FLYINGTRUST: A Benchmark for Quadrotor Navigation Across Scenarios and Vehicles

Gang Li[1], Chunlei Zhai[1], Teng Wang[1], Songyuan Li[1], Shangsong Jiang[2], and Xiangwei Zhu[1*] *Member, IEEE*

***Abstract*—A significant challenge in the practical deployment of quadrotor visual navigation is the inconsistent performance observed when a single algorithm is transferred across different physical platforms and operating environments. This gap stems from a fundamental research shortcoming: evaluating algorithms on single, idealized platforms. This practice ignores the joint effect of vehicle kinodynamics and scenario geometry on robustness, making widespread field deployment costly, time-consuming, and high-risk. We introduce FLYINGTRUST, a high-fidelity, configurable benchmarking framework to systematically quantify this joint effect before deployment. Our framework features two key innovations: 1) A compact, physical model to quantify vehicle capability via maximum thrust-to-weight ratio and axis-wise maximum angular acceleration, and 2) a standardized evaluation pipeline. This pipeline pairs a heterogeneous library of 36 real and virtual platforms with a diverse scenario library, which includes both classic navigation scenes and novel "stress-test" environments. Using a composite scoring method that penalizes instability, we evaluated representative optimization-based and learning-based planners. The results reveal systematic patterns: navigation success depends predictably on platform capability and scene geometry, and different algorithms exhibit distinct preferences and failure modes across the evaluated conditions. FLYINGTRUST is the first tool to de-risk development, enabling predictive algorithm selection and highlighting the practical necessity of integrating platform capability and scenario structure into algorithm design, evaluation, and selection to achieve robust performance.**

***Index Terms*—Benchmark, Quadrotor, Visual navigation, Kinodynamics, Simulation**

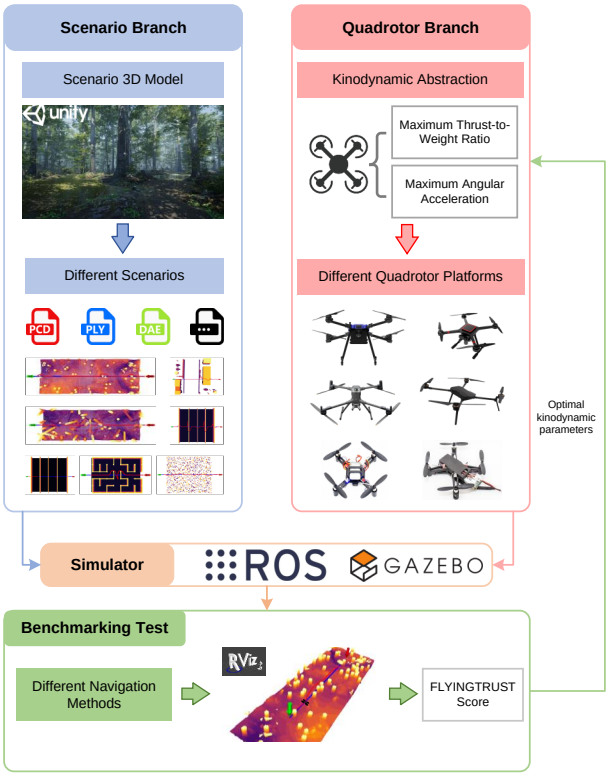


Fig. 1. **Conceptual workflow of the FLYINGTRUST benchmark.** The framework is divided into two parallel branches: (1) Scenario Branch : Diverse navigation scenarios are designed as 3D models in Unity and exported into standard file formats (e.g., .PCD, .PLY, .DAE) to create the scenario library. (2) Quadrotor Branch : A heterogeneous set of quadrotor platforms is defined via "Kinodynamic Abstraction" , which models each vehicle by its Maximum Thrust-to-Weight Ratio and Maximum Angular Acceleration. Both branches converge in the "Simulator" , which uses ROS and Gazebo to create the test environment. Finally, the "Benchmarking Test" systematically evaluates 'Different Navigation Methods' in these combined environments, with results visualized in RViz and aggregated into a final "FLYINGTRUST Score".

I. INTRODUCTION

QUADROTORS are a prominent class of rotary-wing Unmanned Aerial Vehicles (UAVs), operated without onboard human pilots [1]. By independently modulating the speeds of four motor-propeller units, a quadrotor can generate collective thrust for vertical motion and differential thrust and reaction torques for attitude control. These capabilities enable six degrees of freedom motion combined with fine low-speed control, which drive extensive adoption of quadrotors in precision agriculture, infrastructure inspection, high-resolution mapping, environmental monitoring and disaster response [2]–[14]. Central to autonomous operation is visual navigation, a pipeline that integrates camera-based perception, localization and mapping, trajectory planning and continuous control. Over the last decade, many high-performance visual navigation methods have been developed, ranging from classical optimization-based planners to recent learning-based approaches [15]–[18].

In practice, however, this apparent success often proves inconsistent. An algorithm's performance can vary unpredictably when transferred from simulation to reality, or across different physical platforms and new environments. This "sim-to-real-and-platform" gap stems from a fundamental shortcoming in existing research: algorithms are typically evaluated on a sin-

[1]School of Electronics and Communication Enginnering, Sun Yet-sen University, Shenzhen, China
[2]Tianwei Xunda (Hunan) Technology Co., Ltd., Hunan, China
*Corresponding author: Xiangwei Zhu
Manuscript received XX, 2025; revised XX, 2025.
The FLYINGTRUST project website is available at: https://flyingtrust.github.io/.

gle, idealized platform. This practice ignores how the joint effect of vehicle kinodynamics (e.g., thrust and agility) and scenario geometry (e.g., obstacle density and structure) impacts robustness, making widespread field deployment costly, time-consuming, and high-risk. When a platform's physical limits do not match an algorithm's implicit assumptions, specific and catastrophic failure modes emerge. For example, as our own findings in this paper will demonstrate, a high-performing algorithm like Fast-Planner [15] can successfully navigate a forest but fails almost completely in a "Narrow-Gap" scenario that demands precise yaw control. Similarly, an optimizer like Path-Guided PGO [19], while functional in some settings, fails in over 95% of trials in a "Sudden-Drop" scenario that requires high pitch authority. These deployment failures highlight a critical gap: the community lacks a systematic, reproducible tool for quantifying this joint effect before committing to costly hardware tests. To address this requirement, we present **FLYINGTRUST**, a configurable simulation-based benchmark for quadrotor visual navigation.

As illustrated in Fig. 1, FLYINGTRUST is explicitly designed to run in simulation as a proactive, low-risk evaluation stage that guides subsequent real-world testing. The benchmark identifies brittle algorithm-platform-scenario interactions so that researchers and practitioners can prioritize which algorithm-platform pairs deserve costly field trials, and it does so without requiring any new physical flight experiments. This simulation-first design positions the benchmark as a development and prioritization tool, rather than as a replacement for final real-world validation. The benchmark pursues three complementary aims. First, it quantifies navigation success as a function of both environment characteristics and platform capability, thereby making brittle interactions visible early. Second, it exposes common, interpretable failure modes that arise from specific platform limitations, for example insufficient thrust-to-weight ratio or limited angular acceleration about particular axes. Third, it provides a standardized, reproducible evaluation protocol so that different research groups can compare methods fairly and prioritize which algorithm-platform pairs should advance to field testing.

FLYINGTRUST implements the benchmark as an end-to-end simulation pipeline. First, we define a compact, parametric representation of quadrotor performance and construct a fixed, heterogeneous set of platform profiles, including documented real platforms and a set of virtual platforms obtained by interpolation within the documented design space. Next, we assemble a fixed scenario library that contains both common navigation scenes and targeted stress-test environments. The scalable test set is produced by cross-joining the selected platform profiles with the scenario instances to generate all platform-scene combinations to be evaluated. For each platform-scene pair, the framework runs repeated trials under standardized task specifications and flight limits, collects success and diagnostic metrics, and computes uncertainty-aware summary statistics. Finally, results are aggregated via a principled composite score that weights scenario importance, platform importance, and performance stability to produce an intuitive ranking and to highlight brittle algorithm-platform-scene interactions for follow-up field evaluation.

In this paper we adopt a simulation-first benchmarking framework, FLYINGTRUST, to systematically evaluate how environmental geometry and platform kinodynamic capability jointly influence the robustness of visual quadrotor navigation algorithms. The benchmark pairs 18 curated real-world quadrotor platforms with 18 virtual platforms generated by interpolation in the documented design space, and evaluates navigation across seven representative scenes. This yields 252 platform-scene combinations; each combination is tested across multiple independent trials to report success rates, confidence intervals and stability metrics. The manuscript integrates method description, experimental protocol and analysis: Section II classifies current visual quadrotor navigation approaches, Section III describes the FLYINGTRUST design and implementation, Section IV presents the experimental setup and raw results, provides statistical analysis and practical recommendations for algorithm selection, Section V discusses limitations and directions for future extensions. FLYINGTRUST is intended to make brittle algorithm-platform-scene interactions visible in simulation so researchers can prioritize which algorithm-platform pairs merit costly real-world flight tests, thereby reducing development cost and risk.

II. Landscape of Visual Quadrotor Navigation Methods

Most modern visual navigation systems for quadrotors follow a three-stage paradigm [20], with some exceptions that adopt fully end-to-end learning. In the dominant approach, perception produces information from onboard sensors and a planner uses that information to generate a collision-free trajectory, after which a controller tracks the planned trajectory on the physical vehicle. This design pattern highlights two interdependent questions that determine real-world performance. First, given diverse and challenging environments, can the planner reliably produce collision-free, kinodynamically-feasible trajectories? Second, once such a trajectory is available, can the controller faithfully execute it on a particular quadrotor platform under its actuator and inertia limits? The benchmark proposed in this paper explicitly targets these two questions by evaluating planners and controllers jointly across a range of scene geometries and platform performance profiles.

As illustrated in Fig. 2, we now summarize the principal methodological families encountered in the literature, describe their characteristic strengths and limitations, and comment on the kinds of environments and vehicle capabilities to which they are best suited.

A. Optimization-based

Conventional optimization-based pipelines separate perception and mapping, trajectory optimization, and robust control. The typical workflow builds a local environment representation or distance field, parameterizes a trajectory (often with splines), and solves a constrained optimization that balances smoothness, collision avoidance and dynamic feasibility. The main strengths of these methods are interpretability and the ability to enforce kinodynamic constraints explicitly during

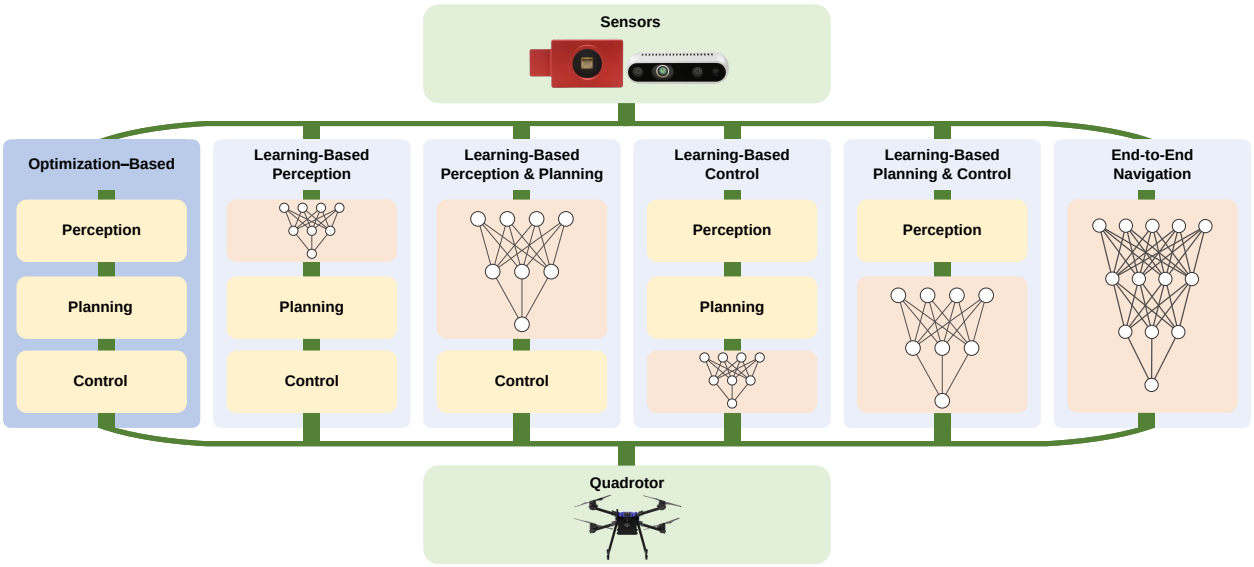


Fig. 2. **Taxonomy of visual quadrotor navigation algorithms.** A flowchart of common navigation pipelines, from sensors to quadrotor. The figure contrasts the modular, optimization-based pipeline (left) with various learning-based approaches, which are categorized by the components replaced with neural networks (e.g., perception, planning, control).

planning. They therefore suit structured or moderately cluttered environments where reliable local maps and distance information are available, and they benefit platforms that provide sufficient thrust authority and angular acceleration to realize aggressive trajectories. Their primary limitations are computation and mapping cost on resource-constrained platforms, and susceptibility to local minima in gradient-based solvers; topology-aware or multi-start strategies help mitigate the latter. Representative works include real-time replanning using a 3D ring buffer and spline parameterization [21], B-spline trajectory optimization with dynamics-aware initial search [15], path-guided optimization that leverages multiple homotopy classes [19], and ESDF-free local replanners that focus computation near a guide path [16].

B. Learning-based

Learning-based methods replace one or more traditional modules with neural networks [22], [23]. Depending on which functions are learned, these methods differ substantially in capabilities and requirements. Below we briefly characterize the main subclasses and note the environments and platform types for which they tend to be most appropriate.

1) Learning-Based Perception: Learned perception modules (for example CNNs) detect task-relevant features or produce compact scene embeddings from RGB, depth or event sensors. When trained appropriately, they can improve robustness to visual degradation and reduce the end-to-end latency of the perception stack. They are most useful in scenarios with challenging lighting or texture conditions where handcrafted detectors fail. Practical constraints include the inference cost on embedded hardware and the need for annotated or high-fidelity simulated training data [24].

2) Learning-Based Perception and Planning: Replacing mapping and local planning with learned models enables navigation without explicit maps. Image-to-waypoint and short-horizon learned planners provide reactive behavior with low online computation, which is advantageous in high-speed or computation-limited platforms. These methods perform well when training data reflect the test distributions; they are vulnerable to out-of-distribution scene geometry and typically require careful dataset design or domain randomization [17], [25].

3) Learning-Based Control: Reinforcement learning can produce low-level controllers that map observed states to actuator commands. RL controllers can achieve excellent performance in simulation and may outperform hand-tuned controllers on some metrics. Their drawbacks are limited formal stability guarantees and sensitivity to simulator mismatch, which complicate direct transfer to hardware without additional safety measures [26], [27].

4) Learning-Based Planning and Control: Methods that jointly learn planning and control produce policies that directly map observations to actions or short trajectories. Such approaches can achieve highly reactive, time-efficient behavior in tasks like racing, provided that training covers the necessary scene and dynamics variability. Their generalization and safety in arbitrary environments remain active research challenges [18], [28], [29].

5) End-to-End Navigation: End-to-end systems map sensor inputs to control outputs. Two operational variants are common:

- **Modular end-to-end:** learned modules replace individual pipeline blocks (perception, planning, control) and are trained jointly.
- **Fully end-to-end:** a single model maps raw sensors to low-level commands.

End-to-end designs can reduce latency and exploit large datasets, but they pose challenges for interpretability, enforcement of hard safety constraints, and cross-platform generalization. Hybrid architectures that combine learned components

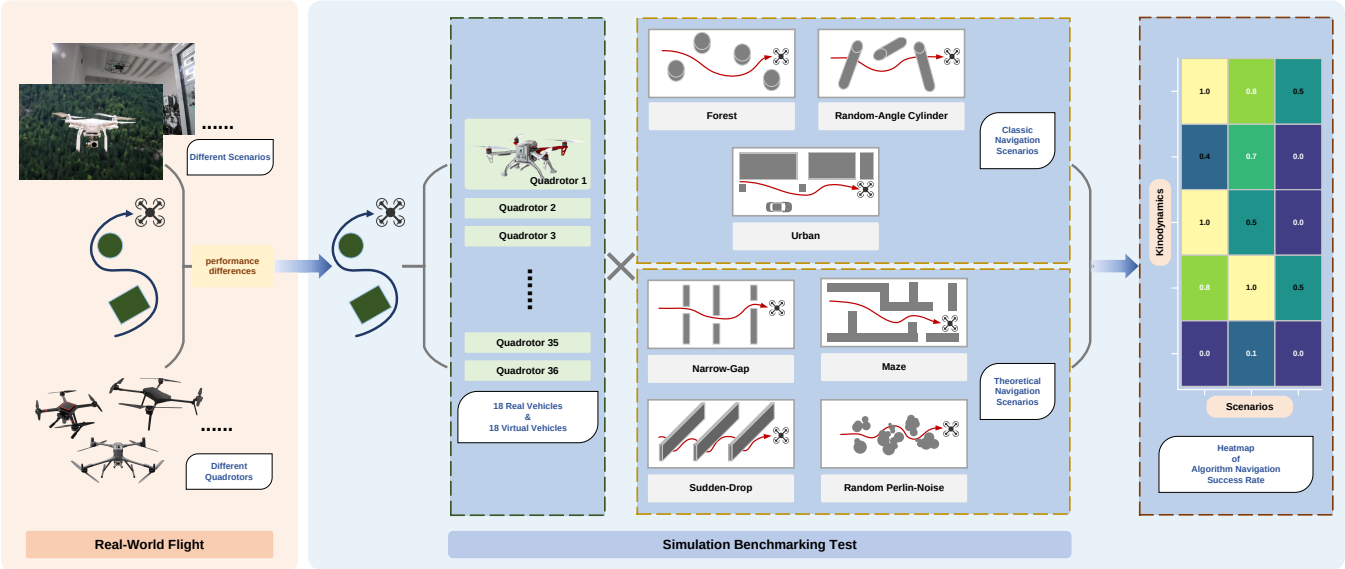


Fig. 3. **Overview of the FLYINGTRUST benchmarking pipeline.** The benchmark pairs a fixed set of platform profiles with a fixed scenario library to form platform-scenario combinations; each combination is evaluated with multiple trials and summarized via a composite scoring scheme.

with optimization-based safety or control layers represent a practical compromise [30]–[32].

Although many visual navigation algorithms, both optimization-based and learning-based, report strong performance in simulation, most are developed and tested under simplified or idealized conditions. These conditions often assume near-ideal actuator response, negligible sensor latency, and stable kinodynamics, assumptions that rarely hold in practice. As a result, algorithm performance often varies or degrades significantly across different platforms and environments. Moreover, the literature provides limited systematic analysis of how specific kinodynamic characteristics and scenario structures jointly influence robustness.

To address this gap, we introduce FLYINGTRUST, illustrated in Fig. 3, a comprehensive and configurable benchmarking framework that enables rigorous and reproducible evaluation of visual navigation algorithms. FLYINGTRUST explicitly considers both platform performance and scenario geometry, providing a principled way to analyze how these factors interact to shape algorithm robustness. By doing so, it supports fair comparison across methods and offers practical guidance for selecting algorithms suitable for specific quadrotor platforms and navigation scenarios.

III. The FLYINGTRUST Benchmarking Framework

A. Kinodynamic performance of quadrotors

A quadrotor is a typical six-degree-of-freedom vertical take-off and landing (VTOL) aircraft [33]. As illustrated in Fig. 4, by adjusting the rotational speeds of its four motors, a quadrotor can perform vertical motion along the z axis, forward and backward motion along the x axis, lateral motion along the y axis, as well as rotational motions including pitch (around the y axis), roll (around the x axis), and yaw (around the z axis) [34].

In extensive engineering practice we observe that quadrotor flight frequently involves unsteady, time varying maneuvers such as acceleration, deceleration, climbing, descending, orbiting and aggressive acrobatics. These behaviors reflect two related but distinct aspects of flight dynamics:

- **Maneuverability** denotes the vehicle's ability to change its speed, altitude or heading over a finite time horizon and can be decomposed into speed, altitude and directional maneuverability.
- **Agility** emphasizes transient response, that is, how quickly and precisely the vehicle can switch between motion states; typical facets are roll agility, pitch agility and yaw agility.

Although some studies report metrics such as "maximum turn rate" or short-duration aggressive turn performance [35], there is little consistency in how sustained, continuous turn rate versus transient peak turn behavior is defined or used. These metrics tend to be numerous, fragmented, and difficult to compare across platforms and scenarios. From rigid body dynamics we note that linear accelerations and angular accelerations are determined directly by the net forces and torques acting on the vehicle, and these accelerations govern position and attitude evolution. Therefore, we adopt two compact, physically interpretable indicators to characterize kinodynamic performance: The **maximum thrust to weight ratio** ($\mathrm{TWR}_{\max}$), which primarily quantifies maneuverability, and the **maximum angular accelerations about the three body axes** ($\boldsymbol{\alpha_{\max}}$), which capture agility.

To validate that these indicators effectively capture the essential flight performance, we conducted a controlled simulation. As illustrated in Fig. 5, five platforms with different kinodynamic parameters were tasked with tracking an aggressive, stepped reference trajectory using an identical controller and a uniform maximum speed limit of $5\ \mathrm{m/s}$. The results clearly visualize the impact of these indicators. Platforms

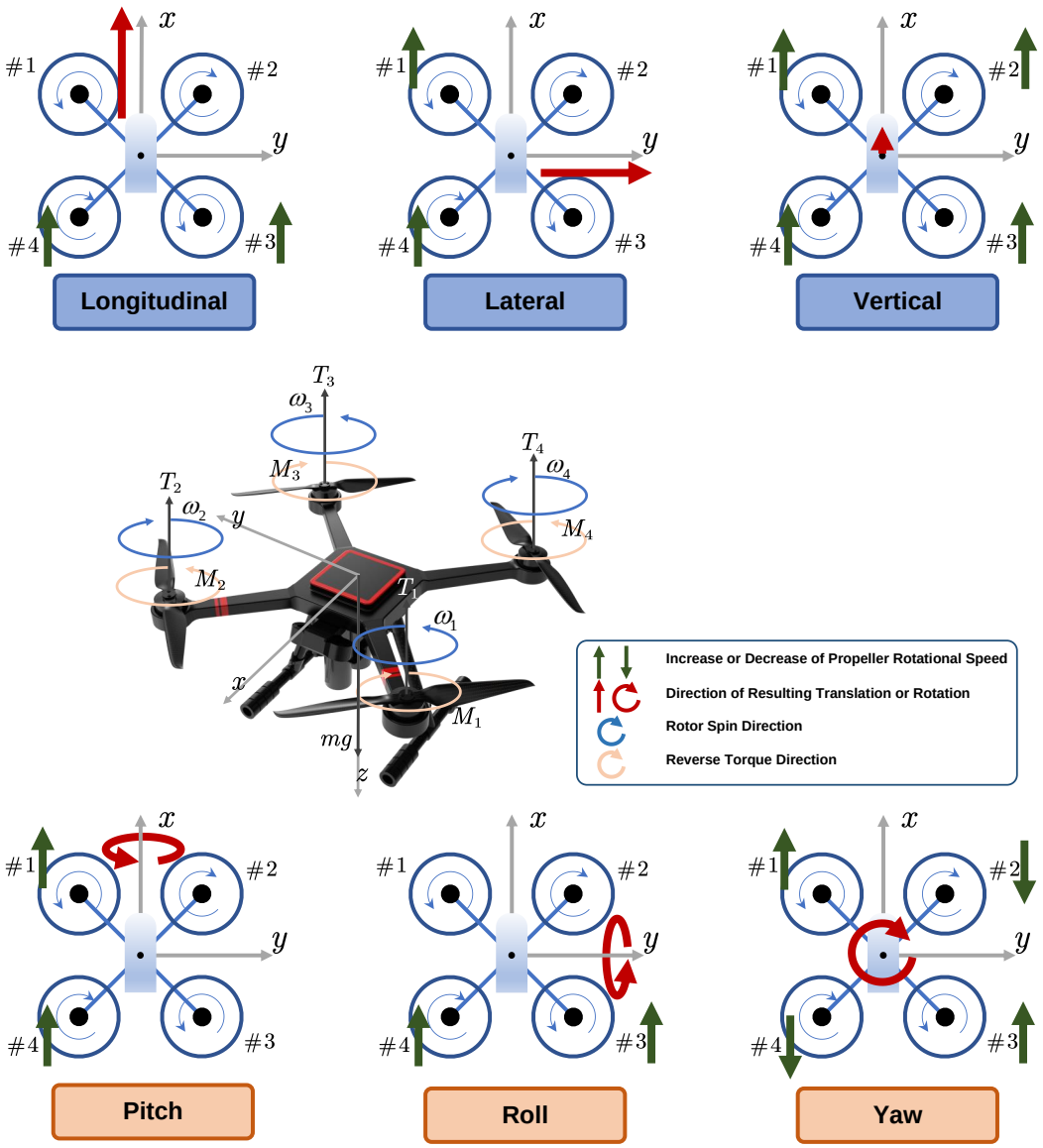


Fig. 4. **Illustration of how rotor speed modulation produces the six degrees of freedom motions of a quadrotor.** The central diagram illustrates forces (e.g., T_i, mg) and torques (M_i) on the vehicle. The surrounding schematics show the required changes in rotor speed (green arrows) to achieve the six canonical motions: longitudinal, lateral, vertical, pitch, roll, and yaw (red arrows).

with high $\mathrm{TWR}_{\max}$ and $\boldsymbol{\alpha_{\max}}$ (e.g., the blue line) exhibit superior tracking, characterized by rapid initial acceleration and tight, precise turning radii at the sharp corners. Conversely, platforms with low kinodynamic capability (e.g., the purple line) demonstrate significant tracking lag and are forced into wide, sweeping turns, failing to follow the reference path. This simulation confirms that $\mathrm{TWR}_{\max}$ and $\boldsymbol{\alpha_{\max}}$ are descriptive and effective metrics for quantifying a platform's maneuverability and agility in dynamically demanding tasks.

Maximum Thrust-to-Weight Ratio: The thrust-to-weight ratio is defined as the total thrust generated by the four rotors divided by the quadrotor's weight, which directly reflects its maximum linear acceleration. The thrust produced by each rotor is proportional to the square of its rotational speed:

$$T = c_{\mathrm{T}} \sum_{i=1}^{4} \omega_i^2, \tag{1}$$

here, ω_i denotes the rotational speed of rotor i, and c_{T} is the thrust coefficient determined by the propeller parameters. The maximum thrust-to-weight ratio is given by:

$$\mathrm{TWR}_{\max} = \frac{T_{\max}}{mg} = \frac{c_{\mathrm{T}} \sum_{i=1}^{4} \omega_{i,\max}^2}{mg} \tag{2}$$

$\omega_{i,\max}$ denotes the maximum achievable rotor speed of motor i under sustained operation.

Maximum Angular Acceleration (Three Axes): During navigation, a quadrotor must also adjust its orientation to change the direction of its thrust vector. This is achieved by

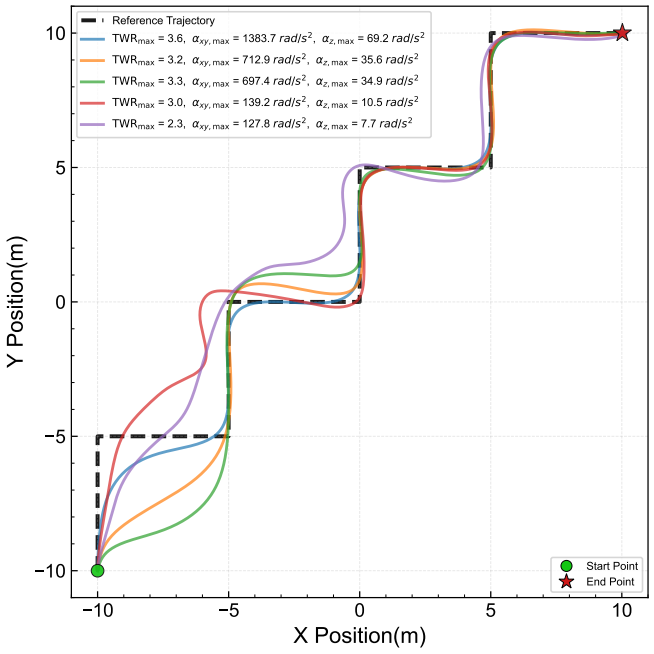


Fig. 5. **Visual validation of the proposed kinodynamic indicators for characterizing platform performance.** Five quadrotor platforms with varying kinodynamic parameters (see legend) were tasked to follow an aggressive, stepped reference trajectory (black dashed line) using an identical controller and a uniform maximum speed limit of 5 m/s.

generating torques around the three body axes. The reaction torque from each rotor is expressed as:

$$M_i = c_{\mathrm{M}}\omega_i^2, \tag{3}$$

here, c_{M} is the torque coefficient of the quadrotor, which is determined by the geometric and aerodynamic properties of the propellers. During rotational motion around the three principal axes, the rotor speeds ω_i determine the control torque vector $\boldsymbol{\tau} = [\tau_x, \tau_y, \tau_z]^{\mathrm{T}}$. It is important to note that the method for computing these torques varies depending on the quadrotor's structural configuration, such as the "plus" or "cross" layout [36], as illustrated in Fig. 6.

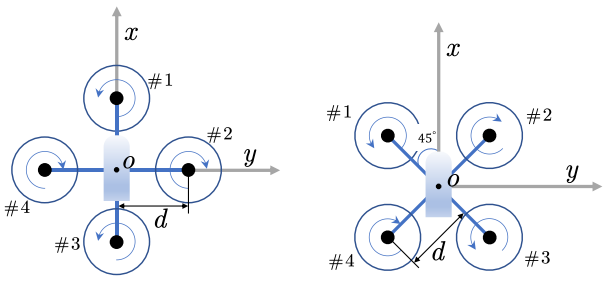


Fig. 6. **Comparison of "plus" and "cross" quadrotor layouts.** This figure illustrates the two common motor configurations: (a) the "plus" layout, where motor arms align with the body's x and y axes, and (b) the "cross" layout, where the arms are rotated by $45°$.

For a quadrotor with a "plus" layout:

$$\begin{cases} \tau_x = dc_{\mathrm{T}}(-\omega_2^2 + \omega_4^2) \\ \tau_y = dc_{\mathrm{T}}(\omega_1^2 - \omega_3^2) \\ \tau_z = c_{\mathrm{M}}(\omega_1^2 - \omega_2^2 + \omega_3^2 - \omega_4^2) \end{cases} \tag{4}$$

For a quadrotor with an "cross" layout:

$$\begin{cases} \tau_x = dc_{\mathrm{T}}(\frac{\sqrt{2}}{2}\omega_1^2 - \frac{\sqrt{2}}{2}\omega_2^2 - \frac{\sqrt{2}}{2}\omega_3^2 + \frac{\sqrt{2}}{2}\omega_4^2) \\ \tau_y = dc_{\mathrm{T}}(\frac{\sqrt{2}}{2}\omega_1^2 + \frac{\sqrt{2}}{2}\omega_2^2 - \frac{\sqrt{2}}{2}\omega_3^2 - \frac{\sqrt{2}}{2}\omega_4^2) \\ \tau_z = c_{\mathrm{M}}(\omega_1^2 - \omega_2^2 + \omega_3^2 - \omega_4^2) \end{cases} \tag{5}$$

In the above equation, $d \in \mathbb{R}^+$ represents the distance from the center of the quadrotor body to any of its motors. This value corresponds to the length of the moment arm during rotational motion around the roll or pitch axis.

By modeling the quadrotor as a rigid body, its moment of inertia can be expressed according to the principles of rigid-body dynamics:

$$\mathbf{J} = \begin{bmatrix} J_{xx} & -J_{xy} & -J_{xz} \\ -J_{yx} & J_{yy} & -J_{yz} \\ -J_{zx} & -J_{zy} & J_{zz} \end{bmatrix}, \tag{6}$$

here, the inertia tensor satisfies $J_{xy} = J_{yx}$, $J_{xz} = J_{zx}$, and $J_{yz} = J_{zy}$. For a standard quadrotor with center-symmetric geometry, all off-diagonal inertia terms vanish:

$$J_{xy} = J_{yx} = J_{xz} = J_{zx} = J_{yz} = J_{zy} = 0, \tag{7}$$

hence the tensor simplifies to:

$$\mathbf{J} = \begin{bmatrix} J_{xx} & 0 & 0 \\ 0 & J_{yy} & 0 \\ 0 & 0 & J_{zz} \end{bmatrix}, \tag{8}$$

here, $J_{xx}, J_{yy}, J_{zz} \in \mathbb{R}^+$ are the principal moments of inertia. Consequently, the quadrotor's maximum angular acceleration can be written as:

$$\begin{aligned} \boldsymbol{\alpha}_{\mathbf{max}} &= \begin{bmatrix} \alpha_{x,\max} \\ \alpha_{y,\max} \\ \alpha_{z,\max} \end{bmatrix} = \mathbf{J}^{-1}\boldsymbol{\tau}_{\mathbf{max}} \\ &= \begin{bmatrix} \frac{1}{J_{xx}} & 0 & 0 \\ 0 & \frac{1}{J_{yy}} & 0 \\ 0 & 0 & \frac{1}{J_{zz}} \end{bmatrix} \begin{bmatrix} \tau_{x,\max} \\ \tau_{y,\max} \\ \tau_{z,\max} \end{bmatrix}, \\ &= \begin{bmatrix} \frac{\tau_{x,\max}}{J_{xx}} \\ \frac{\tau_{y,\max}}{J_{yy}} \\ \frac{\tau_{z,\max}}{J_{zz}} \end{bmatrix} \end{aligned} \tag{9}$$

FLYINGTRUST evaluates a quadrotor's dynamic performance using the maximum thrust-to-weight ratio and the maximum angular acceleration along the three axes, which respectively capture its peak thrust generation capability and its ability to rapidly reorient its thrust vector.

$$\mathbf{P} = \begin{bmatrix} \mathrm{TWR}_{\max} \\ \alpha_{x,\max} \\ \alpha_{y,\max} \\ \alpha_{z,\max} \end{bmatrix} \tag{10}$$

Because a quadrotor is typically designed to be center-symmetric, its principal moments of inertia about the x and y

axes are often treated as equal ($J_{xx} \approx J_{yy}$). However, the inclusion of onboard components, such as a fixed forward-facing camera, introduces a slight mass asymmetry. Strictly speaking, this results in a minor difference between the maximum roll acceleration ($\alpha_{x,\max}$) and pitch acceleration ($\alpha_{y,\max}$). For this benchmark, we justify the use of a unified horizontal angular acceleration metric $\alpha_{xy,\max}$ for two primary reasons:

- **Dominance of Roll in Navigation:** In many autonomous navigation scenarios, especially those involving lateral obstacle avoidance at speed, roll maneuvers are the most frequent and dynamically demanding. The vehicle's ability to quickly roll and redirect its thrust sideways is often more critical than its pitching capability for acceleration or deceleration. Therefore, the roll performance ($\alpha_{x,\max}$) serves as a practical and representative limit for horizontal agility.
- **Negligible Asymmetry Impact:** For the majority of small to medium-sized quadrotors considered in our dataset, the mass of the camera and other directional sensors is a small fraction of the total vehicle mass. This results in a minimal difference between J_{xx} and J_{yy}, making the discrepancy between their corresponding maximum angular accelerations negligible for a high-level performance benchmark.

Given these practical considerations, this paper uses the dominant roll acceleration value $\alpha_{x,\max}$ to represent the combined horizontal agility metric $\alpha_{xy,\max}$. This simplifies the kinodynamic model effectively without a significant loss of fidelity for the evaluation of forward-flight navigation tasks. Given this simplification and the symmetric motor configuration described in Equations (4) and (5), we proceed by using a single horizontal agility metric. Hence, FLYINGTRUST's characterization of kinodynamics can be reduced to:

$$\mathbf{P} = \begin{bmatrix} \mathrm{TWR}_{\max} \\ \alpha_{xy,\max} \\ \alpha_{z,\max} \end{bmatrix} \tag{11}$$

This study surveyed the propeller, motor, and frame parameters of several commercial off-the-shelf quadrotors and custom research platforms, and used these data to estimate each platform's kinodynamics performance metrics [37]. Based on the proprietary platforms used in EGO-Planner, Fast-Planner and Agilicious, we synthesized 16 virtual quadrotors. These models were generated by systematically scaling key parameters, such as vehicle mass, arm length d, and maximum propeller speed within the documented design space. From these scaled values, we then recalculated the associated moments of inertia (J) and the final kinodynamic performance metrics ($\mathrm{TWR}_{\max}$, $\alpha_{xy,\max}$, $\alpha_{z,\max}$) to ensure physical consistency. Each virtual configuration was subjected to basic feasibility checks, for example, the consistency of mass and inertia scaling and motor/propeller limits, and validated in simulation for grossly infeasible behaviour [38]. The resulting benchmark therefore combines 18 real platforms with 18 virtual models to form the FLYINGTRUST kinodynamics dataset.

As illustrated in Table I and Fig. 7, FLYINGTRUST focuses on small to medium sized quadrotors, covering a weight range from 0.5 kg to 5 kg. In the real-world subset, the mean thrust-to-weight ratio is 2.30, the mean maximum angular acceleration about the horizontal axes (x,y) is $99.92\ \mathrm{rad/s^2}$, and about the yaw axis (z) is $7.17\ \mathrm{rad/s^2}$. For the virtual models, the mean thrust-to-weight ratio is 3.47, the mean maximum angular acceleration about the horizontal axes is $824.20\ \mathrm{rad/s^2}$, and about the yaw axis is $41.88\ \mathrm{rad/s^2}$. These results confirm that most commercial platforms prioritize flight smoothness over agility, resulting in substantially lower dynamic performance compared with idealized quadrotors. Moreover, existing navigation algorithms typically assume high thrust-to-weight ratios and angular accelerations during their design, training, and validation stages.

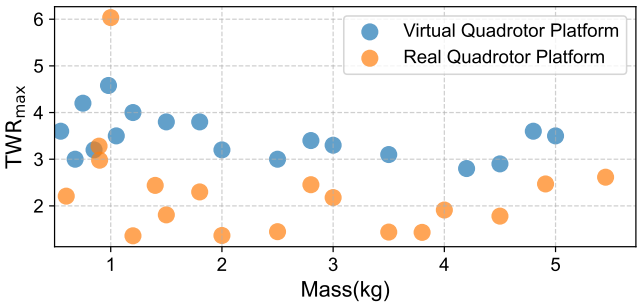
TABLE I
SUMMARY OF PERFORMANCE PARAMETERS FOR REAL AND VIRTUAL QUADROTOR PLATFORMS

Category	Quadrotor Platform	$\mathrm{TWR}_{\max}$	$\boldsymbol{\alpha}_{xy,\max}(\mathrm{rad/s^2})$	$\boldsymbol{\alpha}_{z,\max}(\mathrm{rad/s^2})$
Real Vehicles	0.60kg-EMAX	2.2	114.7	8.4
	0.895kg-DJI	3.3	107.9	14.1
	0.90kg-DJI	3.0	139.2	10.5
	1.00kg-SunnySky	6.0	227.3	13.9
	1.20kg-JFRC	1.4	84.6	7.2
	1.40kg-EMAX	2.5	94.1	6.0
	1.50kg-DJI	1.8	85.8	5.7
	1.80kg-SunnySky	2.3	127.8	7.7
	2.00kg-T-MOTOR	1.4	55.6	3.3
	2.50kg-HLY	1.5	65.1	4.6
	2.80kg-T-MOTOR	2.5	79.9	4.6
	3.00kg-T-MOTOR	2.2	112.0	7.6
	3.50kg-SunnySky	1.4	116.2	9.5
	3.80kg-T-MOTOR	1.4	63.6	4.1
	4.00kg-SunnySky	1.9	83.7	5.0
	4.50kg-T-MOTOR	1.8	95.7	6.8
	4.91kg-DJI	2.5	69.6	6.9
	5.45kg-JFRC	2.6	75.8	3.3
Virtual Vehicles	0.55kg-Quadrotor 1	3.6	1383.7	69.2
	0.68kg-Agile Autonomy DIY [17]	3.0	171.4	17.4
	0.75kg-Quadrotor 2	4.2	1467.0	73.3
	0.85kg-Quadrotor 3	3.2	1052.7	52.6
	0.98kg-EGO Planner DIY [16]	4.6	1083.3	57.7
	1.05kg-Quadrotor 4	3.5	950.7	47.5
	1.20kg-Quadrotor 5	4.0	1164.8	58.2
	1.50kg-Quadrotor 6	3.8	931.1	46.5
	1.80kg-Quadrotor 7	3.8	969.3	48.5
	2.00kg-Quadrotor 8	3.2	712.9	35.6
	2.50kg-Quadrotor 9	3.0	692.8	34.6
	2.80kg-Quadrotor 10	3.4	694.5	34.7
	3.00kg-Quadrotor 11	3.3	697.4	34.9
	3.50kg-Quadrotor 12	3.1	584.4	29.2
	4.20kg-Quadrotor 13	2.8	553.4	27.7
	4.50kg-Quadrotor 14	2.9	507.8	25.4
	4.80kg-Quadrotor 15	3.6	587.3	29.4
	5.00kg-Quadrotor 16	3.5	631.0	31.5

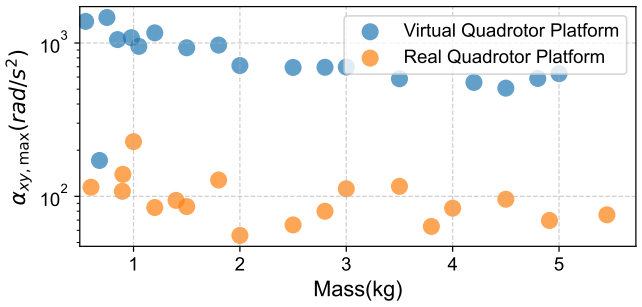
B. Navigation scenarios

A robust navigation benchmark must be comprehensive. It cannot rely on a single environment type but must instead provide a diverse suite of scenarios designed to systematically probe an algorithm's full performance envelope. The FLYINGTRUST scenario library is built on this principle. It is not merely a collection of maps, but a curated and complementary set of challenges that span the spectrum from common, real-world-like conditions to adversarial stress tests. This structure ensures that algorithms are evaluated not only for baseline proficiency but also for their breaking points under specific, isolated challenges. The dataset is provided in .PCD, .PLY, and Gazebo-compatible .dae formats, seamlessly integrating with existing quadrotor simulation platforms. Note that at the time of this study the Random Perlin-Noise scenario has not been successfully migrated to Gazebo; where this scenario is absent we mark the affected algorithm entries and apply the missing-data policy described in Section III-C.

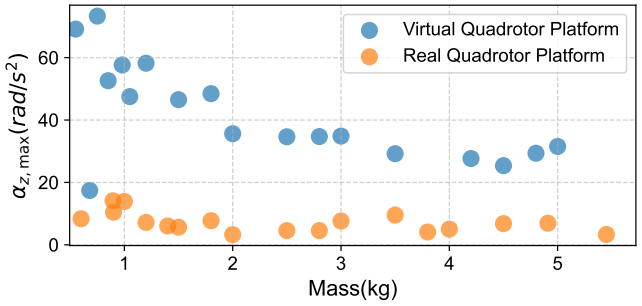
1) Classic Navigation Scenarios: As illustrated in Figs. 8a to 8c, the "Classic" scenarios are designed to measure baseline



(a) $\mathrm{TWR}_{\max}$ vs mass



(b) $\alpha_{xy,\max}$ vs mass



(c) $\alpha_{z,\max}$ vs mass

Fig. 7. **Quadrotor kinodynamic properties in FLYINGTRUST.** Scatter plots showing the relationship between platform mass and the three kinodynamic metrics: (a) maximum thrust-to-weight ratio ($\mathrm{TWR}_{\max}$), (b) maximum horizontal angular acceleration ($\alpha_{xy,\max}$, log scale), and (c) maximum yaw angular acceleration ($\alpha_{z,\max}$). Virtual quadrotors are marked in blue; real quadrotors are marked in orange.

proficiency and real-world applicability. They represent the most common environments found in navigation research, characterized by unstructured obstacle (Forest), structured obstacle (Urban), and 3D geometric variability (Random-Angle Cylinder). We expect high-performing, mature algorithms to succeed consistently here, establishing their suitability for typical field deployment.

- **Forest**: FLYINGTRUST leverages Unity [39], [40] to create a canonical forest scenario measuring 40 m in width and 60 m in length, with a drone flight ceiling of 3 m. Using a tree density of $\delta = 1/49$, we randomly generated ten distinct obstacle configurations to form the forest navigation dataset. These scenarios are subsequently reconstructed and validated in the Gazebo simulator [41]. Such settings are widely adopted in quadrotor studies to benchmark perception and obstacle avoidance in unstructured outdoor environments [16], [17]. The primary challenge lies in reliably detecting narrow gaps between trees while maintaining high-speed maneuvering.
- **Urban**: FLYINGTRUST uses Unity to create a canonical urban environment measuring 60 m in width and 60 m in length, with a drone flight ceiling of 10 m. Ten obstacle configurations were randomly sampled from the Unity urban scenario project to form the urban navigation dataset, which was then reconstructed and validated in the Gazebo simulator. The urban scenario introduces structured but cluttered geometry with vertical obstacles resembling buildings and walls. Compared to forests, urban layouts pose challenges for visual localization (e.g., repeated textures, sharp corners) and can induce failure modes such as drift or premature collision.
- **Random-Angle Cylinder**: FLYINGTRUST builds a randomized, tilted-cylinder obstacle field in Unity, covering an area 40 m wide by 60 m long with a flight ceiling of 3 m. Using a cylinder density of $\delta = 1/36$, radii ranging from 0.25 m to 0.5 m, and tilt angles between $0°$ and $180°$, ten distinct obstacle configurations were generated to constitute the randomly oriented cylinder obstacle dataset. These scenarios are subsequently reconstructed and validated in Gazebo. Randomly tilted cylinders create unpredictable obstacle orientations, breaking the symmetry of standard upright obstacles. This design challenges quadrotor algorithms to handle highly variable geometry and perform adaptive lateral maneuvers [42]. Success in this environment indicates robustness to non-canonical obstacle distributions.

2) Theoretical Navigation Scenarios: As illustrated in Figs. 8d to 8g, the "Theoretical" scenarios are adversarial stress tests designed to find an algorithm's specific breaking point. Crafted from real-world failure cases, each scenario is a diagnostic tool that targets a critical axis of performance, making the benchmark suite complete: "Narrow-Gap" probes yaw agility and perception in confined spaces; "Sudden-Drop" tests pitch authority and vertical control in response to sharp terrain changes; "Maze" challenges global replanning logic and memory; and "Random Perlin-Noise" pushes perception and local planning to their absolute limits in extreme clutter. Given their targeted difficulty, we anticipate lower success rates, which serve to highlight specific algorithmic weaknesses.

- **Narrow-Gap**: FLYINGTRUST constructs a 50 m by 50 m narrow-gap environment in Unity with a flight ceiling of 4 m. Gap widths range from 0.85 m to 0.9 m. Ten obstacle configurations are generated at random, with the number of gaps increasing from Scenario 1 to Scenario 10, forming the narrow-gap dataset. These scenarios are reconstructed in Gazebo. With openings as narrow as 0.85 m, this environment explicitly tests a quadrotor's capability to execute large-yaw maneuvers for lateral gap navigation. Similar settings are often used in agile flight benchmarks, where insufficient yaw agility

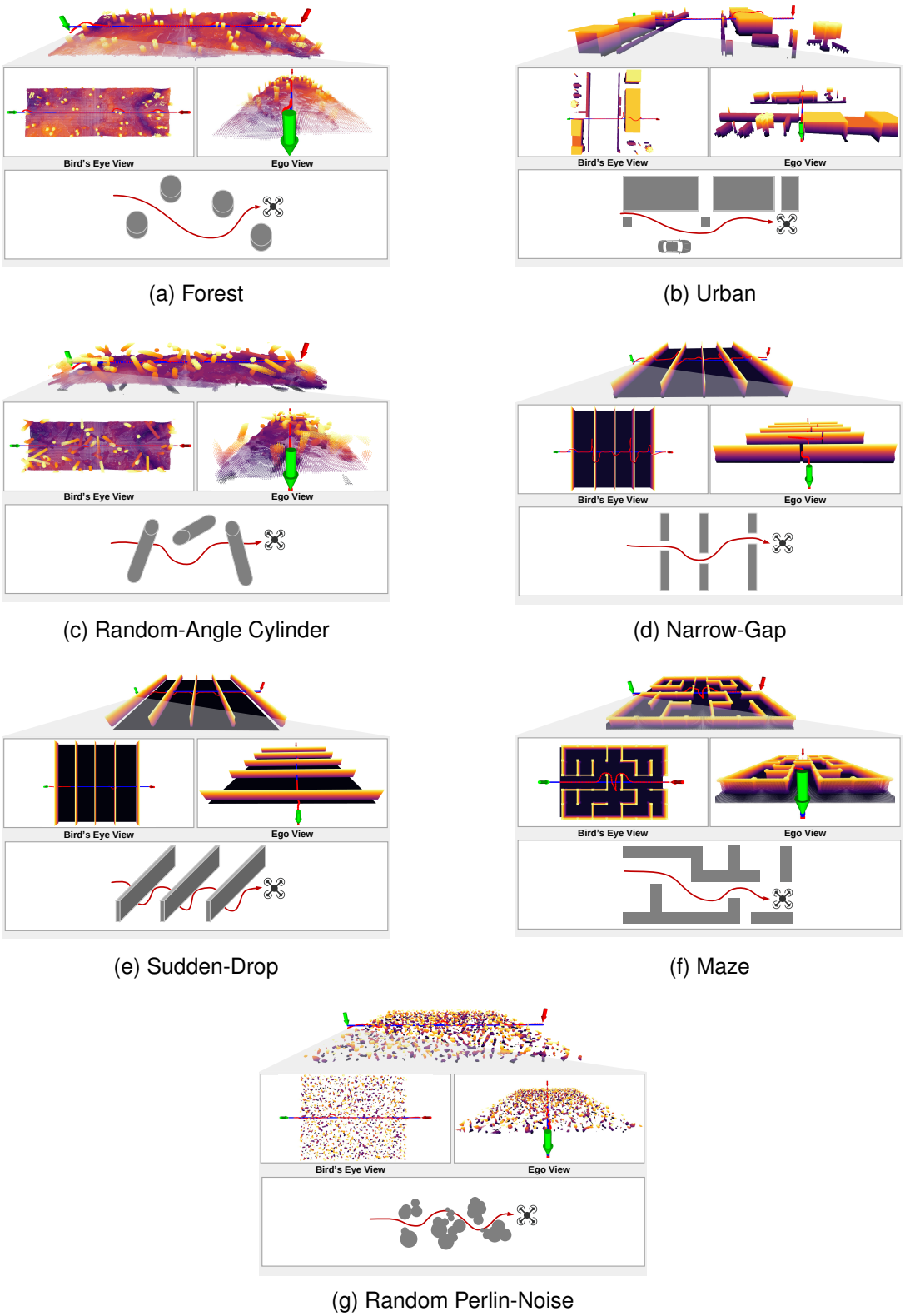


Fig. 8. **Navigation scenarios used in FLYINGTRUST.** The seven scenarios are grouped into (a-c) Classic Navigation Scenarios (Forest, Urban, Random-Angle Cylinder) and (d-g) Theoretical Navigation Scenarios (Narrow-Gap, Sudden-Drop, Maze, Random Perlin-Noise). In each subplot, the green arrow marks the start pose, the red arrow marks the goal, the blue line is the straight-line reference path, and the red curve is an example collision-free trajectory.

or inaccurate perception leads to frequent crashes [43].

- **Sudden-Drop**: FLYINGTRUST creates a sudden-drop environment in Unity measuring 50 m by 50 m by 14 m, where the lowest obstacle point is 1.5 m above ground. The quadrotor starts at 2.5 m altitude with a ceiling of 4 m. Ten random obstacle configurations are generated,

with obstacle count rising from Scenario 1 to Scenario 10, forming the sudden-drop dataset. These scenarios are reproduced in Gazebo to evaluate whether an algorithm can handle rapid pitch adjustments and altitude control when encountering sharp terrain discontinuities. Quadrotors with limited pitch authority often fail by colliding with obstacles after delayed descent response [44].

- **Maze**: FLYINGTRUST uses a maze generator in Unity to design a $25\ \mathrm{m}$ by $40\ \mathrm{m}$ labyrinth, limiting flight height to $2\ \mathrm{m}$. Ten random maze layouts form the maze dataset, which is reconstructed in Gazebo. The maze environment demands deliberate exploration and decision-making under partial observability. It reflects real-world scenarios such as indoor navigation or search-and-rescue, where dead ends force replanning and robust memory integration.
- **Random Perlin-Noise**: FLYINGTRUST designs a $40\ \mathrm{m}$ by $50\ \mathrm{m}$ random Perlin noise environment with a flight ceiling of $4\ \mathrm{m}$, using a noise frequency of 0.05 and voxel fill rate of 0.03. Ten random obstacle configurations constitute the Perlin noise dataset. Migrating this scenario to Gazebo proved challenging, and a reconstruction has not yet been achieved. By generating highly irregular obstacle fields through Perlin noise, this scenario produces extreme clutter and visual aliasing. It pushes algorithms to their limits in perception, mapping, and global planning.

C. Scoring Method

After conducting a series of navigation success rate tests, we further compare and rank different visual navigation algorithms by introducing a composite scoring system based on scenario and platform weighting. This method draws inspiration from composite performance indices and multi-criteria decision making (MCDM) [45], and integrates three dimensions: scenario importance, platform importance, and algorithmic stability.

The overall idea is as follows: For each algorithm, the navigation success rates $S_{a,s,m}$ across different scenarios and quadrotor platforms are aggregated using weighted averages, where the weights encode scenario and platform importance. To account for consistency, a variance-based penalty is then applied to discourage algorithms whose performance fluctuates strongly across conditions [46]. The final score $\mathrm{FinalScore}_a$ serves as an intuitive "benchmark score" for comparison.

Scenario weights: Each scenario type (classic vs. theoretical) is assigned an initial weight $w_s (s \in \mathcal{S})$. After normalization,

$$W_s = \frac{w_s}{\sum_{s'} w_{s'}} \tag{12}$$

Platform weights: Each quadrotor platform type (real vs. virtual) is assigned a weight $w_m (m \in \mathcal{M})$. After normalization,

$$W_m = \frac{w_m}{\sum_{m'} w_{m'}} \tag{13}$$

Initial score: The weighted mean success rate is computed as

$$\widehat{\mathrm{Score}}_a = \frac{\sum_s \sum_m W_s W_m S_{a,s,m}}{\sum_s \sum_m W_s W_m}, \tag{14}$$

which is scaled to percentage form:

$$\mathrm{Score}_a = 100 \times \widehat{\mathrm{Score}}_a \tag{15}$$

Stability penalty: To reflect consistency, the weighted variance is defined as

$$\mathrm{Var}_a = \sum_s \sum_m W_s W_m (S_{a,s,m} - \widehat{\mathrm{Score}}_a)^2, \tag{16}$$

and normalized as

$$\mathrm{Var}_a^{\mathrm{norm}} = \frac{\mathrm{Var}_a}{\max_{a'} \mathrm{Var}_{a'}} \in [0,1] \tag{17}$$

Final score: Incorporating the stability penalty, the final score is

$$\mathrm{FinalScore}_a = \mathrm{Score}_a \times (1 - \beta \cdot \mathrm{Var}_a^{\mathrm{norm}}), \tag{18}$$

where $\beta \in [0,1]$ controls the penalty strength. $\beta = 0$ yields a purely average-based score, while $\beta > 0$ emphasizes algorithms with more stable performance across heterogeneous conditions.

For algorithms that lack results for an entire scenario s_0, we exclude s_0 from the summation and renormalize the remaining scenario weights for that algorithm:

$$W_s' = \frac{W_s}{\sum_{s \in \mathcal{S} \setminus \{s_0\}} W_s}, \tag{19}$$

where W_s denotes the original normalized scenario weight and W_s' denotes the renormalized weight after exclusion.

IV. Experiments Evaluation and Analysis

A. Algorithms

We evaluated several commonly used and widely deployed visual quadrotor navigation algorithms, including:

- **EGO-Planner** [16]: An ESDF-free local trajectory optimizer that relies on guided paths and anisotropic curve fitting to achieve collision avoidance and smooth trajectories.
- **Fast-Planner** [15]: A two-stage approach that first searches for a dynamically feasible path using discrete motion primitives, then refines it via B-spline optimization to enhance geometric quality and ensure dynamic feasibility, striking a balance between real-time performance and flight quality.
- **Path-Guided PGO** [19]: A replanning method that samples diverse topological paths to escape local minima, using these routes as priors to guide the optimizer toward higher-quality trajectories in complex scenarios.
- **NavRL** [18]: A deep reinforcement learning framework employing PPO and specialized perception modules for static and dynamic obstacles, augmented by a velocity-obstacle safety shield to enable zero-shot sim-to-real transfer and robust obstacle avoidance in dynamic environments. Testing employed the authors' publicly released pre-trained model, obtained from their GitHub repository [47], without further fine tuning.
- **Agilicious** [48]: An open-source hardware and software project that provides a modular stack for agile vision-based quadrotor flight, including high-rate control loops,

perception interfaces and a simulation ecosystem for development and real-world deployment. Agilicious emphasizes low-level control fidelity and real-time responsiveness, and it is well suited to research on aggressive maneuvers and high-agility platforms.
- **Agile-Autonomy** [17]: A learning-based approach that targets robust, high-speed navigation across diverse real-world environments by combining data-driven policy training with careful sim-to-real engineering. The method demonstrates strong performance on agile flight tasks and is particularly relevant for high-velocity, perception-driven autonomy.
- **Straight-Flight Baseline**: A constant-velocity direct path from start to goal without obstacle avoidance, used as a baseline to quantify the benefit of active planning methods.

Agilicious and Agile-Autonomy are not evaluated here because their simulation and software ecosystems are tightly coupled to specific vehicle configurations, complicating integration with the FLYINGTRUST scenes. Incorporating these systems in a fair and reproducible way would require extensive engineering adaptation beyond the intended scope of this study.

It is important to note that our evaluation compares multiple optimization-based planners against a single, representative learning-based algorithm (NavRL), using its publicly available pre-trained model. Consequently, the observed behaviors and limitations of NavRL may be specific to its architecture and training data, and should not be interpreted as fundamental properties of all learning-based navigation approaches.

For fair comparison, all methods (including the straight-flight baseline) use the same start pose. All runs respect identical flight ceilings, maximum speed limits and acceleration constraints.

B. Benchmarking results

To evaluate a set of representative visual quadrotor navigation algorithms, FLYINGTRUST conducted a series of simulations. In these tests, the quadrotor's maximum flight speed was capped at $4\ \mathrm{m/s}$. The task consisted of point-to-point navigation: The quadrotor first flew from its takeoff position to a designated start waypoint and then followed the algorithm's planned path to the goal. A straight line connecting start and goal served as the reference trajectory, with obstacles placed along it. Success was defined by the quadrotor reaching the goal within $1.5\ \mathrm{minutes}$ under the $4\ \mathrm{m/s}$ speed limit and remaining stably within a $2\ \mathrm{m}$ radius of the target. Failure was recorded if the quadrotor collided with any obstacle, if its final stable position lay outside the $2\ \mathrm{m}$ radius, or if it failed to plan a viable path to the goal within $1.5\ \mathrm{minutes}$.

Fig. 9 summarizes per-scenario mean navigation success rates for all methods together with 95% bootstrap confidence intervals computed from $B = 1000$ resamples; each (algorithm, scenario, platform) combination is evaluated with 10 independent trials. Building on the scenario overview, Fig. 10 reports algorithm performance aggregated across the heterogeneous set of quadrotor platforms to expose sensitivity to platform capability. Fig. 11 complements this view by showing per-environment performance averaged across all platforms. For a representative, fine-grained behavior analysis, Figs. 12 to 16 provide per-algorithm heatmaps for the subset of platforms weighing between 1kg and 2kg. The complete heatmaps for all 36 platforms are available in the appendix. Confidence intervals follow the bootstrap procedure of Efron and Tibshirani [49]. We discuss the results in a Q&A format.

Q1: Which method performs best in each navigation scenario?

A1: Fig. 9 summarizes the per-scenario mean success rates across all methods. Figs. 12 to 16 provide a more detailed look at these trends for a representative subset of platforms. Based on the overall averages in Fig. 9, we compare methods scenario by scenario.

- **Open fields (Forest, Urban):** In the relatively open Forest and Urban scenarios with sparse obstacles, all algorithms perform well. Fast-Planner and NavRL both exceed 80% average success in the Forest scenario, with narrow confidence intervals indicating excellent stability and robustness. EGO-Planner and PGO lag slightly behind but still outperform the straight-flight baseline.
- **Random-Angle Cylinder:** In the Random-Angle Cylinder scenario, obstacles are randomly oriented. NavRL leads with over 75% average success and a tight confidence interval, Fast-Planner maintains above 55%, while EGO-Planner and PGO drop to around 35%. The straight-flight baseline fails entirely.
- **Narrow-Gap:** The Narrow-Gap scenario demands aggressive yaw maneuvers in confined spaces, resulting in generally low success rates. EGO-Planner slightly outperforms the baseline, NavRL matches the baseline, and Fast-Planner along with PGO perform worse than the baseline, indicating shortcomings in fine attitude control.
- **Sudden-Drop:** The Sudden-Drop scenario tests vertical pitch maneuverability. EGO-Planner performs best at 55% average success, followed by Fast-Planner at 38%. PGO and NavRL both fall below 5%, and the straight-flight baseline fails entirely, highlighting their lack of adaptation to pitch-intensive maneuvers.
- **Maze:** The Maze scenario evaluates exploration and path-finding skills. EGO-Planner, Fast-Planner, and PGO each achieve around 15% average success with similar confidence intervals, NavRL falls below 10%, and the straight-flight baseline fails. This indicates that complex labyrinth tasks challenge all tested methods.
- **Random Perlin-Noise:** In the most complex Random Perlin-Noise scenario, only classical planners are evaluated. EGO-Planner leads with 28% success, Fast-Planner achieves 15%, PGO drops to 4%, and the straight-flight baseline fails entirely, illustrating differences in local mapping and curve optimization efficacy under extreme randomness.

Q2: How do quadrotor kinodynamic properties affect the success rates of visual navigation algorithms, and do the chosen indicators form a reasonable set of kinodynamic descriptors?

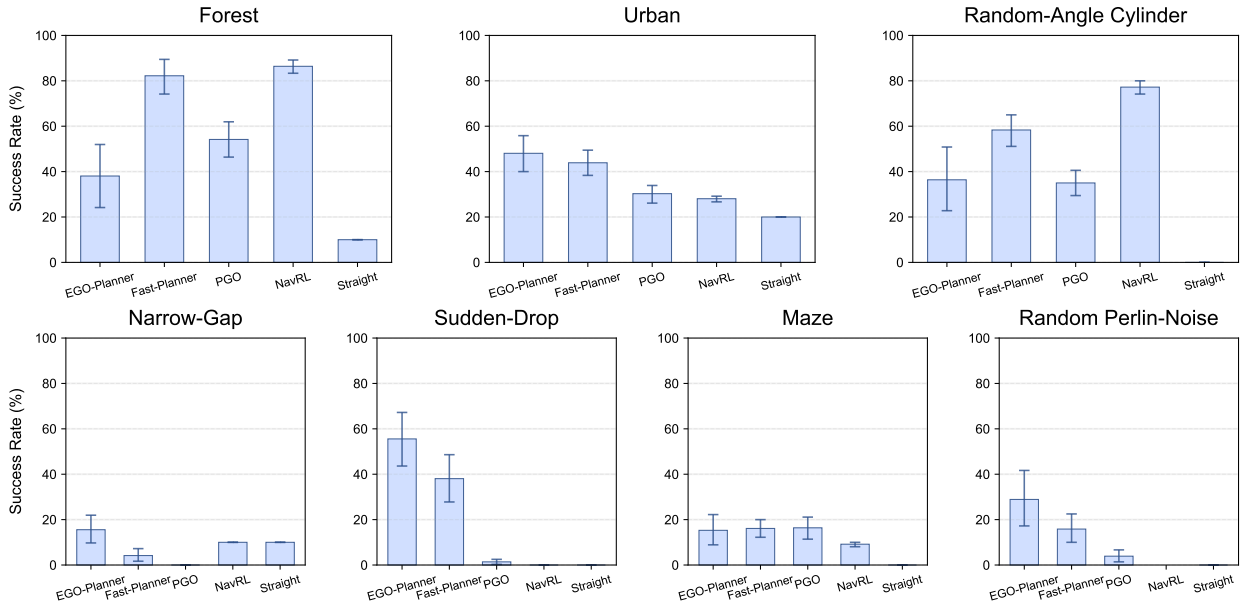
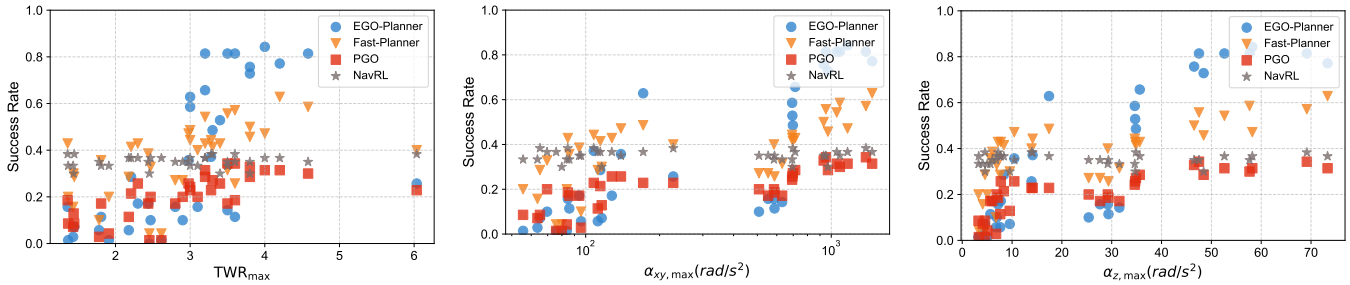


Fig. 9. **Benchmark results by navigation scenario.** The mean navigation success rate for each algorithm, averaged across all 36 quadrotor platforms, is shown for each of the seven scenarios. Error bars indicate 95% bootstrap confidence intervals.



(a) Success rate vs $\mathrm{TWR}_{\max}$ (b) Success rate vs $\alpha_{xy,\max}$ (c) Success rate vs $\alpha_{z,\max}$

Fig. 10. **Per-algorithm success rate vs. kinodynamic capability.** Panels (a)-(c) plot each algorithm's success rate against the three kinodynamic indicators: (a) $\mathrm{TWR}_{\max}$, (b) $\alpha_{xy,\max}$ (log scale), and (c) $\alpha_{z,\max}$. Each marker represents the mean success rate for a single platform (averaged across all scenarios), with marker color indicating the algorithm as shown in the legend.

A2: Conventional optimization-based methods (EGO-Planner, Fast-Planner) tend to achieve higher success rates on platforms with larger $\mathrm{TWR}_{\max}$ and greater $\alpha_{xy,\max}$ (see Fig. 10a and Fig. 10b). Trajectory optimizers produce aggressive, time-critical maneuvers that require both ample translational acceleration and fast reorientation of thrust (roll/pitch agility); platforms lacking these capabilities cannot reliably execute those trajectories. NavRL shows a comparatively flat response across these kinodynamic dimensions in our tests, suggesting either the learned policy cannot exploit higher agility or the training distribution did not emphasize high-agility regimes. $\alpha_{z,\max}$ plays a role in scenarios that require large heading changes, but its overall correlation with success is weaker than that of $\alpha_{xy,\max}$ (see Fig. 10c).

Taken together, $\mathrm{TWR}_{\max}$, $\alpha_{xy,\max}$ and $\alpha_{z,\max}$ capture, respectively, translational power, thrust-direction reorientation capability and yaw responsiveness. As shown in Fig. 10, these three indicators form a compact and physically interpretable set that meaningfully constrains feasible navigation behaviors for quadrotors.

Q3: Which method is most effective across different quadrotor platform?

A3: Fig. 11 summarizes per-platform average success rates grouped by platform category (virtual vs. real and by performance tier).

- **Virtual quadrotor Platforms:** On high-performance virtual platforms, EGO-Planner achieves the highest average success rate, followed by Fast-Planner; PGO and NavRL perform similarly and both clearly outperform the straight-flight baseline. However, on low-performance virtual platforms, the average success rates of EGO-Planner, Fast-Planner, and PGO drop sharply and are no longer significantly better than the baseline. Only NavRL maintains a relatively stable success rate, indicating greater resilience to degraded kinodynamic capabilities.
- **Real quadrotor Platforms:** In real platform tests, NavRL delivers the highest and most consistent average success rates, remaining virtually unaffected by drops in platform

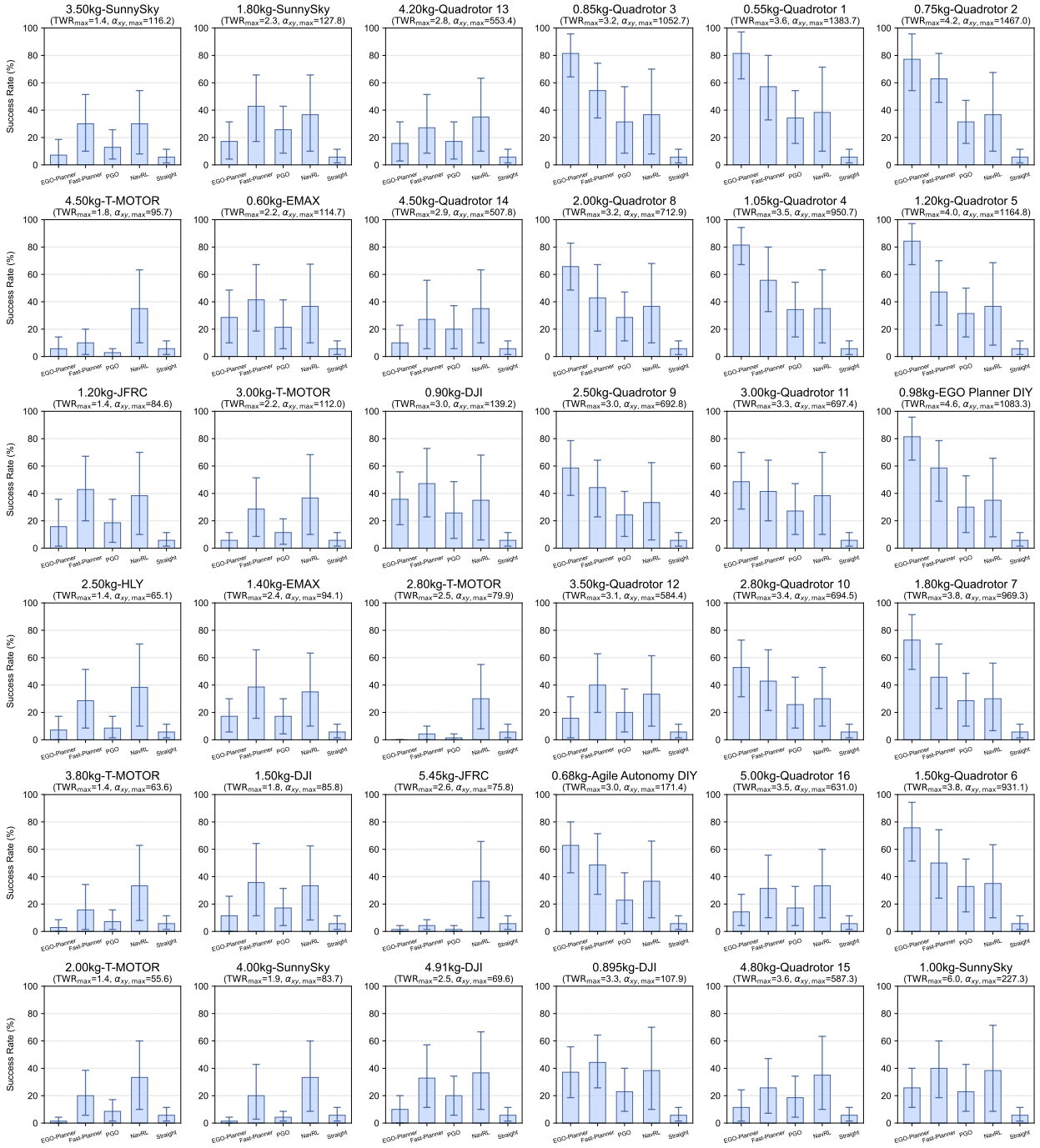


Fig. 11. **Benchmark results by quadrotor kinodynamic profile.** This figure shows the mean navigation success rate for each method across all 36 quadrotor platforms. The 36 subplots are strategically arranged in a 2D grid based on platform kinodynamics: horizontal position (left-to-right) corresponds to increasing $\mathrm{TWR}_{\max}$, and vertical position (bottom-to-top) corresponds to increasing $\alpha_{xy,\max}$. Each individual bar represents a method's average success rate across all seven test scenarios for that specific platform. Error bars indicate 95% bootstrap confidence intervals.

performance. In contrast, EGO-Planner, Fast-Planner, and PGO not only achieve lower success rates overall but also suffer marked declines as platform performance decreases, performing comparably to the straight-flight baseline on low-end real platforms.

Q4: How does the straight-flight baseline differ from active planners?

A4: The straight-flight baseline flies at constant speed along the direct line from start to goal without collision avoidance. As shown in Fig. 9, it attains mean success rates of approximately 10-20% in the open Forest and Urban scenarios, but fails in cluttered environments.

Q5: Overall, which algorithm performs best?

A5: Using the scoring method described in Section III-C, we computed a composite score for each visual quadrotor navigation algorithms. Weight choices were set as follows.

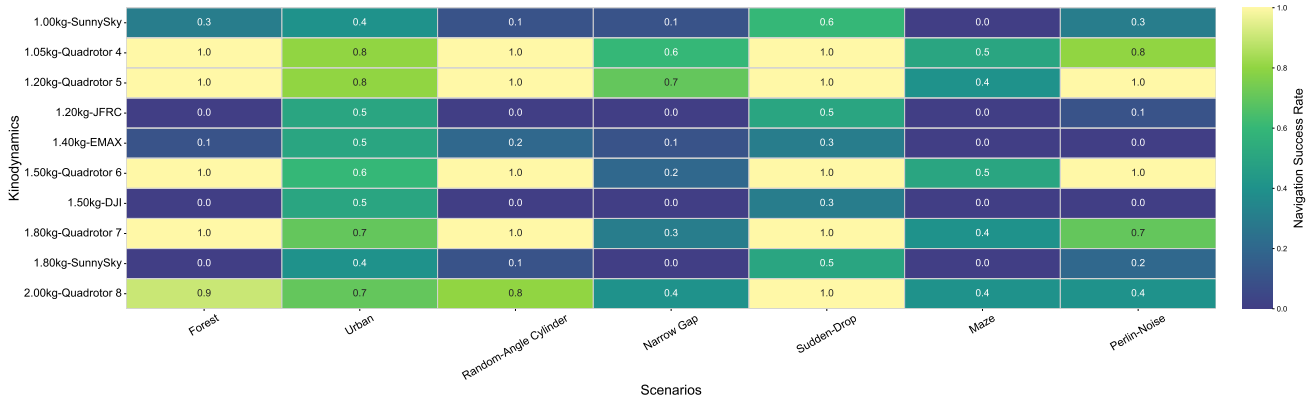


Fig. 12. **EGO-Planner navigation success rate (1kg-2kg platforms).** The heatmap shows the mean success rate for each platform (y-axis) across all seven scenarios (x-axis). Success rates are color-coded from 0.0 (dark purple) to 1.0 (bright yellow).

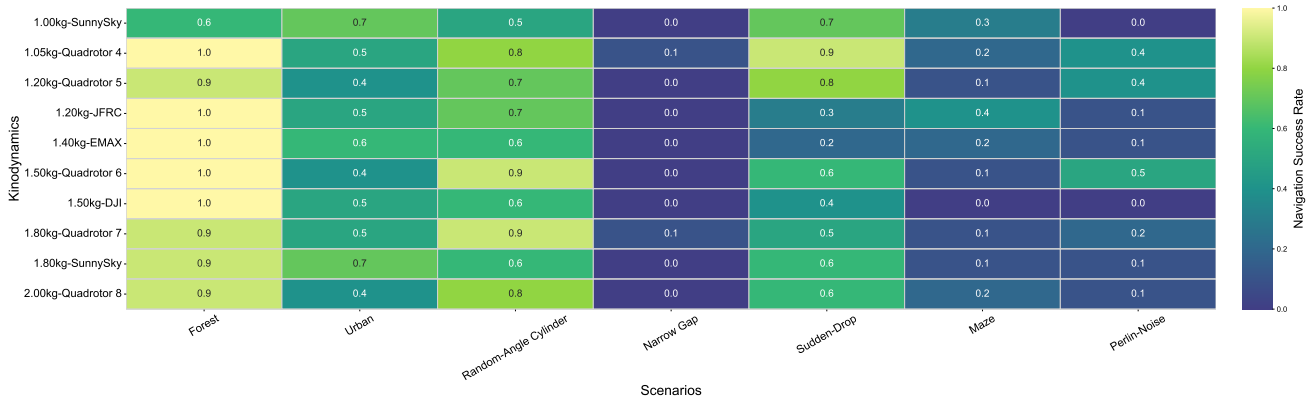


Fig. 13. **Fast-Planner navigation success rate (1kg-2kg platforms).** The heatmap shows the mean success rate for a subset of platforms (y-axis) weighing between 1kg and 2kg, across all seven scenarios (x-axis). Success rates are color-coded from 0.0 (dark purple) to 1.0 (bright yellow).

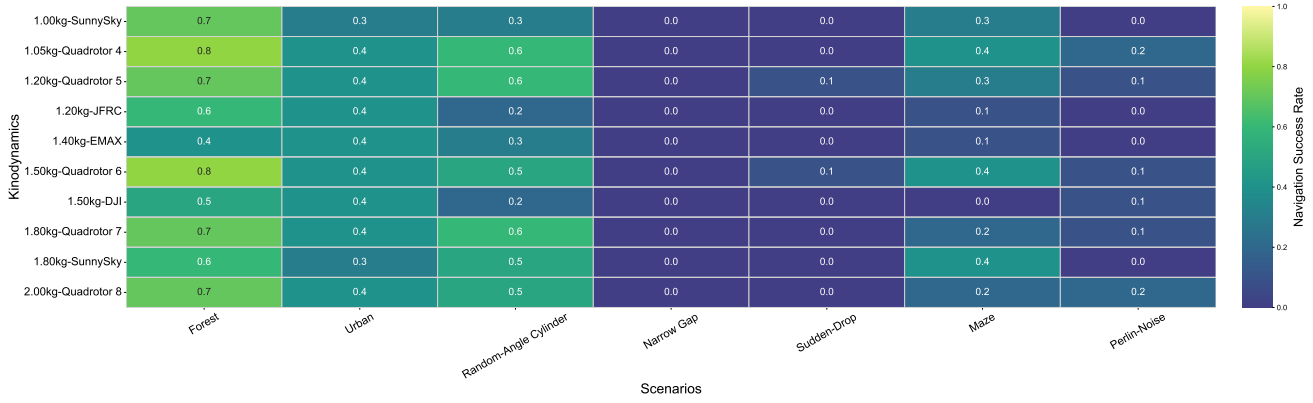


Fig. 14. **PGO navigation success rate (1kg-2kg platforms).** The heatmap shows the mean success rate for a subset of platforms (y-axis) weighing between 1kg and 2kg, across all seven scenarios (x-axis). Success rates are color-coded from 0.0 (dark purple) to 1.0 (bright yellow).

Classic navigation scenarios were assigned an initial weight of 1.2, while theoretical scenarios were assigned 1.0. For platform weights, real quadrotor platforms were assigned 1.5 and virtual platforms 1.0. The stability penalty coefficient was set to $\beta = 0.3$.

The computed scores appear in Table II, As noted, †NavRL has no valid results for the Perlin-Noise scenario. The NavRL score shown here is computed after excluding that scenario and renormalizing the remaining scenario weights (see Section III-C). Because the effective evaluation set differs, NavRL's score is provided as a reference only and is not directly comparable to scores computed over the full scenario set.

Overall, Fast-Planner attains the highest composite score, balancing high success rates with relatively low variance across scenarios and platforms. The ranking from highest

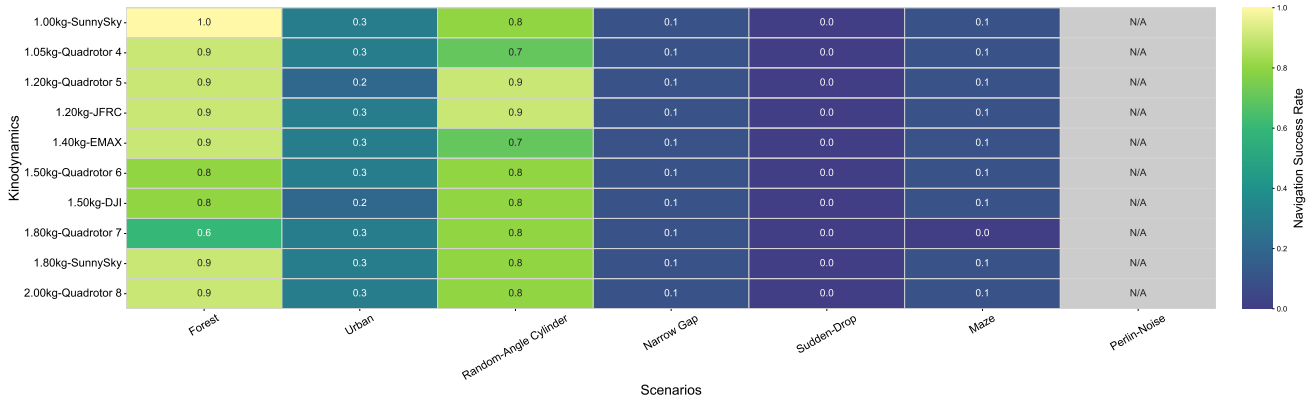


Fig. 15. **NavRL navigation success rate (1kg-2kg platforms).** The heatmap shows the mean success rate for a subset of platforms (y-axis) weighing between 1kg and 2kg, across all seven scenarios (x-axis). Success rates are color-coded from 0.0 (dark purple) to 1.0 (bright yellow).

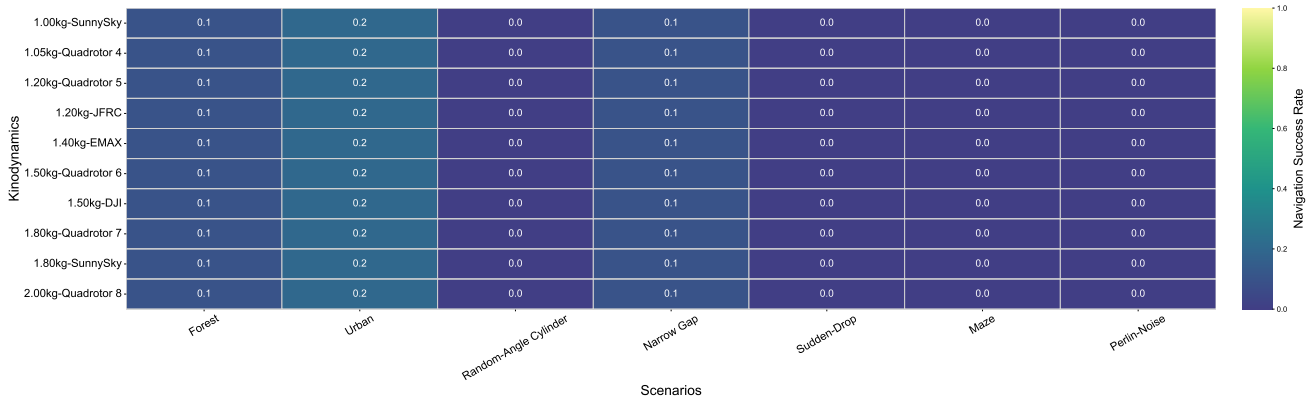


Fig. 16. **Straight-Flight Baseline navigation success rate (1kg-2kg platforms).** The heatmap shows the mean success rate for a subset of platforms (y-axis) weighing between 1kg and 2kg, across all seven scenarios (x-axis). Success rates are color-coded from 0.0 (dark purple) to 1.0 (bright yellow).

TABLE II
COMPOSITE SCORES, VARIANCES AND FINAL SCORES FOR EACH ALGORITHM.

Algorithm	Score (no penalty)	Variance	FinalScore	Missing scenarios
EGO-Planner	30.25	0.120	21.32	-
Fast-Planner	37.34	0.106	**27.58**	-
Path-Guided PGO	20.39	0.054	17.70	-
NavRL	37.78†	0.122	26.41†	Perlin-Noise
Straight Flight Baseline	6.05	0.005	5.97	-

to lowest score is: Fast-Planner, EGO-Planner, Path-Guided PGO, Straight Flight Baseline. NavRL's reference score is also relatively high, but its performance exhibits substantially larger variance, indicating weaker stability compared with the other methods.

Q6: What are the strengths and weaknesses of optimization-based and learning-based navigation methods?

A6: From the benchmarking of representative algorithms, we can distill the respective strengths and weaknesses of optimization-based and learning-based navigation methods:

Optimization-based methods (e.g., Fast-Planner, EGO-Planner, PGO) explicitly encode vehicle dynamics, safety, and efficiency constraints in trajectory planning, yielding several advantages:

- **High performance utilization:** These methods consistently reach the $4\ \mathrm{m/s}$ speed limit and generate dynamically feasible trajectories involving full 3D maneuvers, such as pitch and yaw. This enables superior performance in demanding scenarios like Sudden-Drop and Narrow-Gap.
- **Physically interpretable and generalizable:** The model-based design allows predictable, tunable behavior, facilitating deployment in real-world applications.
- **Strong controllability and stability:** Their reliance on analytical solvers makes them robust to runtime fluctuations in most structured scenarios.

However, limitations include:

- **Dependence on accurate models and sensors:** Any model mismatch or sensor noise can lead to suboptimal paths or even navigation failure.
- **Vulnerability to local minima:** Especially in random or unstructured scenarios, their gradient-based optimization may struggle to escape poor initial paths.

Learning-based methods, as represented by NavRL in our study, offer different trade-offs. While these findings highlight potential patterns in data-driven approaches, it is crucial to recognize that they are based on a single pre-trained model and may not generalize to all learning-based systems.

- **Platform-agnostic performance:** NavRL maintains sta-

ble success rates even on low-performance drones, highlighting its resilience across kinodynamic variations.
- **Strong sim-to-real transfer:** Without retraining or parameter tuning, NavRL is able to navigate successfully in the Gazebo simulator using the pre-trained policy, highlighting its practical transfer potential and limited generalization to familiar settings.

However, we also identified key drawbacks that may stem from its specific implementation and training:

- **Underutilization of flight capability:** NavRL consistently capped at $1\ \mathrm{m/s}$ despite the $4\ \mathrm{m/s}$ limit, indicating poor dynamic exploitation.
- **Behavioral bias from training data:** The policy exhibits only right-turn obstacle avoidance and lacks pitch maneuvers, likely due to skewed training data lacking diverse motion examples. This causes complete failure in tests like Narrow-Gap and Sudden-Drop.

V. Broader Impact and Limitations

Although FLYINGTRUST provides a comprehensive framework, its current implementation has limitations that present clear paths for future extensions. The benchmark currently fixes control-loop parameters (e.g., PID/MPC gains) and does not measure how low-level controller tuning affects higher-level planning performance. Future work should: (1) incorporate control-parameter sensitivity by systematically varying gains to quantify their impact; (2) model actuator and sensor uncertainties, integrating noise models and time delays; and (3) expand scenario diversity with dynamic obstacles and lighting variations to further stress-test perception and planning modules.

Our comparative analysis uncovers clear trade-offs between model-based optimization and data-driven learning. To advance visual quadrotor navigation, we recommend three complementary research directions: (1) hybrid planning and learning to combine principled constraints with data-driven adaptability; (2) kinodynamic integration by embedding platform-specific models into learning objectives; and (3) a strong focus on generalization by training on heterogeneous vehicle profiles and diverse, procedurally generated environments.

More broadly, this paper provides not just a set of results, but a complete, open-source framework and workflow for the systematic evaluation of quadrotor navigation algorithms. We have open-sourced FLYINGTRUST with the hope that it will serve as a foundational tool for the robotics community. We strongly encourage and invite researchers to contribute to its improvement, for instance by:

- Proposing and adding new, challenging navigation scenarios;
- Contributing new real or virtual quadrotor platform profiles;
- Benchmarking their own navigation algorithms against the established baselines;
- Conducting and sharing real-world validation tests to further correlate simulation-first findings with physical deployment.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under Grant No.T2350005 and the Fundamental Research Funds for the Central Universities, Sun Yat-sen University under Grant No.23xkjc008.

References

[1] S. P. Bharati, Y. Wu, Y. Sui, C. Padgett, and G. Wang, "Real-time obstacle detection and tracking for sense-and-avoid mechanism in UAVs," *IEEE Transactions on Intelligent Vehicles*, vol. 3, no. 2, pp. 185–197, 2018.

[2] R. Amin, L. Aijun, and S. Shamshirband, "A review of quadrotor UAV: control methodologies and performance evaluation," *International Journal of Automation and Control*, vol. 10, no. 2, pp. 87–103, 2016.

[3] F. Ahmed, J. C. Mohanta, A. Keshari, and P. S. Yadav, "Recent advances in unmanned aerial vehicles: a review," *Arabian Journal for Science and Engineering*, vol. 47, no. 7, pp. 7963–7984, 2022.

[4] P. Kim, J. Chen, J. Kim, and Y. K. Cho, "SLAM-driven intelligent autonomous mobile robot navigation for construction applications," in *Workshop of the European Group for Intelligent Computing in Engineering*, 2018, pp. 254–269.

[5] G. Kahn, A. Villaflor, B. Ding, P. Abbeel, and S. Levine, "Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation," in *IEEE Int. Conf. Robot. Autom.*, 2018, pp. 5129–5136.

[6] Q. Luo and H. Duan, "Distributed UAV flocking control based on homing pigeon hierarchical strategies," *Aerospace Science and Technology*, vol. 70, pp. 257–264, 2017.

[7] C. Liu, X. Zhao, Y. Du, C. Cao, Z. Zhu, and E. Mao, "Research on static path planning method of small obstacles for automatic navigation of agricultural machinery," *IFAC-PapersOnLine*, vol. 51, no. 17, pp. 673–677, 2018.

[8] P. Marin-Plaza, A. Hussein, D. Martin, and A. de la Escalera, "Global and local path planning study in a ROS-based research platform for autonomous vehicles," *Journal of Advanced Transportation*, vol. 2018, no. 1, p. 6392697, 2018.

[9] J.-R. Ruiz-Sarmiento, C. Galindo, and J. Gonzalez-Jimenez, "Scene object recognition for mobile robots through semantic knowledge and probabilistic graphical models," *Expert Systems with Applications*, vol. 42, no. 22, pp. 8805–8816, 2015.

[10] A. Harrison and P. Newman, "High quality 3D laser ranging under general vehicle motion," in *IEEE Int. Conf. Robot. Autom.*, 2008, pp. 7–12.

[11] J. Zhang, J. T. Springenberg, J. Boedecker, and W. Burgard, "Deep reinforcement learning with successor features for navigation across similar environments," in *IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 2371–2378.

[12] B. Huang, J. Li, J. Chen, G. Wang, J. Zhao, and T. Xu, "Anti-UAV410: A Thermal Infrared Benchmark and Customized Scheme for Tracking Drones in the Wild," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 46, no. 5, pp. 2852–2865, 2024.

[13] C. Zhang, G. Huang, L. Liu, S. Huang, Y. Yang, X. Wan, S. Ge, and D. Tao, "WebUAV-3M: A Benchmark for Unveiling the Power of Million-Scale Deep UAV Tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 7, pp. 9186–9205, 2023.

[14] P. Zhu, L. Wen, D. Du, X. Bian, H. Fan, Q. Hu, and H. Ling, "Detection and Tracking Meet Drones Challenge," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 11, pp. 7380–7399, 2022.

[15] B. Zhou, F. Gao, L. Wang, C. Liu, and S. Shen, "Robust and efficient quadrotor trajectory generation for fast autonomous flight," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3529–3536, 2019.

[16] X. Zhou, Z. Wang, H. Ye, C. Xu, and F. Gao, "Ego-planner: An esdf-free gradient-based local planner for quadrotors," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 478–485, 2020.

[17] A. Loquercio, E. Kaufmann, R. Ranftl, M. Müller, V. Koltun, and D. Scaramuzza, "Learning high-speed flight in the wild," *Science Robotics*, vol. 6, no. 59, p. eabg5810, 2021.

[18] Z. Xu, X. Han, H. Shen, H. Jin, and K. Shimada, "NavRL: Learning safe flight in dynamic environments," *IEEE Robotics and Automation Letters*, 2025.

[19] B. Zhou, F. Gao, J. Pan, and S. Shen, "Robust real-time uav replanning using guided gradient-based optimization and topological paths," in *IEEE Int. Conf. Robot. Autom.*, 2020, pp. 1208–1214.

[20] D. Hanover, A. Loquercio, L. Bauersfeld, A. Romero, R. Penicka, Y. Song, G. Cioffi, E. Kaufmann, and D. Scaramuzza, "Autonomous drone racing: A survey," *IEEE Transactions on Robotics*, vol. 40, pp. 3044–3067, 2024.
[21] V. Usenko, L. V. Stumberg, A. Pangercic, and D. Cremers, "Real-time trajectory replanning for MAVs using uniform B-splines and a 3D circular buffer," in *IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 215–222.
[22] T. Lee, S. Mckeever, and J. Courtney, "Flying free: A research overview of deep learning in drone navigation autonomy," *Drones*, vol. 5, no. 2, p. 52, 2021.
[23] H. X. Pham, H. I. Ugurlu, J. Le Fevre, D. Bardakci, and E. Kayacan, "Deep learning for vision-based navigation in autonomous drone racing," in *Deep learning for robot perception and cognition*, 2022, pp. 371–406.
[24] P. Föhn, D. Brescianini, E. Kaufmann, T. Cieslewski, M. Gehrig, M. Muglikar, and D. Scaramuzza, "Alphapilot: Autonomous drone racing," *Autonomous Robots*, vol. 46, no. 1, pp. 307–320, 2022.
[25] E. Kaufmann, A. Loquercio, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza, "Deep drone racing: Learning agile flight in dynamic environments," in *Conference on Robot Learning*, 2018, pp. 133–145.
[26] W. Koch, R. Mancuso, R. West, and A. Bestavros, "Reinforcement learning for UAV attitude control," *ACM Transactions on Cyber-Physical Systems*, vol. 3, no. 2, pp. 1–21, 2019.
[27] N. O. Lambert, D. S. Drew, J. Yaconelli, S. Levine, R. Calandra, and K. S. J. Pister, "Low-level control of a quadrotor with deep model-based reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4224–4230, 2019.
[28] E. Kaufmann, L. Bauersfeld, A. Loquercio, M. Müller, V. Koltun, and D. Scaramuzza, "Champion-level drone racing using deep reinforcement learning," *Nature*, vol. 620, no. 7976, pp. 982–987, 2023.
[29] R. Penicka, Y. Song, E. Kaufmann, and D. Scaramuzza, "Learning minimum-time flight in cluttered environments," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 7209–7216, 2022.
[30] M. Müller, G. Li, V. Casser, N. Smith, D. L. Michels, and B. Ghanem, "Learning a controller fusion network by online trajectory filtering for vision-based UAV racing," in *IEEE Conf. Comput. Vis. Pattern Recog. Worksh.*, 2019, pp. 0–0.
[31] M. Müller, V. Casser, N. Smith, D. L. Michels, and B. Ghanem, "Teaching UAVs to race: End-to-end regression of agile controls in simulation," in *Eur. Conf. Comput. Vis.*, 2018, pp. 0–0.
[32] L. O. Rojas-Perez and J. Martinez-Carranza, "DeepPilot: A CNN for autonomous drone racing," *Sensors*, vol. 20, no. 16, p. 4524, 2020.
[33] M. Achtelik, A. Bachrach, R. He, S. Prentice, and N. Roy, "Autonomous navigation and exploration of a quadrotor helicopter in GPS-denied indoor environments," in *First symposium on indoor flight*, 2009.
[34] Q. Quan, *Introduction to multicopter design and control*, 2017, vol. 10.
[35] A. Machmudah, M. Shanmugavel, S. Parman, T. S. A. Manan, D. Dutykh, S. Beddu, and A. Rajabi, "Flight trajectories optimization of fixed-wing UAV by bank-turn mechanism," *Drones*, vol. 6, no. 3, p. 69, 2022.
[36] K. M. Ali and A. A. Jaber, "Comparing Dynamic Model and Flight Control of Plus and Cross Quadcopter Configurations," *FME Transactions*, vol. 50, no. 4, 2022.
[37] D. Shi, X. Dai, X. Zhang, and Q. Quan, "A practical performance evaluation method for electric multicopters," *IEEE/ASME Transactions on Mechatronics*, vol. 22, no. 3, pp. 1337–1348, 2017.
[38] D. Mellinger, N. Michael, and V. Kumar, "Trajectory generation and control for precise aggressive maneuvers with quadrotors," *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 664–674, 2012.
[39] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar *et al.*, "Unity: A general platform for intelligent agents," *arXiv preprint arXiv:1809.02627*, 2018.
[40] R. Riochet, M. Y. Castro, M. Bernard, A. Lerer, R. Fergus, V. Izard, and E. Dupoux, "IntPhys 2019: A Benchmark for Visual Intuitive Physics Understanding," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 9, pp. 5016–5026, 2022.
[41] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," in *IEEE/RSJ Int. Conf. Intell. Robots Syst.*, vol. 3, 2004, pp. 2149–2154.
[42] Y. Ren, F. Zhu, G. Lu, Y. Cai, L. Yin, F. Kong, J. Lin, N. Chen, and F. Zhang, "Safety-assured high-speed navigation for MAVs," *Science Robotics*, vol. 10, no. 98, p. eado6187, 2025.
[43] D. Falanga, K. Kleber, and D. Scaramuzza, "Dynamic obstacle avoidance for quadrotors with event cameras," *Science Robotics*, vol. 5, no. 40, p. eaaz9712, 2020.
[44] G. Torrente, E. Kaufmann, P. Föhn, and D. Scaramuzza, "Data-driven MPC for quadrotors," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3769–3776, 2021.
[45] E. Triantaphyllou, "Multi-criteria decision making methods," in *Multi-criteria decision making methods: A comparative study*, 2000, pp. 5–21.
[46] T. Zahavy, B. Kang, A. Sivak, J. Feng, H. Xu, and S. Mannor, "Ensemble robustness and generalization of stochastic deep learning algorithms," *arXiv preprint arXiv:1602.02389*, 2016.
[47] Z. Xu, "NavRL," https://github.com/Zhefan-Xu/NavRL, 2024.
[48] P. Föhn, E. Kaufmann, A. Romero, R. Penicka, S. Sun, L. Bauersfeld, T. Laengle, G. Cioffi, Y. Song, A. Loquercio *et al.*, "Agilicious: Open-source and open-hardware agile quadrotor for vision-based flight," *Science Robotics*, vol. 7, no. 67, p. eabl6259, 2022.
[49] R. J. Tibshirani and B. Efron, "An introduction to the bootstrap," *Monographs on statistics and applied probability*, vol. 57, no. 1, pp. 1–436, 1993.

APPENDIX

This appendix provides the complete supplementary data referenced in Section IV-B of the main paper. While Figs. 12 to 16 in the main text present heatmaps for a representative subset of platforms (1kg-2kg), the following figures (Figs. 17 to 21) show the complete, fine-grained navigation success rate heatmaps for all 36 platforms (18 real and 18 virtual).

These figures visualize the performance of each evaluated algorithm across all seven navigation scenarios, detailing the specific platform-scenario interactions that inform the aggregated analyses in the main paper.

Fig. 17. **Complete EGO-Planner navigation success rate (All 36 platforms).** The heatmap shows the mean success rate for each of the 36 platforms (y-axis) across all seven scenarios (x-axis). Success rates are color-coded from 0.0 (dark purple) to 1.0 (bright yellow).

Fig. 18. **Complete Fast-Planner navigation success rate (All 36 platforms).** The heatmap shows the mean success rate for each of the 36 platforms (y-axis) across all seven scenarios (x-axis). Success rates are color-coded from 0.0 (dark purple) to 1.0 (bright yellow).

Fig. 19. **Complete Path-Guided PGO navigation success rate (All 36 platforms).** The heatmap shows the mean success rate for each of the 36 platforms (y-axis) across all seven scenarios (x-axis). Success rates are color-coded from 0.0 (dark purple) to 1.0 (bright yellow).

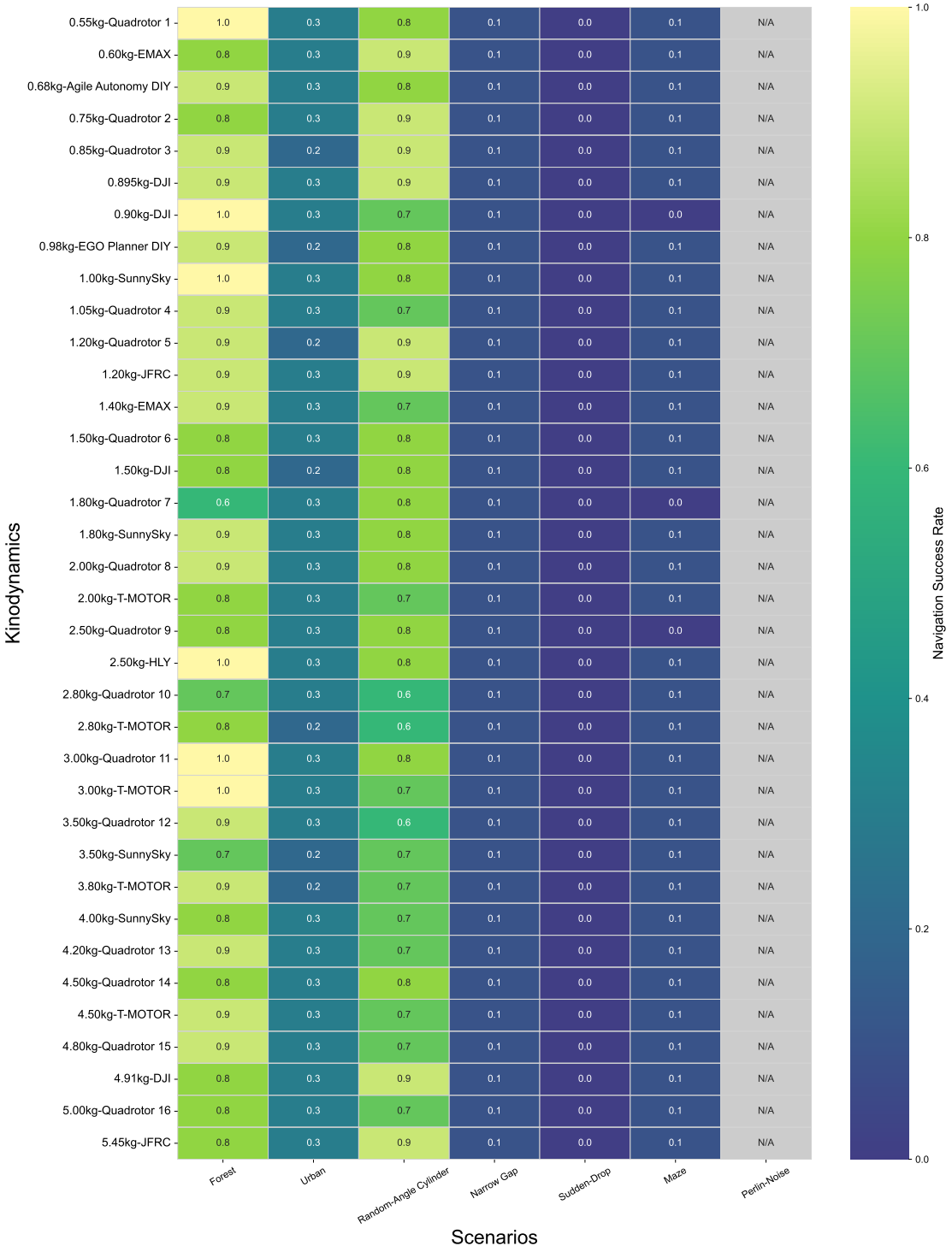


Fig. 20. **Complete NavRL navigation success rate (All 36 platforms).** The heatmap shows the mean success rate for each of the 36 platforms (y-axis) across all scenarios (x-axis). Success rates are color-coded from 0.0 (dark purple) to 1.0 (bright yellow). The ”Random Perlin-Noise” scenario (far right) is marked N/A, as this scenario was not available in the Gazebo simulator used for this algorithm (see Section III-B).

Fig. 21. **Complete Straight-Flight Baseline navigation success rate (All 36 platforms).** The heatmap shows the mean success rate for each of the 36 platforms (y-axis) across all seven scenarios (x-axis). Success rates are color-coded from 0.0 (dark purple) to 1.0 (bright yellow).