

1. NMPC: Suitable for platforms with limited GPU capabilities and applications requiring predictable computational overhead;
2. MPPI: Requires modern embedded platforms with GPU support; optimal for tracking performance-critical applications;
3. MPPI Parameter Selection: Chosen configuration to comply with the 50 Hz computational limit (i.e., execution time below 20 ms) imposed by the PX4 hardware: 40 horizon steps, 800–1250 samples, temperature $\lambda = 10^3$.

Both controllers successfully achieve real-time performance on the NVIDIA Jetson Orin Nano, validating the feasibility of advanced predictive control strategies on standard embedded robotics hardware. Figure 16 provides a decision framework for practitioners to select the appropriate control strategy based on their specific platform capabilities and performance requirements.

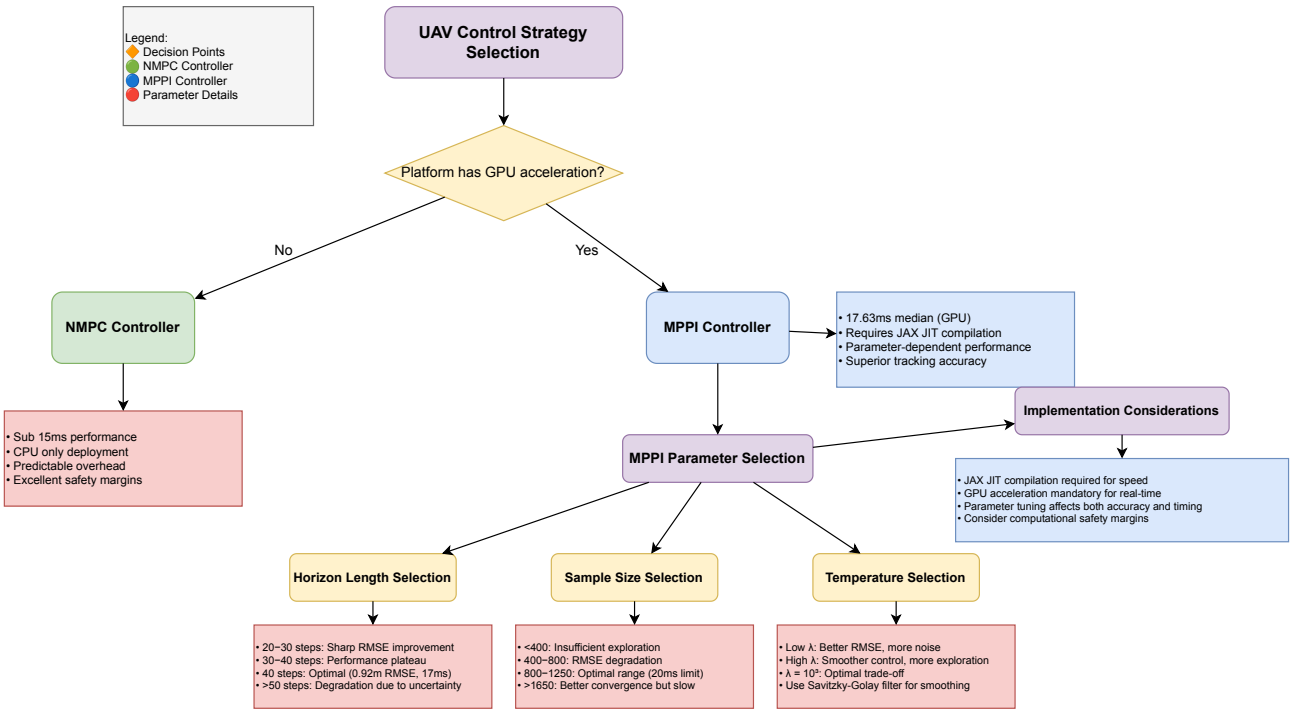


Figure 16. Decision tree for UAV control strategy selection and MPPI parameter configuration based on platform capabilities and real-time constraints.

5. Conclusions and Discussion

5.1. Key Findings and Performance Analysis

The experimental results demonstrate several critical insights for predictive control implementation on embedded UAV systems. MPPI showed promising trajectory tracking performance, with an 18.6% improvement in overall RMSE compared to NMPC at equivalent horizon lengths (0.6897 m vs. 0.8480 m). When comparing NMPC performance across different horizons, extending from N = 20 to N = 40 yielded a modest 4.4% improvement in overall RMSE, though MPPI maintained its performance advantage. While MPPI showed poorer computational performance, the execution times remained within acceptable safety margins for real-time operation, suggesting its potential viability for UAV control applications.

However, the robustness analysis revealed important performance trade-offs (Tables 7 and 8). Although MPPI maintained better absolute tracking accuracy under

noisy conditions (153.30 mm vs. 384.23 mm RMSE), NMPC demonstrated significantly superior robustness, with only 12.1% performance degradation compared to MPPI's 70.1% degradation when sensor noise was introduced. The dynamic reference tracking evaluation (Figure 11) confirmed MPPI's advantages in transient scenarios, showing faster convergence and better overall performance during trajectory switches from figure-8 to circular paths.

5.2. Technical Contributions

This works makes the following contributions to the UAV control research community:

Framework Accessibility: We demonstrated that widely adopted academic and industrial computational frameworks—CasADi/Acados for NMPC and JAX for MPPI—can deliver satisfactory real-time performance on standard robotics hardware without requiring highly specialized implementations. This finding significantly enhances accessibility and reproducibility for the broader research community.

Modular Architecture: The developed ROS 2 architecture enables runtime controller selection through launch parameters, facilitating rapid prototyping and seamless integration of future control algorithms. This design addresses a critical gap in the literature, where controller implementations are often monolithic and difficult to modify.

Comprehensive Evaluation: Unlike previous studies that focus primarily on tracking accuracy, this work provides a detailed analysis of computational overhead and real-time feasibility—factors critical for deployment on resource-constrained embedded systems.

Transparent Implementation: Both algorithms are implemented using open-source frameworks with complete parameter disclosure, ensuring full reproducibility and enabling fair comparative evaluation.

5.3. Practical Implications

The results provide valuable guidance for practitioners implementing predictive control on UAV platforms. The demonstrated effectiveness of standard computational frameworks on commercially available embedded hardware reduces barriers to adoption and enables rapid deployment of these control strategies in real-world applications.

On the MPPI side, the parameter sensitivity analysis and optimal configuration guidelines enable engineers to make informed trade-offs between control performance and computational requirements based on specific mission constraints and hardware limitations.

5.4. Future Research Directions

Several promising research avenues emerge from this work:

Environmental Robustness: Extending the comparative analysis to include disturbance rejection capabilities under realistic wind conditions and varying atmospheric disturbances.

Platform Scalability: Evaluating performance on more resource-constrained platforms with limited or no GPU capabilities, potentially including ARM-based processors and FPGA implementations, such as other NMPC works.

Learning-Enhanced Frameworks: Integration of learned dynamics models within both frameworks to address model uncertainties and unmodeled dynamics, particularly relevant for UAV operations in complex environments.

Electromagnetic Attack Resilience: Investigation of IEMI attacks on IMU sensors and development of hardened quadrotor designs with electromagnetic shielding. Recent research [3] demonstrates that medium-power electromagnetic attacks can significantly compromise sensor performance, indicating the need for integrated hardware protection mechanisms in future UAV platform designs.

This study establishes a foundation for selecting and implementing computationally efficient predictive control strategies for UAV applications, offering clear guidance on framework selection, parameter tuning, and hardware requirements for real-time trajectory tracking in practical deployment scenarios.

Author Contributions: Conceptualization, R.E. and M.M.; methodology, R.E.; software, R.E.; validation, R.E., M.M. and E.C.; formal analysis, R.E.; investigation, R.E.; resources, E.C.; data curation, R.E.; writing—original draft preparation, R.E.; writing—review and editing, R.E., M.M. and E.C.; visualization, R.E.; supervision, E.C.; project administration, E.C.; funding acquisition, E.C. All authors have read and agreed to the published version of the manuscript.

Funding: This study was carried out within the "National Research Centre for Agricultural Technologies–AGRITECH" and received funding from the European Union Next-GenerationEU (PIANO NAZIONALE DI RIPRESA E RESILIENZA (PNRR)–MISSIONE 4 COMPONENTE 2, INVESTIMENTO 1.4–Avviso n. 3138 del 16/12/2021, Codice Programma CN00000022). This manuscript reflects only the authors' views and opinions; neither the European Union nor the European Commission can be considered responsible for them. This research was carried out in the frame of project "4IPLAY", funded within Programme PRIN, Project ID: 20228KBR5R.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data used in the current study is available from the corresponding author upon reasonable request.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Hanover, D.; Foehn, P.; Sun, S.; Kaufmann, E.; Scaramuzza, D. Performance, Precision, and Payloads: Adaptive Nonlinear MPC for Quadrotors. *IEEE Robot. Autom. Lett.* **2022**, *7*, 690–697. [CrossRef]
2. Sreenath, K.; Lee, T.; Kumar, V. Geometric control and differential flatness of a quadrotor UAV with a cable-suspended load. In Proceedings of the 52nd IEEE Conference on Decision and Control, Firenze, Italy, 10–13 December 2013; pp. 2269–2274.
3. Boukabou, I.; Kaabouch, N.; Rupanetti, D. Cybersecurity challenges in UAV systems: IEMI attacks targeting inertial measurement units. *Drones* **2024**, *8*, 738. [CrossRef]
4. Zhang, Z.; Zhou, Y.; Zhang, Y.; Qian, B. Strong electromagnetic interference and protection in UAVs. *Electronics* **2024**, *13*, 393. [CrossRef]
5. Kamel, M.; Burri, M.; Siegwart, R. Linear vs nonlinear mpc for trajectory tracking applied to rotary wing micro aerial vehicles. *IFAC-PapersOnLine* **2017**, *50*, 3463–3469. [CrossRef]
6. Carlos, B.B.; Sartor, T.; Zanelli, A.; Frison, G.; Burgard, W.; Diehl, M.; Oriolo, G. An efficient real-time NMPC for quadrotor position control under communication time-delay. In Proceedings of the 2020 16th International Conference on Control, Automation, Robotics and Vision (ICARCV), Shenzhen, China, 13–15 December 2020. [CrossRef]
7. Verheijen, P.C.N.; Derkani, A.H.; Agarwal, Y.A.; Lazar, M.; Goswami, D. Multilevel parallel GPU implementation of SQP solvers for Nonlinear MPC. In Proceedings of the 2024 European Control Conference (ECC), Stockholm, Sweden, 25–28 June 2024; pp. 2292–2298. [CrossRef]
8. Williams, G.; Drews, P.; Goldfain, B.; Rehg, J.M.; Theodorou, E.A. Aggressive driving with model predictive path integral control. In Proceedings of the 2016 IEEE International Conference on Robotics and Automation (ICRA), Stockholm, Sweden, 16–21 May 2016; pp. 1433–1440. [CrossRef]
9. Williams, G.; Aldrich, A.; Theodorou, E.A. Model predictive path integral control: From theory to parallel computation. *J. Guid. Control Dyn.* **2017**, *40*, 344–357. [CrossRef]
10. Pravitra, J.; Ackerman, K.A.; Cao, C.; Hovakimyan, N.; Theodorou, E.A. $\mathcal{L}1$-Adaptive MPPI Architecture for Robust and Agile Control of Multirotors. In Proceedings of the 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Las Vegas, NV, USA, 24 October 2020–24 January 2021; pp. 7661–7666. [CrossRef]
11. Minařík, M.; Pěnička, R.; Vonásek, V.; Saska, M. Model predictive path integral control for agile unmanned aerial vehicles. In Proceedings of the 2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Abu Dhabi, United Arab Emirates, 14–18 October 2024; pp. 13144–13151. [CrossRef]

12. Vlahov, B.; Gibson, J.; Gandhi, M.; Theodorou, E.A. MPPI-Generic: A CUDA Library for Stochastic Optimization. *arXiv* **2024**, arXiv:2409.07563. [CrossRef]
13. Reinhardt, D.; Johansen, T.A. Control of fixed-wing UAV attitude and speed based on embedded nonlinear model predictive control. *IFAC-PapersOnLine* **2021**, *54*, 91–98. [CrossRef]
14. Pozzan, B.; Elaamery, B.; Cenedese, A. Non-Linear Model Predictive Control for autonomous landing of a UAV on a moving platform. In Proceedings of the 2022 IEEE Conference on Control Technology and Applications (CCTA), Trieste, Italy, 23–25 August 2022; pp. 1240–1245. [CrossRef]
15. Sadi, M.A.; Jamali, A.; Abang Kamaruddin, A.M.N. Optimizing UAV performance in turbulent environments using cascaded model predictive control algorithm and Pixhawk hardware. *J. Braz. Soc. Mech. Sci. Eng.* **2025**, *47*, 1–26. [CrossRef]
16. Macenski, S.; Soragna, A.; Carroll, M.; Ge, Z. Impact of ROS 2 node composition in robotic systems. *arXiv* **2023**, arXiv:2305.09933. [CrossRef]
17. Turrisi, G.; Modugno, V.; Amatucci, L.; Kanoulas, D.; Semini, C. On the benefits of GPU sample-based stochastic predictive controllers for legged locomotion. In Proceedings of the 2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Abu Dhabi, United Arab Emirates, 14–18 October 2024; pp. 13757–13764. [CrossRef]
18. Nascimento, T.; Saska, M. Embedded Fast Nonlinear Model Predictive Control for Micro Aerial Vehicles. *J. Intell. Robot. Syst.* **2021**, *103*, 74. [CrossRef]
19. Elhesasy, M.; Dief, T.N.; Atallah, M.; Okasha, M.; Kamra, M.M.; Yoshida, S.; Rushdi, M.A. Non-Linear Model Predictive Control Using CasADi Package for Trajectory Tracking of Quadrotor. *Energies* **2023**, *16*, 2143. [CrossRef]
20. Verschueren, R.; Frison, G.; Kouzoupis, D.; Frey, J.; van Duijkeren, N.; Zanelli, A.; Novoselnik, B.; Albin, T.; Quirynen, R.; Diehl, M. acados—A modular open-source framework for fast embedded optimal control. *Math. Program. Comput.* **2022**, *14*, 147–183. [CrossRef]
21. Bradbury, J.; Frostig, R.; Hawkins, P.; Johnson, M.J.; Leary, C.; Maclaurin, D.; Necula, G.; Paszke, A.; VanderPlas, J.; Wanderman-Milne, S.; et al. *JAX: Composable Transformations of Python+NumPy Programs*, version 0.3.13; Google: Mountain View, CA, USA, 2018. Available online: https://github.com/jax-ml/jax (accessed on 16 August 2025).
22. Mahony, R.; Kumar, V.; Corke, P. Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor. *IEEE Robot. Autom. Mag.* **2012**, *19*, 20–32. [CrossRef]
23. Meier, L.; Honegger, D.; Pollefeys, M. PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms. In Proceedings of the 2015 IEEE International Conference on Robotics and Automation (ICRA), Seattle, WA, USA, 26–30 May 2015. [CrossRef]
24. Andrejev, E.M.; Manoharan, A.; Unt, K.E.; Singh, A.K. ΠI-MPPI: A projection-based model predictive path integral scheme for smooth optimal control of fixed-wing aerial vehicles. *IEEE Robot. Autom. Lett.* **2025**, *10*, 6496–6503. [CrossRef]
25. Kovryzhenko, Y.; Li, N.; Taheri, E. A Control System Design and Implementation for Autonomous Quadrotors with Real-Time Re-Planning Capability. *Robotics* **2024**, *13*, 136. [CrossRef]
26. Jetson Orin Nano Developer Kit Getting Started Guide | NVIDIA Developer. Available online: https://developer.nvidia.com/embedded/learn/jetson-orin-nano-devkit-user-guide/index.html (accessed on 16 August 2025).
27. Macenski, S.; Foote, T.; Gerkey, B.; Lalancette, C.; Woodall, W. Robot Operating System 2: Design, architecture, and uses in the wild. *Sci. Robot.* **2022**, *7*, eabm6074. [CrossRef] [PubMed]
28. Controller Diagrams | PX4 Guide (Main). Available online: https://docs.px4.io/main/en/flight_stack/controller_diagrams (accessed on 16 August 2025).