

# A Sim-to-Real Deep Learning-based Framework for Autonomous Nano-drone Racing

Lorenzo Lamberti<sup>1</sup>, Elia Cereda<sup>2</sup>, Gabriele Abbate<sup>2</sup>, Lorenzo Bellone<sup>3</sup>, Victor Javier Kartsch Morinigo<sup>4</sup>,  
 Michał Barciś<sup>3</sup>, Agata Barciś<sup>3</sup>, Alessandro Giusti<sup>2</sup>, Francesco Conti<sup>1</sup>, and Daniele Palossi<sup>2,4</sup>

**Abstract**—Autonomous drone racing competitions are a proxy to improve unmanned aerial vehicles’ perception, planning, and control skills. The recent emergence of autonomous nano-sized drone racing imposes new challenges, as their  $\sim 10$  cm form factor heavily restricts the resources available onboard, including memory, computation, and sensors. This paper describes the methodology and technical implementation of the system winning the first autonomous nano-drone racing international competition: the “IMAV 2022 Nanocopter AI Challenge.” We developed a fully onboard deep learning approach for visual navigation trained only on simulation images to achieve this goal. Our approach includes a convolutional neural network for obstacle avoidance, a sim-to-real dataset collection procedure, and a navigation policy that we selected, characterized, and adapted through simulation and actual in-field experiments. Our system ranked 1<sup>st</sup> among seven competing teams at the competition. In our best attempt, we scored 115 m of traveled distance in the allotted 5-minute flight, never crashing while dodging static and dynamic obstacles. Sharing our knowledge with the research community, we aim to provide a solid groundwork to foster future development in this field.

## SUPPLEMENTARY MATERIAL

Supplementary video at: <http://youtu.be/vHTAwUsj-nk>

## I. INTRODUCTION

Competitions have always been a catalyst for scientific and technological progress. As the *space race* was a driver to develop programmable computers and microchips, likewise, in recent years, autonomous drone racing pushed the development of cutting-edge navigation algorithms, including artificial intelligence (AI), running aboard unmanned aerial vehicles (UAVs) [1], [2], [3]. Competitions act as a proxy to improve UAVs’ perception, planning, and control skills: from 2016 to date, the racetrack’s average flight speed of autonomous drones increased from 0.6 to 22 m/s [4]. Eventually, these advancements positively impact a more comprehensive range of applications where robust, agile, and precise autonomous navigation is crucial, such as rescue missions [5] and human-robot interaction [6].

So far, these competitions have focused on micro-sized drones, i.e.,  $\sim 30$  cm-wide robots capable of hosting powerful processors and rich sensors, *autonomous nano-sized drone racing* constitutes a newborn category employing palm-sized

<sup>1</sup>L. Lamberti and F. Conti are with the Department of Electrical, Electronic, and Information Engineering, University of Bologna, Italy, [lorenzo.lamberti@unibo.it](mailto:lorenzo.lamberti@unibo.it)

<sup>2</sup>E. Cereda, G. Abbate, A. Giusti, and D. Palossi are with the Dalle Molle Institute for Artificial Intelligence, USI-SUPSI, Switzerland,

<sup>3</sup>L. Bellone, M. Barciś, and A. Barciś are with the Autonomous Robotics Research Center, Technology Innovation Institute, UAE.

<sup>4</sup>V. J. Kartsch Morinigo and D. Palossi are with the Integrated Systems Laboratory, ETH Zürich, Switzerland.



Fig. 1. Our nano-drone winning the IMAV’22 “Nanocopter AI Challenge.”

UAVs. The first competition of this kind was the “Nanocopter AI Challenge” hosted at the 13<sup>th</sup> International Micro Air Vehicle conference (IMAV’22). This last-born class of robotic competitions poses a new challenge to roboticists due to the small size of nano-drones, i.e., 10 cm in diameter and a few tens of grams in weight. While nano-drone racing targets tasks similar to major competitions for bigger drones (e.g., AlphaPilot, IROS Drone Racing, etc.), their ultra-tiny form factor allows only minimal onboard resources, i.e., memory, computation, and sensors.

In the IMAV’22 competition, all challengers run their navigation algorithms on the same platform: a commercial off-the-shelf (COTS) Crazyflie 2.1 nano-drone equipped with the AI-deck board featuring a GWT GAP8 System-on-Chip (SoC) [10] and a grayscale, low-resolution camera [11]. Compared to the typical processors found on micro-sized racing drones [2], [1], e.g., Nvidia Jetson Xavier, the GAP8 SoC has more than 1000 $\times$  less compute power and memory. Despite this, the competition encouraged onboard computation by granting a 5 $\times$  score multiplier while challenging the nano-drone with agile maneuvers to *i*) avoiding static and dynamic obstacles and *ii*) passing through a set of gates in a **never-seen-before** indoor arena. In fact, before the competition, no arena map or real-world dataset was released; participants could only access a photorealistic simulator.

**This paper’s main contribution is a thorough analysis and description of the strategy, methodology, and technical implementation we employed to win the IMAV’22 competition: a fully-onboard deep learning-based visual navigation framework trained only on simulation data.** In detail, we present *i*) an exhaustive discussion of our strategy, which accounts for both the competition’s guidelines and the nano-drone’s limitations; *ii*) a convolutional neural network (CNN) for obstacle avoidance derived from the open-source *PULP-Dronet* [9] and trained only on simulation images; *iii*) mitigation of the *sim-to-real* gap [12] via aggressive photo-

TABLE I  
LITERATURE REVIEW FOR OBSTACLE AVOIDANCE (OA) AND GATE-BASED NAVIGATION (GN) ON UAVS.

Work	Size	Onboard	Task	Perception	Algorithm	Data	Compute device	Power	Competition
Wagter <i>et al.</i> [1]	Micro	✓	GN	Camera	CNN	Real	Jetson AGX Xavier	30 W	AlphaPilot'21
Foehn <i>et al.</i> [2]	Micro	✓	GN	Stereo camera	CNN	Real	Jetson AGX Xavier	30 W	AlphaPilot'21
Kaufmann <i>et al.</i> [7]	Micro	✓	GN	Camera	CNN	Sim	Intel UpBoard	13 W	IROS'18
Jung <i>et al.</i> [3]	Micro	✓	GN/OA	Stereo camera	CV	None	Jetson TK1s	15 W	IROS'16
Pham <i>et al.</i> [8]	Micro	✓	GN	Camera	CNN	Sim	Jetson TX2	15 W	—
Niculescu <i>et al.</i> [9]	Nano	✓	OA	Camera	CNN	Real	GWT GAP8	100 mW	—
<b>Ours</b>	<b>Nano</b>	<b>✓</b>	<b>OA</b>	<b>Camera</b>	<b>CNN</b>	<b>Sim</b>	<b>GWT GAP8</b>	<b>100 mW</b>	<b>IMAV'22</b>

metric augmentation, label balancing, and comprehensive data generation; *iv)* three alternative *navigation policies*, which we characterize both in simulation and in the field.

Our final system, employing the best-performing navigation policy, ranked first among six contending teams at the IMAV'22 competition. In our best run, we scored 115 m of traveled distance in the allotted 5 min-flight, never crashing, dodging dynamic obstacles, and only using computational resources aboard our nano-drone. Our result demonstrates the effectiveness of the proposed sim-to-real mitigation strategy, our implementation's robustness, and our rationale's soundness, and by sharing our insights, we aim to provide the research community with a solid groundwork for the evolution of this field.

## II. RELATED WORK

As the “Nanocopter AI Challenge” focuses on obstacle avoidance (OA) and gate-based navigation (GN), in this section, we focus on these two complex tasks surveying the SoA for various class sizes of drones (Tab. I).

**Obstacle avoidance:** while racing micro-drones can carry bulky sensors (e.g., Lidars [3], stereo cameras [3], [2]) and GPUs with a power envelope of up to 30 W (Tab. I), nano-UAVs suffer from limited perception capabilities due to their tiny low-power sensors and microcontroller units (MCUs) [9]. State-of-the-Art (SoA) perception algorithms, such as simultaneous localization and mapping (SLAM) [13], can not run onboard nano-drones due to their steep performance requirements. Even when run off-board, they suffer from substantial performance degradation as nano-drones employ low-quality sensors [14], [15].

Lightweight approaches better suited to nano-drones employ different sensors such as Time-of-Flight (ToF) ranging sensors [16], [17], [18]. These approaches can be implemented onboard and provide robust obstacle avoidance, even in unknown environments, with raw sensor readings or minimal onboard processing (e.g., 30 OP/s in [18]), such as simple bug-inspired lightweight state machines [16], [17]. However, in our competition, only a low-resolution monochrome monocular camera was allowed, narrowing the teams’ effort only to visual-based approaches.

A lightweight vision-based system for pocket-sized drones was presented by McGuire *et al.* [19], implementing depth estimation with a stereo camera and achieving obstacle avoidance at low speed (0.3 m/s). The SoA visual-based CNN

for autonomous nano-drone navigation is PULP-Dronet [9], trained on real-world data to predict a steering angle and a collision probability based on a QVGA monocular image. PULP-Dronet runs onboard the GAP8 SoC at 19 frame/s, proving in-field obstacle avoidance capabilities up to a speed of 1.65 m/s when coping with a dynamic obstacle. As a result, at the IMAV competition, 3 of 6 teams, including us, employed this CNN as a starting point to build their visual obstacle avoidance pipeline. First, we modified the PULP-Dronet reference implementation in its task, i.e., the original CNN predicts a collision probability and a steering angle. Instead, our model is optimized to predict three collision probabilities by horizontally splitting the input image into three regions. Then, we introduced a novel training pipeline that exclusively relies on simulation, while the original work uses real-world images based on autonomous driving cars. Finally, we enriched our simulator with a photometric augmentation pipeline, which increased the generalization capabilities of our model up to a top-scoring in-field performance. Compared to PULP-Dronet, this work targets more aggressive obstacle avoidance capabilities: up to 2 m/s speed, concurrent static and dynamic obstacles, and 5 min uninterrupted flight. A thorough discussion on the comparison with the SoA PULP-Dronet is presented in Sec. VI-B.

**Gate-based navigation:** SoA approaches for micro-drone racing competitions tackle trajectory planning and optimization, gate detection, and control [4]. However, their prohibitive complexity prevents implementing them on nano-drones, even without considering the computational budget needed by the obstacle avoidance task. Time-optimal trajectory optimization relies on model predictive control (MPC) solving linear algebra at high-frequency ( $\sim 100$  Hz) and with real-time constraints [20]. To cope with the MPC complexity, Foehn *et al.* [20] exploit an NVIDIA Jetson TX2 GPU, which has  $\sim 60\times$  higher computation capabilities than GAP8.

For gate detection, the SoA exploits either *i)* CNNs for segmentation of high-resolution images [2], which results in a computational complexity of more than 3 GOPs [2] per inference, or *ii)* traditional computer vision approaches with stereo images [3]: both are still out of reach for nano-drones. Kaufmann *et al.* [7] proposed a simpler CNN for predicting the gate’s poses directly from the image, but this technique runs at only 10 Hz on a powerful ( $\sim 13$  W) Intel UpBoard and relies on a coarse map of the gate’s positions. PencilNet [8] is

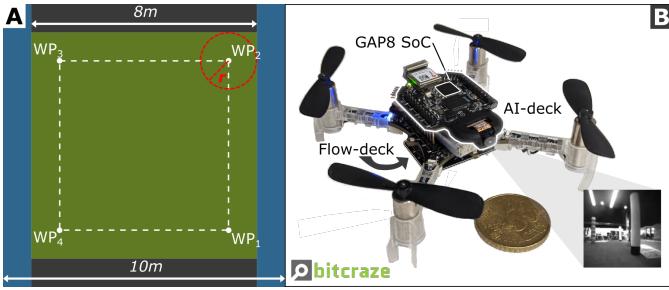


Fig. 2. A) the  $10 \times 10\text{ m}$  competition arena. B) the robotic platform.

a lightweight CNN (32k parameters and 53k MAC operations per frame) for gate pose estimation trained on simulated images, addressing the sim-to-real gap with an intermediate image representation. From a computational/memory point of view, this CNN is suitable for real-time execution on our nano-drone. However, the gate pose estimation is only part of a more complex pipeline to achieve gate-based navigation, which additionally requires a memory-consuming mapping of the environment and a more complex trajectory planning. For this reason, the PencilNet CNN was demonstrated on a drone equipped with a powerful Nvidia Jetson TX2, with a power consumption  $75\times$  higher than our nano-drone's GAP8 SoC, and a high-end Intel RealSense T265 for the state estimation.

More lightweight approaches for visual servoing have also been demonstrated aboard autonomous nano-drones, based on simple computer vision approaches, such as color segmentation, to cope with the platform's limited computational resources. In [21], the authors used raw image data to detect and fly through monochromatic gates, while in [22], a simple target-tracking algorithm for monochromatic objects was introduced. However, both works do not target high-speed scenarios, resulting in too limited agility for drone racing.

### III. THE IMAV'22 NANOCOPTER AI CHALLENGE

**The competition:** the metric to assess each team's score is the distance traveled within the mission area (the  $8 \times 8\text{ m}$  green square shown in Fig. 2-A) within the allotted time (5 min), employing a Crazyflie 2.1 nano-drone. This distance is measured with a motion capture system that ignores the part of the trajectory lying outside the mission area. Participants can choose between two coefficients of difficulty, one accounting for *environmental complexity* ( $\alpha_{\text{env}}$ ) and the other for *computational resources* ( $\alpha_{\text{comp}}$ ). Additionally, in the arena, there are two rectangular gates through which the nano-drone can fly; every time the drone passes through a gate, the distance is increased by an additional 10 m.

Participants can choose among three levels of environmental complexity: only static gates ( $\alpha_{\text{env}} = 1\times$ ), static gates and obstacles ( $\alpha_{\text{env}} = 5\times$ ), and static gates and dynamic obstacles ( $\alpha_{\text{env}} = 10\times$ ). If the nano-drone uses remote computation (i.e., Wi-Fi-connected commodity laptop), the  $\alpha_{\text{comp}}$  is  $1\times$ ; for onboard computation, instead, it is  $\alpha_{\text{comp}} = 5\times$ . The final score is calculated as follows:

$$\text{Score} = (S_{\text{dist}} + 10 \cdot S_{\text{gates}}) \cdot \alpha_{\text{env}} \cdot \alpha_{\text{comp}} \quad (1)$$

where  $S_{\text{dist}}$  is the total traveled distance and  $S_{\text{gates}}$  is the number of times the drone passes through the gates. Time counting is never stopped (except for battery swaps): crashing, re-flashing, rebooting, components replacements, and other flight interruptions are allowed, but they will ultimately penalize the score. The teams can choose the starting position of the drone, but after a flight interruption, subsequent take-offs must occur near the location where the flight was interrupted.

**Robotic platform:** all teams competed with the same robotic platform comprising a COTS Bitcraze Crazyflie v2.1 drone and two additional onboard modules, as shown in Fig. 2-B. The drone is a modular 27 g 10 cm quadrotor integrating sensing, communication, and actuation subsystems with an STM32F405 MCU for sensing, state estimation (through an extended Kalman Filter) and control. The UAV can fly up to  $\sim 7$  min employing a 250 mA h battery. The first additional module is the Flow-deck, a small ( $21 \times 28$  mm) printed circuit board (PCB) that provides additional sensors to improve the state estimation: a VL53L1x Time-of-Flight distance sensor for measuring the height from the floor and a PMW3901 low-resolution ( $35 \times 35$  px) optical-flow camera.

The second expansion board employed is the AI-deck, a PCB for vision-based onboard processing. The module embeds *i*) an Himax HM01B0 low-power ( $\sim 4$  mW) QVGA monochrome image sensor camera, *ii*) a NINA-W102 MCU supporting Wi-Fi and Bluetooth communications, and *iii*) GAP8, an ultra-low-power (ULP) SoC capable of efficiently managing computation-intensive workloads by employing hardware-enabled (8-core cluster of processing cores) parallel programming paradigms. The SoC also features standard interfacing peripherals (SPI, I2C, UART) as well as different hierarchically organized SRAM memories (L1:64 kB L2:512 kB).

**Simulator:** the IMAV challenge organizers provided all teams with a simulator developed and released by Bitcraze, based on the open-source *Webots* robot simulator. The simulator includes a simple PID controller for the attitude control of a Crazyflie quadcopter. It emulates the  $87^\circ$  field of view of the drone's Himax camera, and it provides a photorealistic world model of the competition arena. Several obstacles in the simulator resemble the ones of the IMAV challenge, including an orange pole; an orange gate; three panels with a width of 1 m, 1.5 m, and 3 m, respectively; curtains; carpets; nets.

**Our strategy:** given IMAV's rules, to maximize the overall score, the nano-drone should: *i*) address two concurrent complex vision-based tasks, i.e., *obstacle avoidance* and *gate-based navigation*; *ii*) rely only on onboard computation; *iii*) operate in the most challenging environment. Relying on off-board computation can ease the computational burden. However, it would reduce the overall score due to the lower coefficient of difficulty and decrease the drone's reactivity due to the higher latency of the distributed system connected via Wi-Fi. Instead, individually addressing even one of the two tasks would represent a significant step forward in the SoA. Neither simulation-based obstacle avoidance nor gate-based navigation is still demonstrated in a challenging autonomous nano-drones race. At the same time, as previously discussed, the computational and memory limitations of embedded platforms such as GAP8 still constitute a strong holdback against

deploying multiple tasks simultaneously. Therefore, for the IMAV'22 competition, we prefer to address only one of these tasks while challenging our system with fully onboard computation ( $\alpha_{\text{comp}} = 5\times$ ) and maximum environmental difficulty ( $\alpha_{\text{env}} = 10\times$ ).

To choose which task to perform, we consider two ideal nano-drones *A* and *B* flying at a mean speed of 1.5 m/s in a thought experiment. *A* performs obstacle avoidance in the most complex environment ( $\alpha_{\text{env}} = 10\times$ ), while *B* addresses gate-based navigation in an obstacle-free scenario ( $\alpha_{\text{env}} = 1\times$ ). In a 5 min flight, the *A* system would score 7500 points. Assuming the two gates are 3 m apart, the *B* system would travel 750 m, passing through the gates 250 times, leading to a final score of 3250 points. Therefore, we believe that with the rule set of this competition, an autonomous nano-drone that quickly explores its surroundings and reliably avoids collision with dynamic obstacles (scenario *A*) has the potential to mark a higher score than a system designed for gate-based navigation in the simpler scenario *B*.

#### IV. NAVIGATION POLICIES

**Deep Neural Network:** the neural network used in this work is derived from the open-source PULP-Dronet CNN [9].

We keep the same network topology consisting of three consecutive residual blocks (ResBlocks). Each ResBlock comprises a primary branch that executes two  $3 \times 3$  convolutional layers and a parallel by-pass employing a  $1 \times 1$  convolutional layer. Unlike the previous work, whose outputs consisted of a single collision probability and a steering angle, our CNN features three collision probabilities by splitting the input image horizontally in three  $54 \times 162$  px left, center, and right portions of the FoV. We exploit only simulated data to train the network, and the data collection procedure is detailed in Sec. V. After training the network, we apply fixed-point 8-bit quantization to its weights to: *i*) reduce the CNN's memory footprint by  $4\times$ , resulting in a size of 317 kB, and *ii*) to enable optimized 8-bit fixed-point arithmetic, resulting in a throughput of 30 frame/s when deployed on GAP8. We define three navigation policies that rely on the CNN outputs: Baseline, Policy 1, and Policy 2.

**Baseline:** for each input image, the drone flies towards the direction with the lowest collision probability among the three CNN's outputs. In case all three probabilities are higher than a given threshold  $T_h$ , the drone spins in place by  $180^\circ \pm$  a random angle in the  $[0^\circ, 30^\circ]$  range. To generate the training labels for this model, only actual objects in the environment (i.e., panels, cylinders, walls, gates, etc.) are considered obstacles.

**Policy 1:** same as the baseline policy, with the difference that the training labels are generated considering as an obstacle also the ground outside the mission area, i.e., black and blue stripes in Fig. 2-A, as well as surrounding walls.

**Policy 2:** the three probabilities of collision are trained as in Policy 1, and if at least one of them is higher than a given threshold  $T$ , the drone will behave as in the previous two policies. Otherwise, if all probabilities of collision are lower than  $T_h$ , the drone will try to reach one of the four waypoints

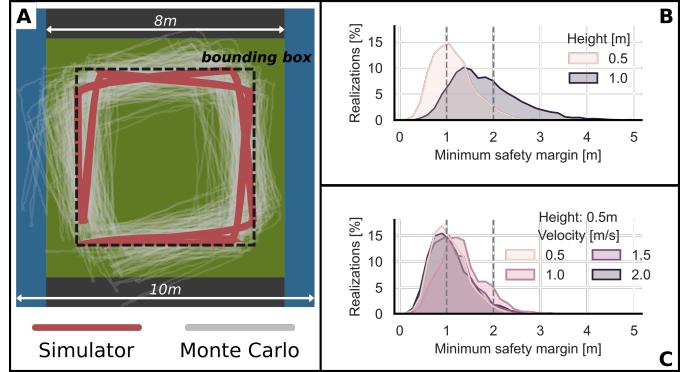


Fig. 3. A) 1024 simulated Monte Carlo trajectory realizations B) minimum safety margin w.r.t. height, C) minimum safety margin w.r.t speed, having fixed the height to 0.5 m.

(WPs) defined as the corners of a square inscribed in the mission area,  $WP_{1,2,3,4}$  in Fig. 2-A. This *WP-based navigation* mechanism leverages *i*) the *a priori* knowledge of the take-off point, i.e., we can choose the take-off point/orientation of the drone, for example, in  $WP_1$ ; and *ii*) the onboard visual-inertial state estimation which provides the relative position of the drone w.r.t. the take-off point. Then, when the drone is in the WP-based navigation mode, it will try to visit the WPs in a predefined cyclical order; for example,  $WP_{1,2,3,4}$  produces a counterclockwise motion, and  $WP_{4,3,2,1}$  a clockwise one. Lastly, to mark a WP as “visited,” we define a circular area of radius  $r$  centered in each WP (in red in Fig. 2-A): when the drone enters such an area, we mark the corresponding WP as visited and continue with the next one. Whenever all three collision probabilities are higher than the threshold  $T_h$ , we invert the sequence of the targeted waypoints, alternating a clockwise and counterclockwise direction. This mechanism also serves as an escape strategy when a WP is unreachable.

**Rationale discussion:** autonomous drones flying with *a priori* knowledge of the environment, i.e., a map, are highly effective as we can plan for the best trajectory. Unfortunately, the ultra-constrained resources (memory, computation, and sensors) aboard a nano-drone make this map-based navigation unfeasible [14] or extremely limited [13]. However, in the IMAV'22 competition, we can exploit coarse-grained information, as we know the size of the mission area, the colors of the ground, and – up to some degree – the shape/textured/colors of the objects thanks to the simulator provided by the organizers. Since obstacles and gates are more likely to populate the inner part of the flying area rather than its borders, Policy 2 tries to maximize the likelihood of a straight obstacle-free path by positioning the WPs at the edges of a square.

Therefore, ideally, we should place our four WPs precisely on the edges of the green square in Fig. 2-A. However, the unavoidable noise/drift of the simple state estimation aboard our nano-drone would often drive the vehicle outside the mission field, negatively impacting the final score. For this reason, we introduce the WPs as the edges of an inner square within the available  $8 \times 8$  m mission area.

**State estimation analysis:** we characterize the position and orientation error of the Crazyflie's state estimation subsystem.

This allows us to determine a margin between the WPs and the corners of the mission area that *i*) keeps the drone flying close to the edges of the area and *ii*) minimizes the risk that state estimation drift drives the drone out of the area. We start by collecting real-world flight data, which we use to build a statistical model. We rely on a setup similar to IMAV competition with artificial turf on the floor, albeit over a smaller  $5 \times 6$  m flying area. We record the drone's movements with a motion-capture system and the drone's onboard state estimation, both at 100 Hz. We perform runs of 5 min each, in which the drone flies using a random walk policy while constrained through the mocap into an empty squared area – smaller than our  $5 \times 6$  m arena. We perform 16 flights: two runs for each configuration combining two flight heights, i.e., 0.5 and 1.0 m, and four mean speeds, i.e., 0.5, 1.0, 1.5, 2.0 m/s.

Similarly to [23, Sec. 5.2.4], we quantify the state estimation error by modeling the uncertainty on incremental state estimation updates. We sample many ten-second time windows from the collected data, considering the relative pose of the drone at the beginning and the end of each window. We measure the estimation error separately for each component of the relative pose:  $x$ ,  $y$ , and yaw. For each component, we fit a Gaussian distribution to the errors measured over many windows, then we rescale the distribution to represent the error accumulating in 1 second.

Then, in the Webots simulator, we collect eight runs, i.e., all combinations of heights and speeds, of 5 min each (red lines in Fig. 3-A). The drone follows a squared trajectory on a  $6 \times 6$  m square, which is intuitively a safe configuration. We invert the flown path for each lap by alternating clockwise/counterclockwise directions to ensure that any systematic errors in yaw estimation (e.g., a consistent under/over-estimation of the amount of rotation) cancel out rather than accumulate. We use these eight flown trajectories to build a *bounding box* enclosing them (dashed black line in Fig. 3-A). We apply the statistical error model to corrupt the ideal squared trajectory, computing 1024 Monte Carlo (M. C.) realizations of the drone's state estimation (pale gray lines in Fig. 3-A). Then, only considering the portion of the M. C. trajectories outside the bounding box, we measure the maximum distance between them and the box itself. This measure gives us a statistical *minimum safety margin* distribution that will likely constrain the flight within the desired flight area.

Fig. 3-B shows (average on all speeds) that flying at 0.5 m height requires a lower safety margin than at 1.0 m, centering the distribution at 1.08 m and 1.56 m, respectively. Therefore, we select 0.5 m as our target height, and we focus only on this configuration in Fig. 3-C, where we explore the four velocities. Among all velocities, 42% of the 1024 realizations never exceed a safety margin of 1 m, while 95.3% satisfy a margin of 2 m. Furthermore, the median time spent outside the 1 m margin is 2 s (95<sup>th</sup> percentile: 31 s) per realization. Finally, of the 4.7% of realizations that cross the margin of 2 m, the median number of crossings is only one per realization (95<sup>th</sup> percentile: 5). These findings confirm that positioning our WPs at the edges of a  $6 \times 6$  m square will *i*) keep the drone in the desired  $8 \times 8$  m mission area on average for 97.6% of the 5 min run's time; and *ii*) cause the drone to exceed the maximum

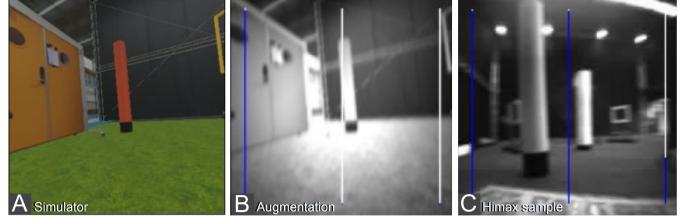


Fig. 4. Images: A) from the simulator, B) after augmentation, C) Himax camera sample collected in the IMAV arena. The three blue bars in images B-C represent the three collision probabilities predicted by our network.

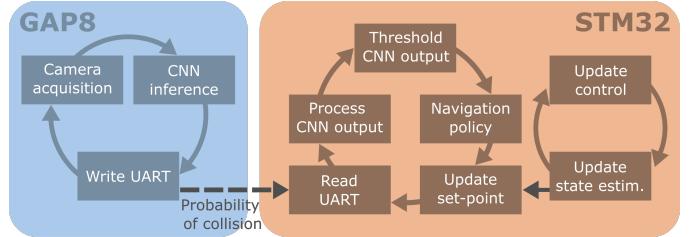


Fig. 5. The drone's state machines mapped on the MCUs available aboard.

$10 \times 10$  m space, *i.e.*, crash, only with 4.7% probability and only once per run. Finally, the four flight speeds analyzed do not significantly impact the safety margin requirements, which leads us to use the highest speed: 2.0 m/s.

## V. CNN TRAINING AND DEPLOYMENT

**Dataset collection:** we use the Webots simulator, shown in Fig. 6-A, to collect a dataset of images for our CNN. To automatically generate the labels for each image, we extended the simulator's capabilities by introducing the generation of depth and segmentation frames from the camera. During the image collection, we use a flight height of 0.5 m, see Sec. IV. Furthermore, to mimic the dynamic effects of an actual UAV flight, we add random variations for pitch, roll, and yaw in a  $[-5^\circ; +5^\circ]$  range and for the height in a  $[0.45; 0.55]$  m range.

We collect images with a  $324 \times 324$  px resolution and  $87^\circ$  FoV, the same as our Himax camera. For each image, we save per-pixel depth and segmentation masks, which we exploit for labeling: we divide the camera FoV into three vertical portions of  $108 \times 324$  px, and for each portion, we set a collision label = 1 if the 10% of the pixels belonging to obstacles has distance  $\leq 2$  m, 0 otherwise. We consider all kinds of objects as obstacles, *including gates* and excluding carpets. We collected 41 k images in 3 simulated scenarios to ensure labels balancing and uniform sample distribution. 10 k images are collected by spawning the drone randomly across the arena populated by obstacles, and 21 k images are acquired by flying around each obstacle in the simulator. This last group comes from a "360 scan" of each obstacle, which we equally split between images with an empty background and a populated one. Finally, 10 k images are taken from the drone flying along a square trajectory, 50 cm within the green flying area edges, equally split between clockwise and counterclockwise flight directions. Eventually, we split each portion of the dataset using 70% of the images for training, 10% for validation, and 20% for testing.

TABLE II

NEURAL NETWORK ACCURACY BY PHOTOMETRIC AUGMENTATION  
METHOD: NOISE (N), BLUR (B), AND EXPOSURE (E).

Aug.	N	<i>None</i>	B	B+E	N+B	E	N+E	N+B+E
Acc.	55%	56%	58%	62%	64%	68%	69%	72%

**Photometric augmentations:** as the simulated images are noise-free 3D renderings of the scene, we introduce an aggressive data augmentation pipeline to bridge the appearance gap with real-world Himax images. We randomly perturb the simulated images to reproduce several real-world image artifacts: motion blur, Gaussian blur to simulate lens defocusing, Gaussian noise, and exposure changes (gain, gamma, dynamic range, and vignetting). Finally, we convert images to grayscale and resize them to  $162 \times 162$  px, as our Himax camera would do. Fig. 4-A-B and the supplementary video show the result of this photometric pipeline, while Fig. 4-C displays a real drone frame. We assess the impact of the photometric augmentations on CNN accuracy by exploring all eight combinations of exposure, blur, and noise augmentations. To evaluate accuracy, we collected 2200 real-world images from our  $5 \times 6$  m indoor flying arena, shown in Fig. 6-B. The results in Tab. II shows that enabling all augmentations scores the highest accuracy, surpassing the non-augmented model by 17%.

**Navigation policies implementation:** Fig. 5 shows the state machines mapped on the two MCUs available aboard our UAV. GAP8 implements a loop for *i*) image acquisition, *ii*) CNN inference, and *iii*) UART transmission of the 3 CNN collision probability outputs, i.e.,  $\{\text{left}, \text{center}, \text{right}\}$ , to the STM32. The STM32 instead implements two loops for high and low-level control of the drone, respectively. The high-level control loop *i*) reads the CNN's output from UART, *ii*) converts the 8-bit fixed-point CNN's outputs ( $[0, 255]$  range) to float32 numbers ( $[0, 1]$  range) and applies a low pass filter, *iii*) thresholds the three probabilities of collision to  $T_h = 0.7$ , getting values  $\in \{0, 1\}$ , *iv*) applies the navigation policy to calculate the next set point, and *v*) updates the low-level controller. Instead, the low-level control loop, running at 100 Hz, updates the state estimation through the eKF and applies a cascade of PID controllers to reach the set point pushed by the high-level control loop.

All three navigation policies described in Sec. IV output a  $\{\text{speed}, \text{yaw rate}\}$  tuple as a set point. For all policies, the forward speed is inversely proportional to the *center* collision probability. The yaw rate is set to -90 or +90 deg/s when the right or the left collision probabilities get over the threshold  $th$ , respectively. If all three collision probabilities exceed the threshold  $T_h$ , the drone spins in place as described in Sec. IV. Conversely, the three policies act differently when all collision probabilities are zero: Baseline and Policy 1 command the drone to fly in a straight line, while Policy 2 activates the WP-based navigation mode, heading the drone to the next WP.

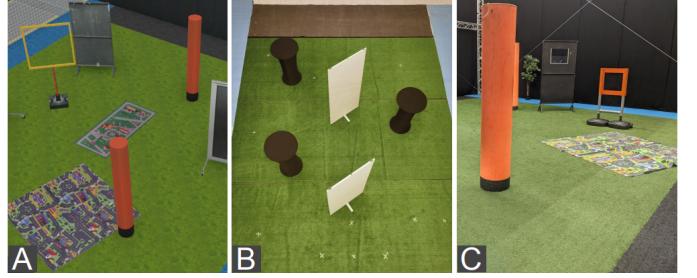


Fig. 6. Three environments: A) the *Webots* simulator, B) our indoor  $5 \times 6$  m arena, and C) the  $8 \times 8$  m competition arena.

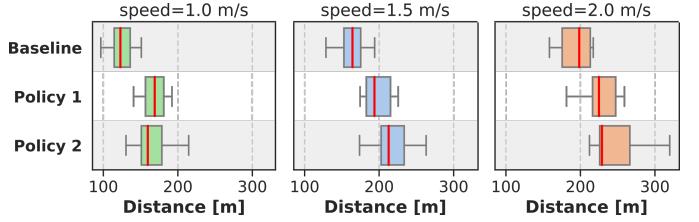


Fig. 7. Median traveled distance (10 runs) in 5 min flights on the simulator of our three navigation policies at 1.0, 1.5, and 2.0 m/s.

## VI. EXPERIMENTAL RESULTS

### A. In-simulator evaluation

We evaluate our three navigation policies with the Webots-based simulator, sample picture shown in Fig. 6-A. We rely on an ideal flight controller inside the simulator, meaning *i*) no state-estimation drift and *ii*) exploiting depth maps for ideal obstacle avoidance. We tested nine configurations combining the three control policies introduced in Sec. IV and at three speeds (1, 1.5, and 2 m/s). For all the tests, we used seven obstacles: two orange poles, two gates, and three panels with a width of 1, 1.5, and 3 m, respectively. We run ten experiments of 5 min each for each configuration, varying the obstacles' initial position but keeping it consistent among the nine configurations. To emulate the dynamic obstacles of the challenge, we move one obstacle every 30 s. We compare the policies based on the distance traveled on the green turf.

Fig. 7 shows that the baseline policy (which does not consider the floor outside the green turf as an obstacle) is consistently worse than the others, spending 16% of the time outside the mission area. Conversely, Policy 1 and 2 detect the external area as an obstacle, spending only 1% and 0.8% of the time on the external area, respectively. Both Policy 1 and 2 benefit from higher flight speeds due to the simulator's perfect sensing preventing collisions. At the lowest speed configuration of 1 m/s, Policy 2 achieves a slightly lower median distance than Policy 1, 169 m vs. 169 m. Instead, for higher speed configurations, the median distance of Policy 1 is slightly lower than Policy 2: 217 m and 246 m at 1.5 m/s, and 193 m and 225 m at 2 m/s, respectively. Since both Policy 1 and Policy 2 score a similar traveled distance, we push forward our analysis by deploying and testing both of them in the field.

### B. State-of-the-Art comparison

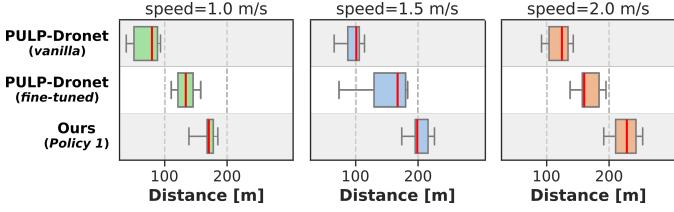


Fig. 8. Median traveled distance (10 runs) in 5 min flights on the simulator comparing our work to the SoA PULP-Dronet [9] at 1.0, 1.5, and 2.0 m/s.

Since our CNNs are inspired by the PULP-Dronet [9], as well as many other teams at the competition, we present a thorough comparison of our system against a *vanilla* PULP-Dronet and a fine-tuned version of it. Given the different outputs between our CNN and the original one, we adapt our navigation Policy 1 (no WPs) to the PULP-Dronet baselines. We use the PULP-Dronet steering output, as in its original implementation, by converting it in a yaw-rate for the controller in case of a probability of collision lower than 0.7, while the drone rotates 180° otherwise. For the fine-tuned version, we use the same dataset collected in simulation to train our models. The single collision label of PULP-Dronet is matched to the central probability of our models, while the steering angle reflects the direction with the lower probability of collision among the three probabilities: left, center, and right.

The results in Fig. 8 are collected in simulation, where we run ten 5-min tests for each model at three target flight speeds (i.e., 1, 1.5, and 2 m/s). The vanilla PULP-Dronet marks the lowest score in all configurations, 84–114% less than our system, while the fine-tuned PULP-Dronet reduces this gap with a performance reduction of 28%–43%, depending on the flight speed. Our network scores a median distance traveled of 171, 199, and 228 m at 1, 1.5, and 2 m/s, respectively, showing the benefit of our architectural changes and training methodology.

### C. In-field evaluation

We evaluate Policy 1 and Policy 2 in our 5×6 m indoor flying arena, testing them with two target speeds, 1.5 and 2 m/s, resulting in four configurations. For each configuration, we perform 5 runs of 5 min each. We use three black cylinders and two white panels as obstacles, as shown in Fig. 6-B. We replicate the competition’s conditions, e.g., periodically moving the dynamic obstacles (every ~30 s), and without stopping the time count if the drone crashes. As shown in Fig. 9-A, Policy 1 scores a median 45 and 65 m, while Policy 2 marks a median 91 and 92 m, while flying at 1.5 and 2 m/s, respectively. The improved performance of Policy 2 derives from the *WP-based navigation*, which provides the drone with boundaries, limiting the time spent in potentially dangerous areas outside the mission field. A sample run of Policy 2 is shown in Fig. 9-B. For the IMAV competition, we choose to use the Policy 2 at 1.5 m/s for the first run (conservative) and push our system to the 2 m/s limit in our second attempt.

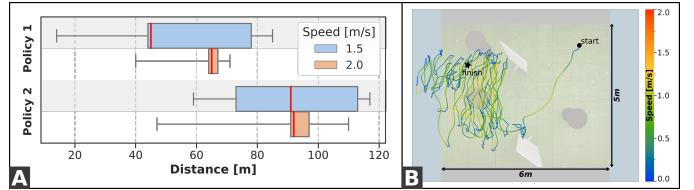


Fig. 9. In-field testing, A) median traveled distance (5 runs) in 5 min flights of our two navigation policies at 1.5 and 2.0 m/s. B) Sample run of Policy 2 with a  $v_{target} = 1.5$  m/s, scoring 117 m of traveled distance.

### D. The nanocopter AI challenge

The arena for the competition was a 10×10 m area, represented in Fig. 2-A, where only the 8×8 m green surface is considered for the final score. The arena had ten objects (Fig. 6): four orange poles, two black panels, two flags, and two gates. The competition allows each team to perform two runs of 5 min each, having only the best one considered in the final leaderboard. Six teams participated, and the video recording is available online<sup>1</sup>. For each team, Tab. III summarizes the distance traveled, the number of gates passed, and the final score computed according to Eq. 1.

The last team achieved a traveled distance of 9 m in an obstacle-free environment while employing an off-board (i.e., WiFi-connected laptop) color segmentation algorithm. The 5<sup>th</sup> ranked team tackled an obstacle avoidance task using an off-board CNN for monocular depth estimation. They exploited the depth to control the drone’s forward speed and steering angle commands, ultimately reaching a traveled distance of 67 m. The 4<sup>th</sup>-classified team proposed a system that passed through four gates, i.e., the highest number of traversed gates among all teams. They tackled gate-based navigation with an off-board color segmentation visual pipeline. However, their final traveled distance of 31 m was penalized by numerous crashes, wasting precious time for resuming the flight.

The team ranked 3<sup>rd</sup> used an onboard vanilla PULP-Dronet for obstacle avoidance, which led to a conservative ~0.1 m/s average speed on their best 5-min run (*run 1*). Nevertheless, they managed to pass through one gate while crashing multiple times (up to five in *run 2*), resulting in a traveled distance of 29 m. The second classified team exploited only onboard computation in a static obstacles environment. Their navigation strategy limited the exploration to a small obstacle-free part of the environment, leading to a total distance of 81 m.

We tackled the most challenging scenario, i.e., dynamic obstacles, with only onboard computation (i.e., no WiFi-connected laptop), while leveraging our Policy 2, described in Sec. IV. In our two runs, we set a maximum target speed of 1.5 2.0 m/s, which resulted in a traveled distance of 115 and 97 m, respectively. During the first and best run, we never crashed for the entire 5-minute flight, resulting in the winning score of 5750 points, i.e., almost 3× more than the second-classified team. Fig. 10 summarizes our two runs, reporting a top-view of the arena where flight trajectories are shown with their mean average speed.

<sup>1</sup><http://youtu.be/WaDU4I2TImA>

TABLE III  
THE FINAL LEADERBOARD OF THE “NANOCOPTER AI CHALLENGE.”

Team	Processing	Obstacles	[m] / Gates	Score
PULP (ours)	Onboard	Dynamic	115 / 0	5750
Black Bee Drones	Onboard	Static	81 / 0	2015
SkyRats	Onboard	Dynamic	29 / 1	1945
CVAR-UPM	Off-board	Dynamic	31 / 4	702
CrazyFlieFolder	Off-board	Dynamic	67 / 0	666
RSA	Off-board	No	9 / 0	9

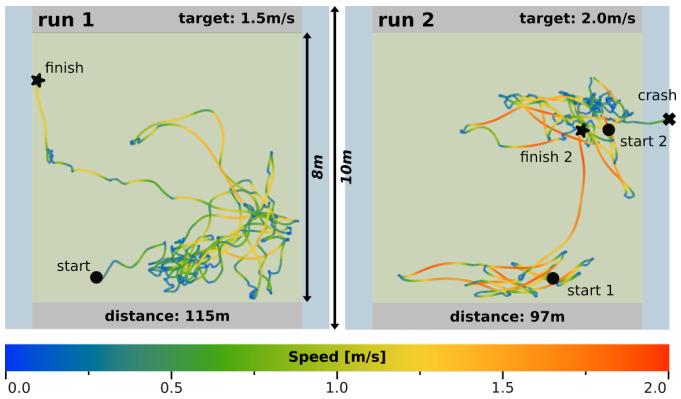


Fig. 10. Trajectories of our two runs during the IMAV’22 race: run 1 ranked our team 1<sup>st</sup> at the “Nanocopter AI challenge.”

## VII. CONCLUSION AND FUTURE WORK

We present the deep learning framework for visual-based autonomous navigation aboard nano-drones, winning the “IMAV’22 Nanocopter AI Challenge” drone race. Our system combines a CNN for obstacle avoidance, trained only in simulation, a sim-to-real mitigation strategy, and a navigation policy, defining the drone’s control state machine. Our system scored 115 m of traveled distance at the competition while coping with static and dynamic obstacles. Since we focused on the most challenging obstacle avoidance task while leaving out the gate-based navigation task, future work will address developing lightweight perception modules for both tasks. Nevertheless, our system marks the SoA being the first example of an autonomous nano-drone completing its mission (5 min flight, at 1.5 m/s with no crashes) in a challenging never-seen-before race environment. Additionally, any bigger robot can exploit our lightweight yet accurate and reactive perception, freeing a vast amount of computational resources and memory that can be allocated to tackle additional complex tasks. By sharing our knowledge, we foster the research by providing a solid foundation for future development in the newborn field of nano-drone racing.

## ACKNOWLEDGMENT

We thank the Autonomous Robotics Research Center of the Technology Innovation Institute, Abu Dhabi, UAE, for granting us access to their indoor flying arena. We also thank the organizers of the IMAV’22 conference for sharing with us the trajectory recordings.

## REFERENCES

- C. D. Wagter *et al.*, “The sensing, state-estimation, and control behind the winning entry to the 2019 Artificial Intelligence Robotic Racing Competition,” *Field Robotics*, vol. 2, no. 1, pp. 1263–1290, mar 2022.
- P. Foehn *et al.*, “Alphapilot: Autonomous drone racing,” *Autonomous Robots*, vol. 46, no. 1, pp. 307–320, 2022.
- S. Jung *et al.*, “A direct visual servoing-based framework for the 2016 IROS Autonomous Drone Racing Challenge,” *Journal of Field Robotics*, vol. 35, no. 1, 2018.
- D. Hanover *et al.*, “Autonomous Drone Racing: A Survey,” *ArXiv*, vol. abs/2301.01755, 2023.
- A. Birk *et al.*, “Safety, Security, and Rescue Missions with an Unmanned Aerial Vehicle (UAV),” *Journal of Intelligent & Robotic Systems*, vol. 64, no. 1, pp. 57–76, Oct. 2011.
- D. Palossi *et al.*, “Fully Onboard AI-Powered Human-Drone Pose Estimation on Ultralow-Power Autonomous Flying Nano-UAVs,” *IEEE Internet of Things Journal*, vol. 9, no. 3, pp. 1913–1929, 2021.
- E. Kaufmann *et al.*, “Beauty and the Beast: Optimal Methods Meet Learning for Drone Racing,” in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 690–696.
- H. X. Pham *et al.*, “PencilNet: Zero-Shot Sim-to-Real Transfer Learning for Robust Gate Perception in Autonomous Drone Racing,” *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 11 847–11 854, Oct. 2022.
- V. Niculescu *et al.*, “Improving Autonomous Nano-drones Performance via Automated End-to-End Optimization and Deployment of DNNs,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, pp. 1–1, 2021.
- E. Flamand *et al.*, “Gap-8: A risc-v soc for ai at the edge of the iot,” in *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2018, pp. 1–4.
- D. Palossi *et al.*, “An Open Source and Open Hardware Deep Learning-powered Visual Navigation Engine for Autonomous Nano-UAVs,” in *2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS)*. IEEE, 2019, pp. 604–611.
- W. Zhao *et al.*, “Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: a Survey,” in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2020, pp. 737–744.
- B. Bodin *et al.*, “SLAMBench2: Multi-Objective Head-to-Head Benchmarking for Visual SLAM,” in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 3637–3644.
- H. Zhou *et al.*, “Efficient 2D Graph SLAM for Sparse Sensing,” in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022, pp. 6404–6411.
- O. Dunkley *et al.*, “Visual-Inertial Navigation for a Camera-Equipped 25g Nano-Quadrotor,” in *Proc. IROS Aerial Open Source Robot. Workshop*, 2014, pp. 1–2.
- K. McGuire *et al.*, “Minimal Navigation Solution for a Swarm of Tiny Flying Robots to Explore an Unknown Environment,” *Science Robotics*, vol. 4, p. eaaw9710, Oct. 2019.
- L. Lamberti *et al.*, “Bio-inspired Autonomous Exploration Policies with CNN-based Object Detection on Nano-drones,” in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2023, pp. 1–6.
- N. Elkunchwar *et al.*, “Bio-Inspired Source Seeking and Obstacle Avoidance on a Palm-Sized Drone,” in *2022 International Conference on Unmanned Aircraft Systems (ICUAS)*, Jun. 2022, pp. 282–289.
- K. McGuire *et al.*, “Efficient Optical Flow and Stereo Vision for Velocity Estimation and Obstacle Avoidance on an Autonomous Pocket Drone,” *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 1070–1076, Apr. 2017.
- P. Foehn *et al.*, “Time-Optimal Planning for Quadrotor Waypoint Flight,” *Science Robotics*, vol. 6, no. 56, p. eabhl221, Jul. 2021.
- S. Li *et al.*, “Visual Model-Predictive Localization for Computationally Efficient Autonomous Racing of a 72-g Drone,” *Journal of Field Robotics*, vol. 37, no. 4, pp. 667–692, 2020.
- D. Palossi *et al.*, “Target Following on Nano-Scale Unmanned Aerial Vehicles,” in *2017 7th IEEE International Workshop on Advances in Sensors and Interfaces (IWASI)*, Jun. 2017, pp. 170–175.
- R. Siegwart *et al.*, *Introduction to autonomous mobile robots*. MIT press, 2011.