Downscaling Autonomous Navigation of Tiny Drones under Dense Forest Canopies

Andreas Zwanenburg, Salua Hamaza
BioMorphic Intelligence Lab, Dept. of Control & Operations, Faculty of Aerospace Engineering,
TU Delft, The Netherlands

Abstract

This study investigates the effects of downscaling hardware and software on the performance and processor load of small drones, with a focus on rainforest exploration. Through systematic benchmarking, we evaluate state-of-the-art path-planning on resource-limited platforms, examining trade-offs between computational load and key navigation metrics including gap size, obstacle detection, and reaction time. We offer practical guidelines for autonomous navigation on low-power processors in dense, narrow-gap environments. Finally, we demonstrate the feasibility of this approach through a 100-gram prototype achieving full onboard autonomous navigation through unknown, unstructured forest environments.

Figure 1: The 100-gram autonomous drone (DJI Tello with Raspberry Pi Zero 2W, and Arducam ToF camera) navigating in a dense forest.

1 Introduction

Tropical rainforests, essential to the planet's ecological health, are increasingly threatened by deforestation and climate change [1]. Preserving these ecosystems demands a deeper understanding of their biodiversity, a task complicated by the sheer size and inaccessibility of rainforest environments. In recent years, drones have emerged as a promising solution, enabling researchers to monitor forest conditions, gather critical biodiversity data, and evaluate the effects of climate change and human activity, as shown by [2–6]. A key advantage of drones lies in their ability to navigate remote, hard-to-reach areas with relative ease [7, 8].

Advances in aerial robotics have greatly expanded the autonomy of unmanned aerial vehicles (UAVs). The EGO-Planner, introduced by Zhou et al. [9], marked a milestone in making high-performance local planning feasible on small quadrotors, while recent work demonstrated vision-based route following indoors on a 56-gram drone [10], hinting at the potential for outdoor use. However, that system is limited to retracing previously flown routes in obstacle-free conditions, whereas our system performs fully autonomous navigation through unknown, unstructured environments. Such route-following approaches, while highly memory-efficient, require prior traversal of the environment and cannot handle unknown obstacles — capabilities that are essential for rainforest exploration, where terrain is unstructured and no prior map is available. Yet many state-of-the-art obstacle-avoiding algorithms still depend on high-performance processors and sensors, adding significant weight and reducing endurance. For instance, the EGO-Planner implementation employs a 150-gram microprocessor and a 40-gram stereo depth camera on a 900-gram platform, leading to a take-off weight of over 1 kg and flight time of 8 minutes. Lighter onboard processors and sensors directly extend flight endurance by reducing payload mass; this also enables the use of smaller, cheaper, and safer drone platforms that achieve comparable endurance to heavier systems, making them more practical for real-world applications.

Within this work, our objective is to advance the understanding of ultra-lightweight autonomous drones and their performance in complex natural environments. We present a fully autonomous aerial system weighing only 100 g, with all computations performed onboard. Its lightweight design enables agile navigation in cluttered environments such as dense tropical forests, addressing critical challenges of rainforest deployment. We systematically analysed the computational demands of path planning, establishing performance metrics to evaluate navigation effectiveness. Benchmarking tests measured computer processor load, memory usage, and loop durations across different configurations, while simulation experiments incorporating real-forest data allowed us to explore the relationship between navigation performance, CPU load, and sensor data quality. This process identified computational bottlenecks within path planning sub-processes, of-

fering insights into power optimization. This paper begins by describing our baseline 500-gram rainforest drone, developed for the XPRIZE Rainforest competition, which served as the starting point for our downscaling experiments.

The main contributions of this work are:

- A systematic benchmarking framework for evaluating the trade-offs between computational load and navigation performance in resource-constrained path planning, using four defined metrics: minimum solvable gap size, minimum detectable obstacle size, solvable obstruction width, and reactiveness time.
- A Pareto-front analysis revealing feasible configurations for autonomous navigation under processor constraints, providing practical guidelines for system designers.
- A proof-of-concept implementation of autonomous 3D obstacle mapping and path planning on a 100-gram drone platform, demonstrating that real-time autonomous navigation through unknown, unstructured outdoor environments is feasible at this scale.

2 Autonomous Flight with a 500-Gram Drone

A 500-gram autonomous drone was developed in the context of the XPRIZE Rainforest Competition [11], which tasks teams with surveying large forest areas without human entry. To address this challenge, we derived a set of technical requirements: fully autonomous flight without external signals (GPS, joystick, or video stream), robust navigation through dense vegetation with obstacles at all heights, and sufficient flight time to cover meaningful ground area. The platform is a Parrot Bebop 2 drone equipped with an Intel RealSense D435 depth camera for obstacle sensing and an ODROID-C4 computer for onboard autonomy. Low-level flight control is handled by the Bebop 2's internal controller, interfaced via the Robot Operating System (ROS) using the bebop_autonomy SDK [12].

2.1 Energy Usage Benchmarking

When downscaling a drone to a smaller form factor, energy consumption becomes one of the biggest bottlenecks. With a smaller battery, smaller motors and smaller propellers, the payload that the drone can handle becomes more critical. Sub-100-gram drones typically have a flight time of approximately 5 minutes. But with even the smallest addition of mass, the available flight time drops significantly. To integrate an autonomous path planner on tiny drones, extra sensors such as a depth camera and a processor unit are required. To optimise path planning in forest conditions on resource-limited processors, benchmark tests are performed to measure CPU and memory load. To interpret these benchmarking results meaningfully, we define a set of performance metrics that capture both navigation safety and computational demand.

Performance Metrics

Autonomous planning involves perception and planning steps. This section introduces four metrics created for safe and robust navigation in dense environments: reactiveness time, solvable obstruction width, minimum detectable object size, and minimum solvable gap size. Experiments on a 500-gram drone indicate that processing load is more affected by perception configurations than planning configurations. Depth image rate and resolution significantly influence processing load. Planner configurations, like cost function and maximum iterations, impact performance but do not significantly alter processor load, therefore these are not considered in the trade-off comparison.

Depth Image Projection

Depth image projection translates 2D depth images into 3D detection points. This process involves using intrinsic and extrinsic camera matrices to determine the 3D position of every pixel with respect to the camera's coordinate frame, illustrated in Figure 2 [13].

Our benchmarking explores how lowering image resolution and frame rate affects perception performance, CPU load and memory load. Once points are projected into 3D, we must update the map to reflect both obstacles and free space—this is handled by raycasting.

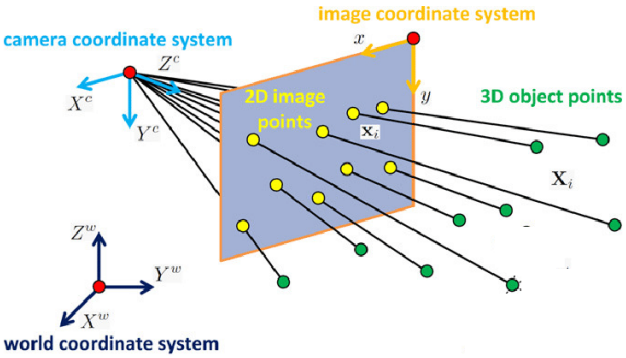

Figure 2: Illustration of depth image projection from a 2D camera frame into a 3D obstacle map, based on camera intrinsics and extrinsics [13].

Raycasting Process

Raycasting updates the 3D map by considering obstacles detected by the drone. The probability of currently mapped obstacles that lie in between the drone and newly detected obstacles will be lowered, applying the probabilistic occupancy mapping concept [14]. It assumes no obstacles lie between the drone and the detected points, crucial for updating the obstacle map. Raycasting addresses sensor noise from the camera, essential to creating a robust obstacle map in challenging environments. With the map updating in real time, we can now quantify what sizes of gaps the drone can navigate.

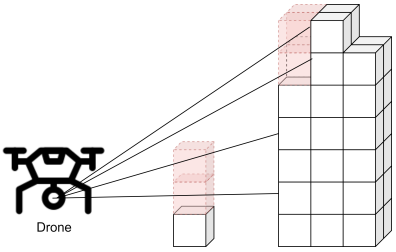

Figure 3: Illustration of the raycasting process used in occupancy mapping. Voxels marked as obstacles but lying between the drone and new detections (red blocks) are removed to reduce false positives and sensor noise.

Minimum Solvable Gap Size

To find path-planning solutions in a dense forest, identifying small gaps is crucial. Therefore, we created the minimum gap size metric, which can be used to determine the sizes of gaps that can be detected. The minimum gap size depends mainly on voxel grid resolution, with smaller voxels improving accuracy but significantly increasing processor and memory load — for example, reducing voxel size from 0.20 m to 0.05 m can increase memory requirements by tens of times, depending on environment structure and octree pruning, as illustrated in Figure 4, observed in our benchmarking, and consistent with voxel-based frameworks like OctoMap [15].

This metric considers factors like pixel size, camera resolution, inflation distance, and grid resolution. Hypotheses suggest CPU load increases with decreasing gap sizes, impacting the performance of the algorithm.

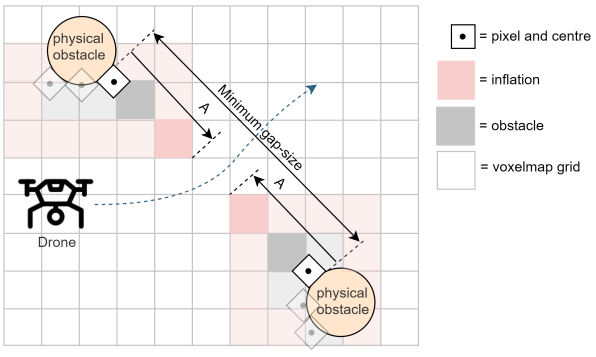

Figure 4: Illustration of the minimum solvable gap size for diagonal gaps in voxel-based mapping. The resolution of the voxel grid determines the smallest gap that can be reliably detected.

Equation 1 defines the guaranteed minimum gap size based on configurations and camera specifications. In the equation, PS is the pixel size, FOV the camera's field of view, R_{cam} the camera range, C_{res} the camera resolution, D_{infl} the inflation distance, D_{min} the minimum inflation distance, GR the voxel grid resolution, and GS_{min} the minimum solvable gap size. A refers to the dimensioning illustrated in Figure 4. The square brackets, also known as the ceiling brackets, indicate that the decimal number inside should be rounded up to the nearest whole number.

$$
\begin{aligned}
PS &= \frac{\tan\left(\frac{FOV}{2}\right) \cdot R_{cam} \cdot 2}{C_{res}} \\
D_{infl} &= \lceil \frac{D_{min}}{GR} \rceil \cdot GR \\
A &= \frac{PS}{2} + \sqrt{2 \cdot (GR)^2} + \sqrt{2 \cdot (D_{infl})^2} \\
GS_{min} &= A \cdot 2 + \sqrt{2 \cdot (GR)^2} \\
GS_{min} &= PS + 3 \cdot \sqrt{2 \cdot (GR)^2} + 2 \cdot \sqrt{2 \cdot (D_{infl})^2}
\end{aligned}
\tag{1}
$$

Depth Image Resolution

Lowering the depth image resolution, either by using a lower-quality camera, or by skipping pixels defined in the configuration, negatively impacts the detection of small obstacles. Thereby creating a risk of not detecting tiny branches in the forest and crashing. Figure 5 illustrates the potential blind zones for a certain configuration and distance to an object. But, at the same time, lowering the number of computations for the projection, thereby lowering the processor load. In benchmarking, the depth camera resolution is varied to assess its impact on drone perception performance.

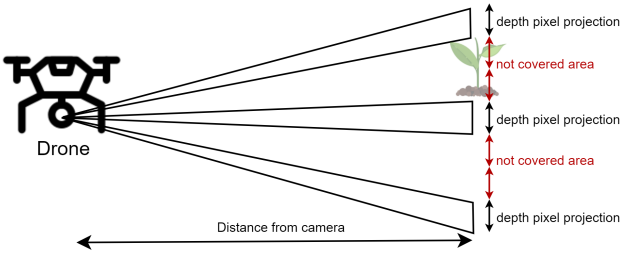

Figure 5: Illustration of pixel skipping in the projection step, where 2 out of every 3 pixels are ignored. This reduces processor load but increases blind zones and the risk of missing thin obstacles.

To estimate the minimal size of a potentially missed object at maximum camera distance, Equation 2 provides an approximation. This equation considers parameters such as FOV field of view, CR_{max} camera range, OR original resolution, and SP skipped pixels. The result, denoted as SM_{max}, represents the maximum object size that may be overlooked. The pixel projection scales linearly with the camera range, allowing detection of smaller objects at closer distances.

$$
\begin{aligned}
PP_{max} &= \frac{\tan\left(\frac{\text{FOV}}{2}\right) \cdot CR_{max} \cdot 2}{OR} \\
SM_{max} &= SP \cdot PP_{max}
\end{aligned}
\tag{2}
$$

Local Planner Range

The EGO-planner's local range is the area around the drone in which obstacles are considered to plan a path around. By only using a local region, the path planner can calculate trajectories more efficiently. When the obstacle is wider than the local planner range, and the planner assumes all space outside the local planner range is obstacle-free, an issue arises. Figure 6 illustrates this issue where the planner gets stuck in a loop where it never succeeds in navigating to the goal. A well-known failure mode of local, potential-field–style planners, a limitation first noted in classic potential-field navigation studies. A smaller range reduces CPU load but may hinder efficient route planning. Balancing local planner range is crucial to avoid navigation loops and ensure effective obstacle avoidance.

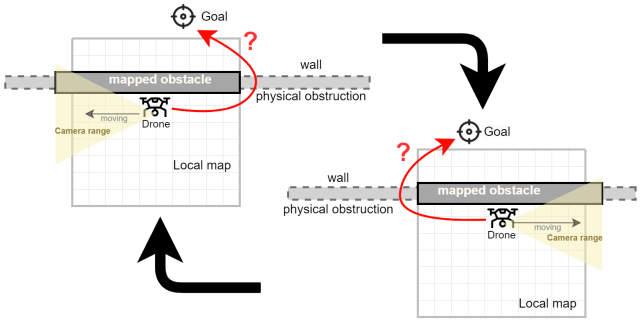

Figure 6: Illustration of the local planner range limitation. When obstructions are wider than the local planning horizon, the planner incorrectly assumes free space beyond the map and may enter an infinite replanning loop.

Equation 3 can be used to determine what obstruction width can be solved by the algorithm. In the equation, D_{infl} is the inflation distance, D_{min} the minimum inflation distance, GR the voxel grid resolution, R_{loc} the local range, and SOW the solvable obstruction width.

$$\begin{aligned} D_{infl} &= \lceil \frac{D_{min}}{GR} \rceil \cdot GR \\ SOW &= R_{loc} - 2 \cdot D_{infl} \end{aligned} \tag{3}$$

Reactiveness

Reducing the depth image rate, by using a lower-quality camera or skipping images by configuration, the processor load can be lowered. But, for newly occurring obstacles, it takes more time to map and detect a possible collision. By varying the depth image rates during benchmarking, the processor load and reactiveness are compared.

Equation 4 shows what sub-processes and configurations affect the theoretical reactiveness duration. This is the duration between the occurrence of a new obstacle, up to the moment it is mapped and a collision check is performed. In this equation, PI is the number of iterations until the threshold is reached, P_t and P_i are respectively the probability threshold and probability increase, R_{cam} the camera rate, and T_{proj}, T_{ray}, T_{upd}, T_{col} the loop times for projection, raycasting, updating, and collision check.

$$\begin{aligned} PI &= \lceil \frac{P_t}{P_i} \rceil \\ T_{react} &= PI \cdot \frac{1}{R_{cam}} + T_{proj} + T_{ray} + T_{upd} + T_{col} \end{aligned} \tag{4}$$

2.2 *Configuration Variations*

Four parameters are varied to compare planner performance by the metrics mentioned above. These are voxel grid resolution, local map range, camera resolution, and camera rate. Each parameter variation explores its impact on the processor and memory load, offering insights into optimal configurations for autonomous flight in challenging environments. Table 1 shows per metric what parameter combinations are benchmarked.

Table 1: Simulation Experimental Setup

	Voxelgrid res. [m]	**Local map range [m]**	**Camera res. [-]**	**Camera rate [Hz]**
Gap size	[.05 .10 .15 .20]	10	[1 1/2 1/3 1/4]	15
Obstacle size	0.10	10	[1 1/2 1/3 1/4]	15
Obs. width	[.05 .10 .15 .20]	[1 2 ... 35]	1	15
Reaction time	0.10	10	1	[15 7.5 5 3.8 3]

3 Dataset Collection

Creating a dataset for simulations involves several critical steps, from data collection to processing. Initially, ROS nodes are configured to output loop durations for specific sub-processes, providing valuable insights into performance. Ground-truth data is also gathered during dataset creation to enable comparison of different methods. This section details the process of generating an outdoor dataset.

1) Data Collection Setup: To monitor the performance of the EGO-planner algorithm, ROS publishers are integrated to output data related to the duration of key sub-processes: projection, raycasting, map updates, and collision-checking for each processed depth image. This data is logged for later analysis.

2) Processor and Memory Load Monitoring: To evaluate the processor load and memory usage, the `cpu_monitor` ROS package [16] is used, publishing system metrics at a 10

Hz interval. This helps assess the computational resources consumed by the algorithm during operation.

3) Evaluation of State Estimation Methods: The study evaluates a drone path-planning algorithm based on dead reckoning by conducting performance tests in both indoor and outdoor environments, each requiring distinct data collection methods. For indoor testing, high-precision motion data is obtained using the Optitrack Motion Capture System, which integrates seamlessly with the ROS network via MAVros. Outdoor tests take place in dense forest settings, where GPS data provides the ground-truth position. A UBLOX F9P GPS module equipped with a helix antenna is mounted on the drone to enable comparison between dead reckoning and GPS-based navigation.

In both scenarios, inertial measurement unit (IMU) and image data are collected to enhance position estimation through visual-inertial odometry. This is implemented using VINS-Fusion [17], built on VINS-Mono [18] and leveraging techniques also common to ORB-SLAM2 [19], which tracks landmarks in infrared images similarly to the EGO-planner [9]. Together, these approaches allow the study to comprehensively assess the effectiveness of the dead reckoning-based path-planning method across varied environmental conditions by leveraging motion capture, GPS, and visual-inertial odometry data.

4) Outdoor Dataset Collection: The outdoor dataset is recorded in a dense forest environment resembling rainforest conditions. During autonomous flight, the drone navigates a walking trail with obstacles, capturing sensor data throughout the flight. This dataset ensures repeatability, using real-forest sensor data, and minimises the need for further outdoor testing to validate algorithm configurations [20]. Our 3D mapping pipeline is conceptually aligned with recent under-canopy photogrammetric work using stereo vision to generate dense point clouds for parameter extraction [21].

5) Simulation Benchmarking: To assess the drone path-planning algorithm, simulations are run using different configurations while replaying the sensor data from the dataset. This section outlines the process of preparing datasets, running simulations, and extracting results for analysis. The dataset is curated by filtering and trimming the logged data from real flights, focusing on the drone's IMU data and the depth and grayscale images captured by the camera.

For efficient simulation and data collection, a bash script automates the process, running the planner algorithm on the dataset with various configurations. The results are then processed into Python data for further analysis. This automated approach allows for batch processing of over 100 simulations, making it a time efficient method for bulk analysis.

4 Analysis

The analysis script processes all data using specific function definitions for calculating performance metrics as described in Section 2.1. This includes pixel size, minimum gap size, maximum obstruction width, minimum obstacle size, and reactiveness.

When downsizing computational load for tiny drones, it is crucial to closely evaluate the CPU load profile during simulations. The perception phase, focused on updating the local map using depth images, differs from the planning phase, marked by peak loads during path sampling, optimization, and refinement. Figure 7 exemplifies the CPU load distribution during a simulation run with varying voxel grid resolutions. In this example, left of the vertical dotted line is only perception. While on the right side of the dotted line, the perception and path planning are both active. For the benchmarking, the path planning is triggered to re-plan every 2 seconds, causing the peaks in the CPU load.

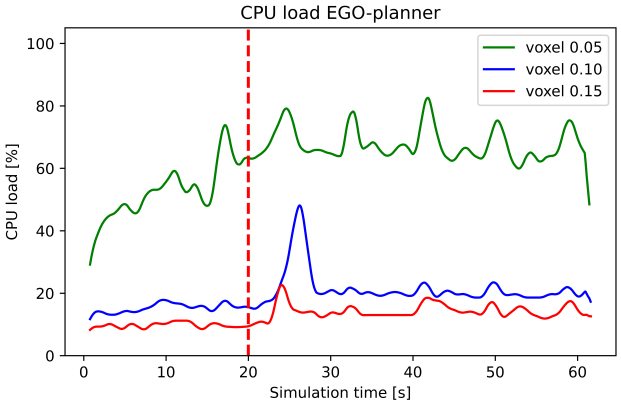

Figure 7: CPU load distribution over the benchmark simulation with voxel-grid resolutions of 0.05, 0.10, and 0.15 m.

5 Results

This section presents the benchmarking results for each of the four performance metrics defined in Section 2.

5.1 Gap Size Performance

CPU load was measured for varying voxel sizes and image resolutions. The relationship is non-linear, with smaller voxels increasing load. Figure 8 shows how grid resolution affects CPU load and the ability to navigate narrower gaps. Memory usage follows the same trend, increasing significantly with resolution; reducing voxel size from 0.20 m to 0.05 m can increase memory requirements by tens of times, depending on pruning and environment structure. This indicates that higher resolution improves gap navigation but at steep computational cost—critical for small onboard processors.

5.2 Obstacle Size Performance

For detecting smaller obstacles, the relation is tested with decreasing CPU load during the benchmarking. This test involves varying the depth image resolution and voxel grid resolution. Results, shown in Figure 9, indicate that at the

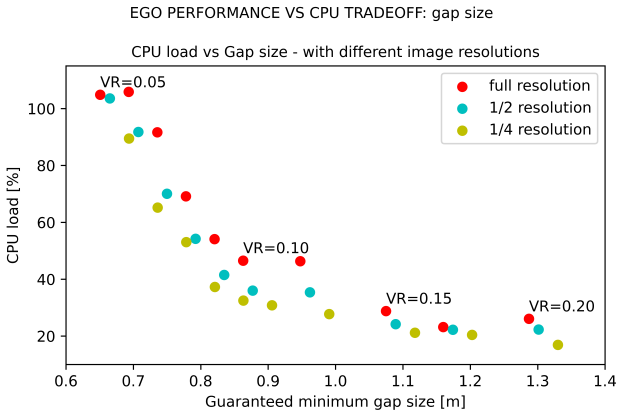

Figure 8: CPU load (%) vs. minimum solvable gap size (m) at full, half, and quarter image resolution.

finest voxel resolution tested (5 cm), relaxing the minimum detectable obstacle size leads to a significant, non-linear reduction in CPU load. At coarser voxel resolutions (7–15 cm), the same relaxation yields only a modest ∼5% reduction, which is not considered significant. This suggests that detecting only larger obstacles yields modest computational savings unless voxel resolution is already high.

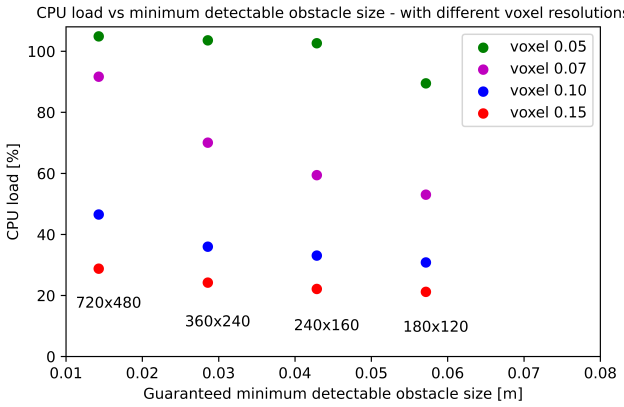

Figure 9: CPU load (%) vs. minimum detectable obstacle size (m) for voxel resolutions of 0.05 m, 0.07 m, 0.10 m, and 0.15 m.

5.3 *Obstruction Width Performance*

As mentioned earlier, path planning around wide obstructions requires a larger range for the local planner. In the benchmark, CPU load was tested for increasing local planner ranges and varied across different voxel grid resolutions. The measured loads exhibit a roughly linear increase as the obstruction width that can be handled grows, see Figure 10.

For high voxel resolutions, such as 10 cm, CPU load rises significantly—from about 35% for a 2 m wide obstruction to 85% for a 35 m wide obstruction. In contrast, for lower voxel resolutions, such as 20 cm, the load increases by only 5–10% across the same range, which is considered insignificant.

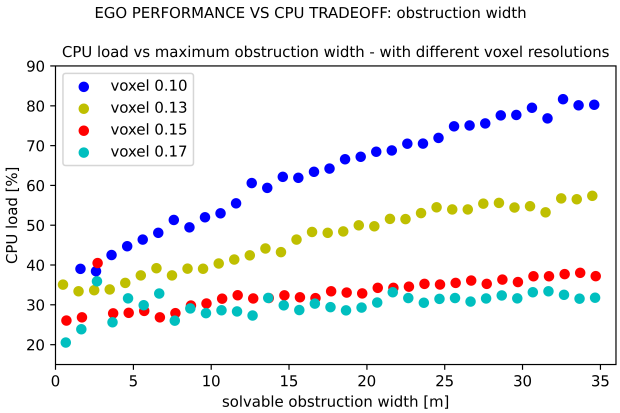

Figure 10: CPU load for various minimal obstacle widths with voxel resolutions 0.10 m, 0.13 m, 0.15 m, and 0.17 m.

5.4 *Reactive Performance*

To test the potential load increase for a system that has a quicker reaction to newly occurring obstacles, the rate of the depth images is varied in the benchmark tests from 3 to 15 depth images per second. The voxel grid resolution is also varied to increase the general system load to check if this differs in the results. The results in Figure 11 show a slightly curved decrease in load when the image rate is lowered. The plot shows that for a system with high load, the impact of decreasing the rate has significantly more impact (25% reduction) compared to the test where the general system load was low (5% reduction).

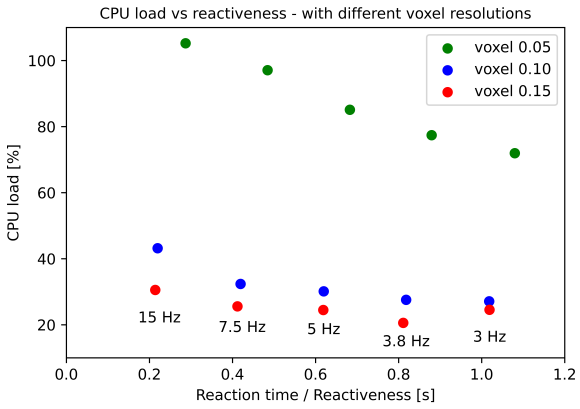

Figure 11: CPU load for various reaction times with voxel resolutions 0.05 m, 0.10 m, and 0.15 m.

5.5 *Trade-off: Safety vs. Performance*

By combining the results, trade-offs can be visualised in which the CPU loads can be plotted for a combination of two performance variations, also known as a Pareto-front [22]. The main objective is the comparison of CPU load relative to flight performance, which is represented by the minimal gap size and maximum obstruction width. Safe and robust flight was one of the secondary objectives, and this is represented by reactiveness and minimal obstacle-size performance. Exploring the trade-off between safety and performance at different CPU utilization, using reactiveness and minimal gap size, reveals a Pareto-front.

The Pareto-fronts, shown in Figure 12, indicate the performance combinations where the CPU load is equal, drawn in yellow for 90% CPU utilisation, and drawn in purple for 50% CPU utilisation. This Pareto-front shows the trade-off where a combination of safety and performance has to be chosen, with a limited CPU load, that is most suitable for a design situation.

For example, in the trade-off in Figure 12, for a CPU load limit of 50%, a minimum gap size of 0.8 m can be chosen with a reactiveness of 1.2 seconds. Or, the combination with a much quicker reactiveness can be chosen, for example, 0.2 seconds, but now the minimum gap size is 1.1 m. Any combination on the purple line would be valid within the 50% load limit.

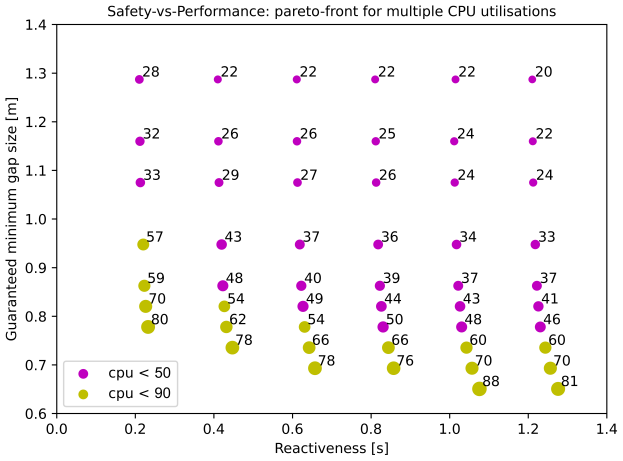

Figure 12: Trade-off between safety and performance shown as Pareto fronts for CPU utilization levels of 50% and 90%. Each curve illustrates feasible combinations of reactiveness and minimum gap size under processor constraints.

6 Autonomous Flight with a 100-Gram Drone

Unlike prior sub-100 g autonomous systems, which are limited to route following in controlled indoor environments [10], the platform presented here performs real-time 3D obstacle mapping and path planning in unknown, unstructured environments. This is achieved by fitting a Raspberry Pi Zero 2W processor and an Arducam ToF camera onto a DJI Tello drone, resulting in a fully autonomous 100-gram, 10×10×10 cm system with all computation performed onboard. The total material cost is approximately 200 euros, of which 100 euros is the drone platform itself, making it a low-cost solution compared to existing autonomous systems of comparable capability. Low-level flight stabilization and velocity control are handled by the Tello's internal controller, with high-level commands sent via the DJITelloPy SDK [23].

The parameter configuration for the 100-gram platform was derived directly from the benchmarking results presented in Section 5, using real forest sensor data as input. By selecting a 10 cm voxel grid resolution, 10 depth images per second at 120×90 px resolution, and a 5 m local planner range, the processor operates at 80–95% CPU load — within the safe limits of stable operation on the Raspberry Pi Zero 2W. This configuration yields the following navigation performance: a minimum solvable gap size of 1.0 m, a reaction time of 300 ms, a minimum detectable obstacle size of 2 cm, and a maximum solvable obstruction width of 3.6 m. These values represent a deliberate trade-off between computational feasibility and navigation safety, selected using the Pareto-front analysis described in Section 5.

Preliminary flight tests were conducted in a controlled indoor environment with artificial vegetation arranged at approximately 1.0 m gap spacing, consistent with the predicted minimum solvable gap size. The drone successfully navigated autonomously between obstacles, confirming the benchmarking-derived parameter configuration in practice. Outdoor flights in a forest environment demonstrated functional obstacle avoidance, though thin branches were occasionally detected only at close range, consistent with the 2 cm minimum detectable obstacle size and 300 ms reaction time reported above. The addition of the companion computer and sensor payload reduced flight time from the platform's original 13 minutes to approximately 6–7 minutes. Full quantitative outdoor validation remains future work.

7 Conclusion

This study demonstrates that downsizing hardware and adapting algorithms enable small drones to achieve effective autonomous navigation in dense rainforest conditions. By systematically varying parameters such as voxel grid resolution, camera resolution, and local planning range, this research shows clear trade-offs between processor load, memory usage, and crucial performance metrics like minimal gap size, obstacle detection, and reaction time. Our findings indicate that reductions in image rate or resolution can substantially decrease CPU load but must be carefully balanced against the increased risk of missing small obstacles, or reacting too slowly to new hazards. This balance is further illustrated through Pareto-front analyses, offering insights into optimal configurations for different mission requirements. Fi-

nally, the successful implementation of autonomous 3D obstacle mapping and path planning on a 100-gram drone platform confirms that real-time autonomous navigation through unknown, unstructured environments is feasible at this scale. This was demonstrated in both controlled indoor environments and outdoor forest conditions. These results open a promising path for future applications in biodiversity monitoring and ecological conservation.

ACKNOWLEDGEMENTS

The authors would like to thank Erik van der Horst for his support at the lab and during outdoor flights.

REFERENCES

[1] Stephen Pringle, Martin Dallimer, et al. Opportunities and challenges for monitoring terrestrial biodiversity in the robotics age. *Nature Ecology & Evolution*, pages 1–12, 2025.

[2] Salua Hamaza, H Nguyen, et al. Sensor delivery in forests with aerial robots: A new paradigm for environmental monitoring. In *IEEE IROS Workshop on Perception, Planning and Mobility in Forestry Robotics*, 2020.

[3] Liming Zheng and Salua Hamaza. Albero: Agile landing on branches for environmental robotics operations. *IEEE Robotics and Automation Letters*, 9(3):2845–2852, 2024.

[4] Emanuele Aucone, Steffen Kirchgeorg, et al. Drone-assisted collection of environmental DNA from tree branches for biodiversity monitoring. *Science Robotics*, 8(74):eadd5762, 2023.

[5] Christian Geckeler, Steffen Kirchgeorg, et al. Field deployment of BiodivX drones in the Amazon rainforest for biodiversity monitoring. *IEEE Transactions on Field Robotics*, 2025.

[6] Rita Santos Raminhos and Salua Hamaza. Task-oriented path planning towards autonomous sensor network deployment in rainforest canopies. *Advanced Robotics Research*, 2026.

[7] Yulun Tian, Katherine Liu, et al. Under canopy search and rescue using multiple UAVs. *International Journal of Robotics Research*, 39(10-11):1201–1221, 2020.

[8] Philipp Foehn, Elia Kaufmann, et al. Agilicious: Open-source and open-hardware agile quadrotor for vision-based flight. *Science Robotics*, 7(67):eabl6259, 2022.

[9] Xin Zhou, Zhepei Wang, et al. Ego-planner: An ESDF-free gradient-based local planner for quadrotors. *IEEE Robotics and Automation Letters*, 6(2):478–485, 2020.

[10] Tom van Dijk, Christophe De Wagter, and Guido C.H.E. de Croon. Visual route following for tiny autonomous robots. *Science Robotics*, 9(92), 2024.

[11] XPRIZE. XPRIZE rainforest—XPRIZE foundation. `https://www.xprize.org/prizes/rainforest`, 2023.

[12] Mani Monajjemi. bebop_autonomy: ROS driver for Parrot Bebop drone. `https://bebop-autonomy.readthedocs.io/en/latest/`, 2014.

[13] Ma Weinmann, Mi Weinmann, et al. Fast and automatic image-based registration of TLS data. *ISPRS Journal of Photogrammetry and Remote Sensing*, 66(6 SUPPL.), 12 2011.

[14] Sebastian Thrun, Wolfram Burgard, et al. *Probabilistic Robotics*. Intelligent Robotics and Autonomous Agents. MIT Press, Cambridge, MA, 2005.

[15] Armin Hornung, Kai M. Wurm, et al. Octomap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 34(3):189–206, 2013.

[16] Alexander Spitz. cpu_monitor: CPU and memory monitoring ROS package. `https://github.com/alspitz/cpu_monitor`, 2019.

[17] Tong Qin, Shaozu Cao, et al. A general optimization-based framework for global pose estimation with multiple sensors. *arXiv preprint*, 2019.

[18] Tong Qin, Peiliang Li, and Shaojie Shen. VINS-Mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics*, 34(4):1004–1020, 2018.

[19] Raul Mur-Artal and Juan D. Tardós. ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-D cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017.

[20] Eric Hyyppa, Juha Hyyppa, et al. Under-canopy UAV laser scanning for accurate forest field measurements. *ISPRS Journal of Photogrammetry and Remote Sensing*, 164:41–60, 6 2020.

[21] Vaino Karjalainen, Niko Koivumaki, et al. Autonomous under-canopy forest mapping using stereo vision and photogrammetric reconstruction. *International Journal of Remote Sensing*, 2025.

[22] Kalyanmoy Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, Chichester, UK, 2001.

[23] Damia Fuentes. DJITelloPy: DJI tello drone Python interface. `https://github.com/damiafuentes/DJITelloPy`, 2021.