Distilling Tiny and Ultrafast Deep Neural Networks for Autonomous Navigation on Nano-UAVs

Lorenzo Lamberti, *Member, IEEE*, Lorenzo Bellone, Luka Macan, Enrico Natalizio, *Senior Member, IEEE*, Francesco Conti, *Member, IEEE*, Daniele Palossi, and Luca Benini, *Fellow, IEEE*

***Abstract*—Nano-sized unmanned aerial vehicles (UAVs) are ideal candidates for flying Internet-of-Things smart sensors to collect information in narrow spaces. This requires ultrafast navigation under very tight memory/computation constraints. The PULP-Dronet convolutional neural network (CNN) enables autonomous navigation running aboard a nano-UAV at 19 frame/s, at the cost of a large memory footprint of 320 kB and with drone control in complex scenarios hindered by the disjoint training of collision avoidance and steering capabilities. In this work, we distill a novel family of CNNs with better capabilities than PULP-Dronet, but memory footprint reduced by up to 168× (down to 2.9 kB), achieving an inference rate of up to 139 frame/s; we collect a new open-source unified collision/steering 66 k images data set for more robust navigation; and we perform a thorough in-field analysis of both PULP-Dronet and our tiny CNNs running on a commercially available nano-UAV. Our tiniest CNN, called Tiny-PULP-Dronet v3, navigates with a 100% success rate a challenging and never-seen-before path, composed of a narrow obstacle-populated corridor and a 180° turn, at a maximum target speed of 0.5 m/s. In the same scenario, the State-of-the-Art PULP-Dronet consistently fails despite having 168× more parameters.**

***Index Terms*—Artificial intelligence (AI), autonomous nano-unmanned aerial vehicle (UAV), embedded devices, mobile and ubiquitous systems, ultralow power.**

I. Introduction

With the growth of the Internet of Things (IoT), autonomous nano-sized unmanned aerial vehicles (UAVs) empowered with onboard artificial intelligence (AI) are becoming increasingly important [6]. With a diameter of less than 10 cm and weighing only tens of grams, these nano-UAVs can serve as ubiquitous IoT nodes, autonomously navigating environments while sensing and analyzing their surroundings [7]. Their compact form factor allows them to operate in narrow/cluttered spaces [5], [8] and safely in the proximity of humans [9] (as shown in Fig. 1). These characteristics enable nano-UAVs to be suitable in many use cases, such as exploring hazardous indoor environments [10] and in rescue missions [11], [12].

Fig. 1. Our autonomous nano-UAV navigating an unknown environment.

To act as autonomous smart IoT sensors, nano-UAVs must execute concurrent real-time tasks entirely onboard, including multiple heavy AI perception workloads [4], [5], [9], [13], [14]. However, achieving such a high level of autonomy presents significant challenges. Their extremely limited payload restricts them to accommodate only ultra-low-power microcontroller units (MCUs) that have stringent computational and memory constraints, e.g., a few hundred kB of on-chip memory and few giga operations per second (GOps/s). These constraints have prevented the deployment of multiple AI tasks on nano-UAVs. Therefore, overcoming the computational/memory burden imposed by MCUs becomes paramount. We focus on optimizing and minimizing the AI workloads without compromising the drone's behavior when stressed in real-world testing scenarios.

We specifically address a critical AI task required by any IoT node moving within an environment: *visual navigation*, i.e., the capability to autonomously navigate an environment by relying solely on local visual information while avoiding

Manuscript received 28 March 2024; revised 31 May 2024 and 24 June 2024; accepted 5 July 2024. Date of publication 22 July 2024; date of current version 9 October 2024. *(Corresponding author: Lorenzo Lamberti.)*

Lorenzo Lamberti, Luka Macan, and Francesco Conti are with the Department of Electrical, Electronic and Information Engineering, University of Bologna, 40126 Bologna, Italy (e-mail: lorenzo.lamberti@unibo.it; luka.macan@unibo.it; f.conti@unibo.it).

Lorenzo Bellone is with the Autonomous Robotics Research Center, Technology Innovation Institute, Abu Dhabi, UAE (e-mail: lorenzo.bellone@tii.ae).

Enrico Natalizio is with the Autonomous Robotics Research Center, Technology Innovation Institute, Abu Dhabi, UAE, and also with the Centre national de la recherche scientifique, LORIA, University of Lorraine, 54506 Nancy, France (e-mail: enrico.natalizio@loria.fr).

Daniele Palossi is with the Dalle Molle Institute for Artificial Intelligence, USI-SUPSI, 6962 Lugano, Switzerland, and also with the Integrated Systems Laboratory, ETH Zürich, 8092 Zürich, Switzerland.

Luca Benini is with the Department of Electrical, Electronic and Information Engineering, University of Bologna, 40126 Bologna, Italy, and also with the Integrated Systems Laboratory, ETH Zürich, 8092 Zürich, Switzerland (e-mail: lbenini@iis.ee.ethz.ch).

Digital Object Identifier 10.1109/JIOT.2024.3431913

TABLE I
RELATED WORKS BASED ON CNNS, RUNNING ABOARD NANO-UAVS. THE TASK IS EITHER CLASSIFICATION (C) OR REGRESSION (R)

Work	Device	Description	Task	Power [mW]	Parameters	Operations [MAC]	Throughput [inference/s]
Kooi and Babuška [1]	STM32F405	RL for landing	R	250*	4.7 k	4.7 k	400
Shi *et al.* [2]	STM32F405	Interaction forces	R	250*	9.5 k	9.5 k	—
Cereda *et al.* [3]	GWT GAP8	Pose estimation	R	90	65 k	7.4 M	51
Lamberti *et al.* [4]	GWT GAP8	Object detection	C+R	134	4.7 M	534 M	1.6
Niculescu *et al.* [5]	GWT GAP8	Navigation	C+R	102	320 k	41 M	19
Ours	**GWT GAP8**	**Navigation**	**C+R**	**100**	**2.9 k**	**1.1 M**	**139**

*estimated

collisions with obstacles. The State-of-the-Art (SoA) convolutional neural network (CNN) for visual-based autonomous navigation on nano-UAVs is PULP-Dronet v2 [5], outputting a collision probability and a steering angle following visual cues in an environment, such as lanes. This CNN has been deployed on a nano-sized UAV and demonstrated in both indoor and outdoor environments, enabling turns and avoiding collisions with dynamic obstacles.

However, this CNN has some shortcomings. First, its navigation capabilities come at a nonnegligible computational cost, allowing for a maximum throughput of 19 frame/s when running on the SoA Greenwaves Technologies (GWT) GAP8 System-on-Chip (SoC) [5]. Moreover, we show that this CNN performs poorly when navigating static obstacles, limiting its real-world applicability. This limitation stems from the training data set featuring disjoint image sets for steering and collision labels [5]. A solution to this issue is to collect new data that integrates joint information on obstacle presence and yaw rate into all the training images.

In this article, we set a new milestone in the SoA of autonomous visual-based nano-UAV navigation by shrinking the CNN memory footprint and computational burden while simultaneously improving its ability in static obstacle avoidance. Our key contributions are as follows.

1) We develop a methodology for data set collection on a nano-UAV. We collect *unified collision avoidance and steering* information only with nano-UAV onboard resources, without dependence on external infrastructure like motion capture systems. The resulting *PULP-Dronet v3* data set consists of 66 k labeled images, which we release open-source along with our data set collection framework.
2) We train and deploy a novel family of CNNs, i.e., PULP-Dronet v3, that enables *visual-based static obstacle avoidance and autonomous navigation* on nano-UAVs.
3) We perform an extensive study for minimizing the memory footprint and computational complexity of CNNs for autonomous nano-UAV navigation. The resulting CNNs offer different tradeoffs between in-field performance and model size.

We validate all our CNNs on our collected data set and characterize their end-to-end execution time on the GAP8 SoC. Our tiniest CNN, i.e., Tiny-PULP-Dronet v3, has only 2.9 k parameters and 1.1 M multiply-accumulate (MAC) operations, 168× smaller and 7.3× faster (up to 139 frame/s) than the baseline CNN, i.e., the SoA PULP-Dronet v2 [5], when running on the same parallel ultralow-power (PULP) GAP8 SoC. This outcome allows us to free up precious computational resources that can be exploited to tackle additional tasks onboard. This CNN only requires 0.7 mJ for each inference, with an average power consumption of 100 mW when running on the GAP8's cores at their maximum frequencies.

Furthermore, we deployed and extensively tested in-field two CNNs: 1) a larger one (PULP-Dronet v3), matching the size of the baseline CNN and 2) our tiniest distilled CNN (Tiny-PULP-Dronet v3). These tests evaluate how reducing the computational CNN workload impacts the navigation performance. In a challenging scenario with a narrow corridor containing four static obstacles and a 180° turn, both CNNs outperformed the baseline one, as shown in the supplementary video. At a target speed of 0.5 m/s, our large and tiny CNNs successfully navigate through the corridor with a 80% and 100% success rate, respectively, while the baseline CNN consistently fails. At higher speeds, the larger CNN demonstrated better robustness, succeeding 60% of the time at a target speed of 1 m/s, whereas the tiny CNN failed. We foster the research on autonomous nano-UAV navigation by releasing our new data set, data set collection framework, CNN weights, and code as open-source.

The remainder of this work is organized as follows. Section II provides an overview of tinyML for autonomous nano-UAVs. Section III covers the background of our work. Section IV details our data set and collection methodology. Section V describes our CNN design/shrinking approach. Section VI compares the CNNs we introduced with experimental results. Section VII describes the in-field experiments with our nano-drone. Finally, Section VIII concludes this article.

II. RELATED WORK

A. TinyML on Autonomous Nano-Sized UAVs

This section presents an overview of lightweight and compact deep learning (DL) algorithms deployed on autonomous nano-UAVs, specifically nano-drones, which we summarize in Table I. To cope with the limited resources of nano-UAVs, multiple works in literature aim to achieve only minimal functionalities and/or rely on low-dimensional input signals (e.g., no images) to minimize execution workload [1], [2], [15].

In the work of Kooi and Babuška [1], a deep reinforcement learning method employing proximal policy optimization

enables autonomous landings of nano-drones on inclined surfaces. The CNN designed for this purpose requires around 4.5 kMAC operations per forward step. Although the CNN operates efficiently, computing an inference in approximately 2.5 ms on a single-core Cortex-M4 processor, its functionality remains constrained solely to the landing task. Similarly utilizing the Cortex-M4 processor, Neural-Swarm2 [2] leverages a controller based on DL for managing interaction forces during nano-drones' formation flights. With a processing cost of ∼9.5 kMAC, each nano-drone in the swarm processes the relative position and velocity data of surrounding UAVs, tackling safe maneuvers in close proximity with other UAVs.

To overcome computational constraints posed by single-core MCUs, multicore flight controllers designed for AI workloads address the limitations imposed by higher complexity DL-based workloads. The SoA MCU for commercial off-the-shelf (COTS) nano-UAVs is the GAP8 SoC, an embodiment of the PULP paradigm with a general-purpose 8-core parallel cluster. This SoC offers a peak throughput of 5.4 GOps/s [16] and 512 kB of on-chip memory. This fully programmable MCU was successfully exploited to enable the execution of more sophisticated visual-based AI workloads [4], [5], [9].

Cereda et al. [9] exploited GAP8 to demonstrate a fully autonomous nano-drone performing a human pose estimation task. Their CNN, called PULP-Frontnet, predicts the drone's relative pose with respect to a freely moving human subject. This prediction aims at maintaining a consistent distance in front of human subjects while following their movements, and it only exploits low-resolution grayscale images captured from a front-facing camera. Their model requires 14.7 MMAC and achieves an inference rate of 48 frame/s while consuming 96 mW. Cereda et al. [3] exploited neural architecture search techniques to design a tinier version of the PULP-Frontnet CNN, obtaining a 7.4 MMAC and 65 kB model, while retaining good regression performance when tested in-field. Like our work, their approach to CNN architecture design explores structures inspired by Mobilenet v2 [17].

Exploiting the same GAP8 SoC, Lamberti et al. [4] tackled a more complex high-level task for nano-UAVs: object detection, which has a computational complexity significantly greater (over an order of magnitude higher) than [3], [9]. They deployed a 4.7-MB CNN capable of detecting two classes of objects, demonstrating its application in the context of an exploration and search mission. Such workload requires a significant effort of 534 MMAC per inference and exceeds the SoC on chip L2 memory of 512 kB, introducing additional latency cost for transferring data between on-chip and off-chip memories. As a result, the system only achieved a throughput of 1.6 frame/s when tested onboard.

Zhang et al. [13] deployed a compute-intensive CNN with 310 k parameters for relative visual-based depth estimation on a nano-drone. The depth information from the front-facing grayscale camera is exploited for obstacle avoidance. However, the CNN runs onboard only at 0.2 frame/s and 1.2 frame/s on QVGA and QQVGA images, respectively, limiting the navigation capabilities of the drone to low speeds. Niculescu et al. [5] used instead a CNN for enabling end-to-end visual-based autonomous navigation on a nano-drone embedding the GAP8 SoC. The CNN, namely, PULP-Dronet v2, has a computational cost of ∼41 MMAC per inference and requires 320 kB of memory footprint. When tested on an autonomous nano-drone, this neural network allowed it to tackle turns and avoid dynamic obstacles appearing along the way.

However, PULP-Dronet v2 has two main limitations. First, as highlighted by Niculescu et al. [5], this CNN lacks the ability to guide the nano-UAV around static obstacles. This poses a significant limitation in its practical application within challenging real-world scenarios. This missing capability is attributed to the training data set of PULP-Dronet v2, as it comprehends multiple sets of images with disjointed labels for the steering and collision tasks [5]. The second limitation of this work is its nonnegligible workload of ∼41 MMAC, limiting its throughput to a maximum of 19 frame/s. Despite being enough to enable autonomous navigation, this system would greatly benefit from solving the same task with only a minimal fraction of the current workload, as it would free precious resources that can be exploited to tackle additional tasks [11], [12], [18]. Reference [18] demonstrates that the PULP-Dronet v2 CNN architecture is over-parameterized for the task it solves. However, it lacks an in-field demonstration that a smaller neural network can achieve sufficient accuracy for safe flight.

Therefore, in our work we move from PULP-Dronet v2, focusing on: 1) enabling new navigation skills, i.e., tackling the static obstacle avoidance scenario, demonstrating with in-field results that the data set was the limiting factor for achieving this task and 2) we enable autonomous navigation with a CNN that is 168× smaller than PULP-Dronet v2, leaving plenty of memory and computing resources that can be exploited to embed additional AI tasks on the nano-drone.

B. Data Sets for Nano-UAV Autonomous Navigation

As we highlighted the importance of introducing a new data set for visual-based autonomous nano-drone navigation in Section II-A, we review the data sets for UAVs in the literature. Dupeyroux et al. [19] released a data set for obstacle detection and avoidance, specifically targeting micro-sized drones (i.e., weighting ∼0.5 kg). It has 92 GB of data, including full-HD images for ∼ 80% of the acquisitions, and the drone pose is tracked with a motion capture system. Thus, adapting these images to the nano-drone use case would require a photometric augmentation pipeline to convert full-HD images to the format of a low-quality and low-resolution camera commonly found on nano-UAVs [5]. Conversely, we collect a data set exploiting nano-drones exclusively: we log the low-quality images from the onboard camera along with the pilot's yaw-rate input, which is fed to the flight controller during the data acquisition. This approach allows us to train an end-to-end CNN for autonomous navigation by exploiting the pilot's input commands rather than the UAV's external tracking.

The Dronet data set, introduced by Loquercio et al. [20], was created to enable autonomous navigation on large UAVs. They merged two distinct image sets: one with 39.1 k high-resolution images from car driving scenarios labeled only with

steering information and another with 32.2 k high-resolution images from bicycle rides labeled with binary collision information. This data set was expanded with approximately ∼ 1300 grayscale low-quality images collected from a nano-UAV and labeled only with binary collision information [5]. The combination of these three disjointed sets of images forms the Himax+Original data set used to train PULP-Dronet, which we exploit for comparison in Section VI.

However, this data set does not provide joint steering and collision labels for the navigation, leading to poor performance when tackling static obstacle avoidance [5], as detailed in Section II-A. Similarly, Chang et al. [21] collected ∼15 k images for solving the same task as Dronet [20]. However, this data set is also divided into two disjointed sets of images for steering and collision labels. Navardi et al. [22] used the Dronet data set to train an off-board CNN for nano-drones autonomous navigation, still not tackling the static obstacle avoidance scenario.

The data set presented in this article overcomes these limitations as all the ∼66 k collected images feature both collision and steering information, i.e., it does not have disjointed labels. Specifically, we log the input of a human pilot navigating a nano-drone in multiple environments. These labels can be used to train an end-to-end CNN that imitates the behavior of a human pilot. Furthermore, we log the drone's estimated state and label all the images with a binary label for obstacle avoidance. We open-source our data set, data set collector framework, and pretrained models to foster research on autonomous nano-drone navigation.

III. Nano-UAV Robotic Platform

A. Robotic Platform

The robotic platform employed in this work encompasses the Bitcraze Crazyflie 2.1 quadrotor, a commercially available nano-drone weighing 27 g and measuring 10 cm in diameter. This open-source and open-hardware drone uses the STM32F405 MCU as its flight controller coupled with the Bosch BMI088 inertial measurement unit (IMU), which combines an accelerometer and gyroscope. The STM32 MCU operates at speeds up to 168 MHz and integrates 48-kB static RAM (SRAM) and 128-kB flash memories, facilitating onboard inertial state estimation and actuation control. The IMU data drives an extended Kalman filter for state estimation at a rate of 100 Hz, while a proportional-integral-derivative control loop cascade manages actuation. This cascade comprises two control loops, with one governing attitude at 500 Hz and the second updating position at 100 Hz. The Crazyflie 2.1 also integrates a nRF51822 MCU for 2.4-GHz radio band communication, which we use only for the purpose of data set collection (Section IV-B).

Our configuration extends the robotics platform with two pluggable printed circuit boards (PCBs) from Bitcraze: 1) the flow deck and 2) AI-deck. The 3.5 g flow deck incorporates the PMW3901 optical flow visual sensor and the VL53L1x Time-of-Flight (ToF) ranging module. The optical flow sensor detects drone motion in multiple directions, while the ToF sensor measures distance from the ground. These inputs enhance onboard state estimation accuracy, minimizing long-term drift.

The second expansion board, the AI-deck, serves as the primary onboard computing unit responsible for executing heavy AI workloads, such as the proposed autonomous navigation CNNs. This PCB, weighing 4.4 g, integrates the GWT GAP8 SoC as the onboard computer for visual processing and AI-driven autonomous navigation. GAP8 is accompanied by onboard memories: 64 MB of HyperFlash and 8 MB of HyperRAM. The PCB also mounts a Himax HM01B0 ultralow-power monochrome QVGA camera, an ESP32-based Wi-Fi transceiver, and a UART communication channel between the STM32 and the GAP8. In this work, we focus on a configuration where the power-intensive Wi-Fi remains off, aiming for a fully autonomous system where all navigation intelligence resides onboard the nano-drone without relying on external communication or computation. However, we exploited the high bandwidth of the Wi-Fi communication for the data set collection, as detailed in Section IV-B.

For the sake of data set collection only (Section IV-B), we add an extra PCB to our drone, i.e., the *Multiranger deck*. This deck weighs 2.3 g and comprises five single-beam VL53L1x ToF distance sensors placed on the drone's top and lateral sides. These sensors provide line-of-sight distance measurements with a frequency of 20 Hz and have an operative range that goes from 0 m to 4 m.

B. GAP8 System-on-Chip

The GWT GAP8 SoC,[1] which is used in this work for all onboard vision and tinyML processing, is an ultralow-power SoC that combines the capabilities of a regular MCU, the fabric controller (FC) domain, with a fully programmable parallel accelerator called cluster (CL) that can yield up to 5.4 GOps/s [16] in a power envelope of ∼100 mW. The FC domain includes a low-power processor implementing the RISC-V `RV32IMCXpulpv2` instruction set architecture (ISA) based on the RI5CY design, paired with on-chip SRAM memory and peripherals supporting protocols like SPI, I2C, HyperBus, and camera parallel interface (CPI). These can be accessed through a dedicated direct memory access (DMA) engine called μDMA to offload the communication burden from the FC.

The CL accelerates heavier parallelizable workloads typical of digital signal processing and AI applications. It includes 8 RISC-V cores with the same ISA extensions as the FC; in particular, the `Xpulpv2` extension includes 8-bit and 16-bit packed single instruction multiple data (SIMD), MAC, and dot-product operations, which enhance the SoC's linear algebra capabilities for such workloads. The cores are connected over a logarithmic interconnect to 64 kB of L1 tightly coupled data memory (TCDM) comprising 16 memory banks. The logarithmic interconnect assures 1-cycle latency access for all the cores when there is no bank conflict, enabling fast data parallelism among the cores. The cluster has a DMA to offload data transfers between the TCDM and the 512-kB L2 memory.

[1] https://greenwaves-technologies.com/low-power-processor/

Fig. 2. (A) Our data set collection methodology, (B) our data set collector GUI, and (C) sample sequence from our collected data set.

The cores are programmed with the Single-Program Multiple-Data programming model and synchronized using dedicated hardware for low-latency barriers.

The FC and cluster domains are separately clocked to tune for best energy efficiency and performance tradeoff. The FC domain can be clocked between 50 MHz and 250 MHz, while the cluster domain can be between 100 MHz and 175 MHz. The SoC voltage (V_{dd}) can be set between 1 V and 1.2 V depending on the FC and cluster clock frequency.

C. Automatic Deployment Tools

Developing CNNs for an MCU-class processing device, like the GAP8 on our nano-drone, is a multiobjective optimization problem. In our case, it must take into account: 1) memory limitations (L2 512 kB and L1 64 kB) and 2) the hardware limitations (i.e., no floating point unit), power envelope (∼100 mW), and throughput (i.e., a flying drone needs to react in real-time). As for PULP-Dronet v2 [5], we use an automated flow that works in two steps: first, we quantize the neural network [23] and then we perform a hardware-aware deployment of the quantized model.

For quantization, we take `float32` pretrained CNN models, and we apply post-training quantization using NEMO [24], a deep neural network (DNN) quantization tool that performs uniform asymmetric static layerwise quantization. We apply fixed-point 8-bit (`int8`) post-training quantization to the weights and activations of our CNNs. By doing so, we enable optimized 8-bit fixed-point arithmetic on GAP8, i.e., packed-SIMD instructions [16]. Moreover, the conversion from `float32` data type and the quantized `int8` reduces by 4× the memory footprint of the CNNs models.

For hardware-aware deployment, we use DORY [25], a SoA code generation tool for quantized DNNs. It is tuned for deployment on memory-constrained embedded devices with multiple levels of memory hierarchy to optimize performance. To fit the data in the available memory resources, large operations need to be split into smaller pieces called tiles. DORY models the tile size optimization problem as a constraint programming problem with the memory size as the main constraint and hardware-aware heuristics to guide the ILP solver toward the best performing solutions.

While tiling makes the operations fit into our desired memory, it still requires data marshaling between the memory levels to process the whole layer. DORY overlaps this data movement cost with computation by employing multibuffering and software pipelining to hide this data movement cost. For efficient operation computation, DORY exploits the PULP-NN [16] open-source library, which offers hand-optimized kernels for quantized neural networks executing on the PULP cluster.

D. Baseline PULP-Dronet v2 CNN

PULP-Dronet v2 [5] is an end-to-end vision-based CNN for autonomous navigation aboard nano-drones and suited for deployment on the AI-deck. We employ PULP-Dronet v2 as a baseline model for comparison with our work. The architecture of this CNN is represented in Fig. 3, and it is based on a sequence of three residual blocks [26], which we denote as ResBlock (RB). Each block consists of two parallel branches: the main branch performs two 3×3 convolutions, while the bypass one performs one 1×1 convolution. The number of output channels is 32, 64, and 128 for the three blocks, respectively. For each block, the last convolution on both branches applies a stride factor of 2 to half the feature map size. Following each convolution is a batch normalization (BN) layer and a rectified linear unit (ReLU) activation function. The ReLU output is capped to the value 6, therefore called ReLU6.

The CNN processes a grayscale image of size 200×200. This image is the bottom-center crop extracted from the QVGA image captured by the AI-deck's Himax camera. The final layer produces two distinct outputs: 1) a collision probability (classification problem) and 2) a steering angle (regression problem). Consequently, the model is trained using two distinct metrics: 1) the mean squared error (MSE) and 2) the binary cross-entropy (BCE). These metrics are combined into a unified loss function, $\text{Loss} = \text{MSE} + \beta\text{BCE}$, which we also employ in this work. β is set to 0 for the first 10 epochs, gradually increasing in a logarithmic way to prioritize the regression problem. The training employs a dynamic *negative hard-mining* approach, gradually focusing the loss computation on the k-top samples exhibiting the highest error. PULP-Dronet v2 is deployed in fixed-point 8-bit arithmetic on the multicore GAP8 SoC, yielding 41 MMAC operations per frame and 320 kB of weights.

IV. Data Set Collection Methodology

A. Data Set Collection Framework

We developed a software framework for data set collection, outlined in Fig. 2(a). It enables a pilot to manually control

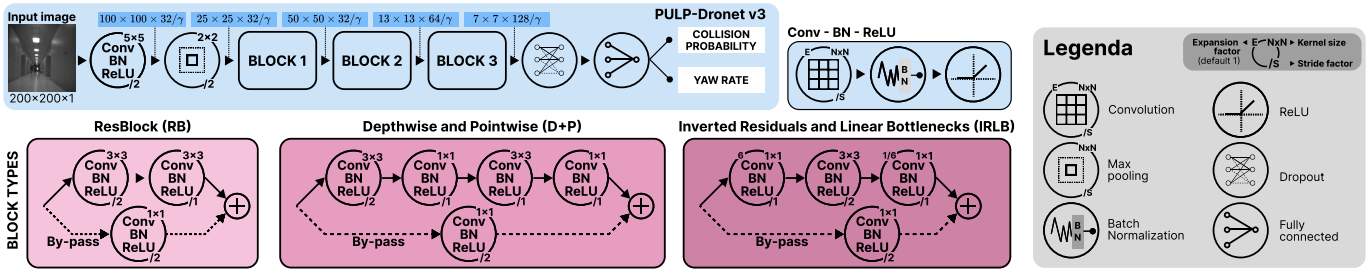


Fig. 3. Our CNN architecture exploration includes: 1) three block types—RB, D+P, and IRLB; 2) an optional bypass connection (dashed line); and 3) variations on the number of channels based on γ. Output feature map sizes are represented as (width × height × channels).

the nano-drone and simultaneously log: 1) any data from the STM32 flight controller (e.g., the drone's state estimation) and sensors attached to it (e.g., additional expansions decks of the Crazyflie 2.1,[2] such as ranging sensors) and 2) images captured by the AI-deck's front-facing camera. All data is recorded on a PC base station, which independently receives the two data streams: 1) the STM32 packets, transmitted via radio and 2) AI-deck images, transmitted via Wi-Fi.

To ensure the data from these two separated streams can be accurately matched in post-processing, we 1) periodically synchronize both the STM32 and GAP8 MCUs with a global clock, generated from a GPIO of GAP8 and 2) send each chuck of data associated with a timestamp that is globally synchronized across our robotic platform. This synchronization methodology minimizes matching errors by ensuring each packet is timestamped at its source, thus avoiding errors related to wireless communication and leaving only minimal discrepancies caused by the MCUs' oscillator drifts.

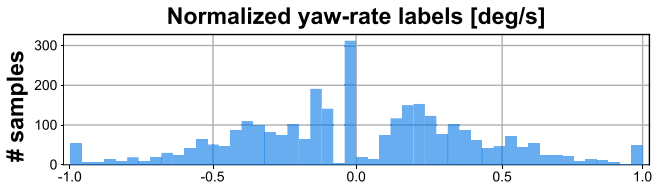
Our data set collector framework consists of: 1) code for the STM32 and GAP8 SoC [Fig. 2(a)], which enables sending the data streams while keeping the global clock synchronized and 2) a graphical user interface (GUI) [Fig. 2(b)] to plot the data streamed from the flight controller and visualize images collected from the AI-deck.

B. Our Data Set for Nano-Drone Autonomous Navigation

We introduce a new data set for visual-based autonomous nano-drone navigation. We move on from the limitations of the past PULP-Dronet data set [5], which was created by assembling different sets of data, each one having images labeled either for the steering or the collision avoidance task, as explained in Section II-A, ultimately penalizing the static obstacle avoidance [5]. To tackle this limitation, we collect a new unified data set from scratch.

With the data set collector framework introduced in Section IV-A, we collected a data set of 66 k images for nano-drones' autonomous navigation, for a total of ∼600 MB of data. We used the Bitcraze Crazyflie 2.1, described in Section III-A. A human pilot manually flew the drone, collecting 1) images from the grayscale QVGA Himax camera sensor of the AI-deck; 2) the gamepad's yaw-rate, normalized in the $[-1; +1]$ range, inputted from the human pilot; 3) the drone's estimated state; and 4) the distance between obstacles and the drone measured by the front-looking ToF sensor.

[2]https://store.bitcraze.io/collections/decks



Trivial Predictor	Collision (Acc)	Steering (RMSE)
Never Collision	**0.58**	**N/A**
Always Collision	**0.42**	**N/A**
Always straight	**N/A**	**0.41**

Fig. 4. Distribution of the yaw-rate labels (normalized in $[-1, +1]$) of our testing data set and classification/regression performances of three trivial predictors.

After the data collection, we labeled all the images with a binary collision label whenever an obstacle was in the line of sight and closer than 2 m. We recorded 301 sequences in 20 different environments. Each sequence of data is labeled with high-level characteristics: scenario (i.e., indoor or outdoor), path type (i.e., presence or absence of turns), obstacle types (e.g., pedestrians, chairs), flight height (i.e., 0.5 m, 1 m, 1.5 m), light conditions (dark, normal, bright), acquisition date, and a location name identifier.

For training and validating our CNNs, we post-processed the data sets as follows. We used 70%, 10%, and 20% of the images as training, validation, and testing sets, respectively. We augmented the training images by applying random flipping, brightness augmentation, vignetting, and blur. The resulting training data set has ∼124 k images, split as follows: 109, 5, and 10 k images for training, validation, and testing, respectively. To address the labels' bias toward the center of the $[-1; +1]$ yaw-rate range in our testing data set—specifically, the over-representation of images associated with a yaw-rate of 0, indicating no input from the human pilot and thus the drone flying straight—we balanced the data set by selectively removing a portion of images that had a yaw-rate of 0.

The final distribution of our balanced data set is represented in Fig. 4. In the same Figure, we also report the RMSE and Accuracy for three trivial predictors, i.e., a predictor that either always predicts collision to 1 or 0, and a predictor that always predicts a yaw-rate of zero (i.e., go straight). These values can be later used in Section VI as a baseline to assess the RMSE and Accuracy performance of our trained CNNs.

V. Neural Network Architecture Design

The original PULP-Dronet v2 CNN's architecture [5] exploits three RBs [26], as described in Section III-D. In other vision tasks, the RB architecture has been largely replaced by other kinds of CNN tolopogies, which provide similar accuracy characteristics but with lower workload and memory footprint [3]. Therefore, we design the new PULP-Dronet v3 architecture by exploring several modifications to its baseline topology.

Our modifications, detailed in Fig. 3, involve substituting the three RB blocks of PULP-Dronet v2, removing parallel by-pass branches, and varying the number of channels in the intermediate feature maps. To optimize autonomous navigation and enable more complex onboard functionality, we studied several modifications to this setup. First, we replace these blocks with two other blocks we designed, taking inspiration from the well-known Mobilenet family of CNNs [17], [27]. These CNNs have been shown to be both accurate for visual AI tasks and suitable for efficient MCU deployment [3]. For each block we substitute, we keep the same output feature map dimensions (width and height) of PULP-Dronet v2.

The first new block we design is based on the Mobilenet v1 [27]. It uses separable depthwise and pointwise (D+P) convolutional layers instead of traditional ones. Such layers factorize a standard convolution into a sequence of $K \times K$ *depthwise* convolution and a 1×1 convolution called *pointwise* convolution, where K is the kernel size. We define our D+P block as consisting of two branches: the main branch performs two 3×3 D+P convolutions, while the parallel by-pass branch executes a standard 1×1 convolution. We apply convolutional strides different from 1 only to the last convolution in both branches. Each convolution (depthwise, pointwise, and standard) is subsequently followed by a BN layer and a ReLU6. We refer to the sequence of these three layers as a Conv-BN-ReLU. This Conv-BN-ReLU pattern simplifies both quantization and deployment explained in Section VI-D: the tensor outputted by the Conv operation demands a finer grain representation (utilizing 32 bit) compared to the inputs and weights. Therefore, the BN is used to accumulate a 32 bit representation, while the ReLU reduces it back immediately to 8 bit.

The second block we define is inspired by Mobilenet v2 [17], using inverted residuals and linear bottlenecks (IRLBs) layers. This block comprises 1) a 1×1 convolution, expanding the number of channels by an expansion factor (E); 2) a D+P convolution; and 3) a 1×1 projection convolution that inverts the expansion, reducing the number of output channels. Stride factors different from 1 are applied in the D+P convolution. We choose an expansion ratio equal to 6 as in [17]. Additionally, we add a by-pass branch performing a 1×1 convolution, ensuring the same output size as the main IRLB branch.

We explore additional architecture variations by considering the removal of parallel by-pass branches from each block type. These branches primarily serve to mitigate vanishing gradient effects in deep CNNs [26]. However, recent studies [18] have highlighted their inefficacy in shallow CNN models, such as the seven-layer PULP-Dronet. Lastly, we vary the number of channels of the intermediate CNN feature maps to investigate the accuracy, memory footprint, and computational cost tradeoffs. As in [18] and [27], we thin the CNNs' tensors by applying a γ dividing factor to the number of channels across all convolutional layers. We span the γ parameter in the range $[1, 2, 4, 8]$, where $\gamma = 1$ corresponds to the baseline size for PULP-Dronet v2 [5].

TABLE II
Cross-Validation Between PULP-Dronet v2 Data Set and Ours (v3)

Training \ Testing	**v2** (6.9 k images)		**v3** (10 k images)	
	RMSE	**Accuracy**	**RMSE**	**Accuracy**
v2 (64 k images)	0.118	90%	0.556	54%
v3 (109 k images)	0.236	67%	0.350	83%

VI. Experimental Results

A. Data Sets Evaluation

In Table II, we assess the impact of training and testing on our new data set (v3) versus the Himax+Original PULP-Dronet v2 data set used in the SoA work [5] (v2); each data set is split into different training and testing sets. We perform this assessment by keeping the original PULP-Dronet v2 CNN architecture in both cases. When the CNN is trained on v2 and tested on v2, we achieve performance consistent with the results reported in [5]. When training on v3 and testing on v3, both performances decrease compared to the previous case due to the higher complexity of the v3 testing set, such as a more uniform distribution of steering labels and a richer, and therefore, more challenging, data set. We also report the cross-validation of training on v2 and testing on v3, and training on v3 and testing on v2. In both cases, performances drop compared to training and testing on the same data set version, but the v2-trained CNN (tested on v3) has a higher drop on both RMSE and Accuracy than the v3-trained (tested on v2), suggesting better generalization capabilities for the proposed data set.

B. CNN Architectures Exploration

To evaluate the PULP-Dronet v3 family of CNNs proposed in Section V, we start by analyzing different block types (RB, D+P, and IRLB) and the effect of removing the by-pass branches. In Table III, we assess the CNNs in terms of parameter count, MAC operations, and key performance metrics — the accuracy for the classification problem (the higher, the better) and the RMSE for the regression one (the lower, the better).

Removing the by-pass branches negligibly affects the CNN performance metrics in all cases, i.e., at most +0.005 RMSE and +1% in accuracy with IRLB CNN. On the other side, the by-pass removal brings the benefit of reducing the CNN size by 3%−11% and the number of MAC by 3%−15%,

TABLE III
ANALYSIS OF THE PULP-DRONET V3 CNN ARCHITECTURES, VARYING ITS BLOCKS AND BY-PASS BRANCHES. THE FIRST ROW MATCHES THE SOA BASELINE ARCHITECTURE [5]

Blocks	by-pass	γ	Type	RMSE	Acc	MAC	Param	Size [B]
RB	yes	/1	fp32	0.339	83%	41M	320k	1.3M
RB	no	/1	fp32	0.339	83%	40M	309k	1.2M
D+P	yes	/1	fp32	0.352	83%	14M	63k	252k
D+P	**no**	**/1**	**fp32**	**0.350**	**84%**	**12M**	**51k**	**204k**
IRLB	yes	/1	fp32	0.369	82%	43M	140k	560k
IRLB	no	/1	fp32	0.364	83%	41M	128k	513k

TABLE IV
ACCURACY, RMSE, MACS, AND MEMORY FOOTPRINT OF OUR CNN ARCHITECTURES BY VARYING γ. THE LAST ROW CORRESPONDS TO TINY-PULP-DRONET V3

Blocks	by-pass	γ	Type	RMSE	Acc	MAC	Param	Size [B]
D+P	no	/1	fp32	0.350	84%	12M	51k	204k
D+P	no	/2	fp32	0.367	84%	5.2M	17k	69k
D+P	no	/4	fp32	0.373	81%	2.4M	6.6k	26k
D+P	no	/8	fp32	0.379	78%	1.1M	2.9k	12k

depending on the architecture. Focusing on the three CNNs without by-pass, the RB variant achieved the lowest RMSE of 0.339, whereas D+P and IRLB models score 0.350 and 0.369, respectively. Instead, the D+P neural network achieves the highest performance of 84% on the accuracy score. The D+P CNN is also the faster CNN, requiring only 12 MMAC per inference, and the smaller in memory usage, being 6.2 × smaller than RB and 2.5 × smaller than IRLB. Ultimately, we select the D+P model without by-pass branches for its efficiency and minimal performance drop.

C. CNNs Size Analysis

In this section, we analyze how the number of channels across our CNNs affects their memory requirements, number of operations, and regression/classification performances. Starting from the CNN architecture D+P without by-passes, selected in Section VI-B, we span the γ parameter (see Section V) in the range $[1, 2, 4, 8]$, where $\gamma = 1$ corresponds to the baseline size for PULP-Dronet v2 [5]. Table IV shows that using a dividing factor $\gamma = 2$ does not affect the classification accuracy of the CNN when compared to $\gamma = 1$, both scoring 84%. When using smaller CNNs, the classification accuracy drops by 3% with $\gamma = 4$ and by 7% with $\gamma = 8$.

On the regression task, the RMSE gradually increases as the CNNs become smaller. For $\gamma = 1$, the RMSE stands at 0.350, but increase to 0.367, 0.373, and 0.379 for $\gamma = 2$, $\gamma = 4$, and $\gamma = 8$, respectively. On the other hand, the γ factor greatly impacts the number of the CNN's parameters. The biggest CNN ($\gamma = 1$) has 204 k parameters, the CNN with ($\gamma = 2$) has 3× less parameters (69 k parameters). Increasing the γ to 4 leads to a model with 26 k parameters, and finally the smallest CNN ($\gamma = 8$), which we call Tiny-PULP-Dronet v3, leads to only 1.9 k parameters.

TABLE V
ACCURACY, RMSE, AND MEMORY FOOTPRINT OF OUR QUANTIZED CNNS. THE LAST ROW CORRESPONDS TO TINY-PULP-DRONET V3

Blocks	by-pass	γ	Type	RMSE	Acc	MAC	Param	Size [B]
D+P	no	/1	int8	0.361	84%	12M	51k	51k
D+P	no	/2	int8	0.373	82%	5.2M	17k	17k
D+P	no	/4	int8	0.378	81%	2.4M	6.6k	6.6k
D+P	no	/8	int8	0.388	78%	1.1M	2.9k	2.9k

TABLE VI
CNNS' THROUGHPUT WHEN DEPLOYED TO GAP8 AT ITS MAXIMUM PERFORMANCE CONFIGURATION. THE ENERGY PER INFERENCE IS REPORTED FOR TWO CONFIGURATIONS (E_{ee} AND E_{mp}). THE LAST ROW CORRESPONDS TO TINY-PULP-DRONET V3

Blocks	by-pass	γ	Cycles	$\frac{MAC}{Cycle}$	frame/s	E_{ee} [mJ]	E_{mp} [mJ]
D+P	no	/1	5.1M	2.4	34	2.1	3.0
D+P	no	/2	2.9M	1.8	61	1.1	1.7
D+P	no	/4	1.7M	1.4	101	0.6	1.0
D+P	no	/8	1.3M	0.9	139	0.4	0.7

D. Quantization and Deployment

In this section, we progress with quantizing and deploying the four D+P models without by-passes, introduced in Section VI-C, to evaluate their tradeoff between accuracy/RMSE and onboard execution efficiency. We apply 8-bit post-training quantization, as detailed in Section VI-D. Table V outlines the quantized models' regression/classification, compared to nonquantized models in Table IV. Quantization introduces only 2% reduction in accuracy on the CNN employing $\gamma = 2$, while other CNNs keep the same score of the 32-bit precision. Instead, the RSME is more sensitive to the new int8 data type, with the error slightly increasing of 0.011, 0.006, 0.005, and 0.009, for $\gamma = 1, 2, 4, 8$, respectively. Compared to the original float32 models (Table IV), we reduce by 4× the memory footprint of each CNN.

Then, we deploy these four quantized CNNs on the GAP8 SoC to analyze their on-device performances. Table VI shows their inference rate (frame/s) when GAP8 is running at its max performance (mp) configuration, i.e., FC@250 MHz, CL@175 MHz, and V_{dd}@1.2 V. Our largest CNN model ($\gamma = 1$) achieves a throughput of 34 frame/s, which is 1.8× higher than the SoA PULP-Dronet v2, peaking at 19 frame/s, despite having the same number of channels across the CNN architecture; this improvement derives from our architecture modifications. Other configurations of γ result in 61 frame/s with 14 kB, 101 frame/s with 4.7 kB, and 139 frame/s with 1.9 kB for $\gamma = [2, 4, 8]$, respectively. Our smallest model ($\gamma = 8$), called Tiny-PULP-Dronet v3, improves the throughput by 7.3× and reduces the memory footprint by 168× compared to PULP-Dronet v2.

Finally, we measure the power consumption of all CNNs under SoC configurations: 1) the mp setting and 2) the energy-efficient (ee) one, which operates at FC@50 MHz, CL@100 MHz, and V_{dd}@1.0 V. In the ee configuration, the CNN with $\gamma \in 1, 2$ consume 38 mW, while the CNNs with $\gamma \in 4, 8$ show an average power consumption of 34 mW. On

Fig. 5. (a) Our U-shaped path for the in-field experiments. (b) Top-view 2-D representation of the path, highlighting its division into three segments (S1, S2, and S3) and the position of the four obstacles (represented with black rectangles).

the other hand, the *mp* setting sees all CNNs averaging a power consumption of 100 mW. As shown in Table VI, the energy for one-frame inference with the ee configuration (E_{ee}) is 2.1, 1.1, 0.6, 0.4 mJ for the $\gamma = 1, 2, 4, 8$, respectively. On the other hand, the one-frame inference energy needed for the *mp* configuration (E_{mp}) is always $\sim 1.5\times$ higher.

VII. In-Field Testing

We evaluate the navigation capabilities of our PULP-Dronet v3 and Tiny-PULP-Dronet v3 CNNs with in-field experiments. We record all experiments in a flying room equipped with a Qualisys motion capture system (24 cameras) with mm-precise tracking of our drone. We track the position and pose of our drone @100 Hz to analyze the drone's flight in post-processing. We investigate if the new data set we collected, having unified labels for *yaw-rate* and *collision probability*, improves the navigation capabilities of the SoA PULP-Dronet v2, which was trained with disjoint *steering* and *collision probability* labels, and therefore struggles in avoiding static obstacles, as described in [5].

We devised a challenging navigation scenario involving a U-shaped corridor, illustrated in Fig. 5, and divided it into three segments. Segments S1 and S3 are straight paths, each one presenting two obstacles, whereas segment S2 features a 180-degree turn. The 2 m wide corridor has obstacles 1 m wide that are leaning on opposite walls, blocking straight pathways. We designed two experiments: 1) one where all obstacles are static and 2) one where the second one is dynamic. In the experiment with a dynamic obstacle, obstacle 2 appears in the center of the lane as soon as the drone passes the first obstacle, providing 1.5 m of braking space, and it is removed 5 s later.

These scenarios aim at testing the nano-UAV's capabilities on multiple skills: 1) avoiding both static and dynamic obstacles; 2) navigation through a narrow environment (a corridor); and 3) a sharp 180° turn. The corridor environment we use in our tests is *never-seen-before* for both PULP-Dronet v2 and PULP-Dronet v3, not part of either CNN's training set.

We must stress that we designed these tests to challenge PULP-Dronet v2, focusing on its weaknesses specifically. While the CNN has demonstrated excellent navigation capabilities in corridor turns and dynamic obstacle avoidance, the original tests revealed that it struggles with static obstacles in narrow tunnels [5].

We evaluate our closed-loop system across three specific target speeds: $v_{\text{target}} = 0.5, 1.0, 1.5$ m/s. We set a target flight altitude of 0.5 m in every test. We conduct five experiments for each combination of CNN and v_{target} for statistical relevance (total 90 tests), and we compare them with the SoA PULP-Dronet v2 [5]. For the sake of comparability, we always use the same drone's control state machine of PULP-Dronet v2 [5], including a first-order low-pass filter on both the forward speed of the drone v_{unfilt} in (1) and yaw rate ω_{unfilt} in (2). The filtered values for the forward velocity (v_{filter}) and the yaw rate (ω_{filter}) are fed to the drone's flight controller. We set the low-pass filtering parameters $\alpha_1 = \alpha_2 = 0.3$. We use the stock hardware from Bitcraze, including 1) stock motors (max current of 1 A); 2) a 250 mA h battery; and 3) stock propellers with a 45 mm diameter, for a total weight of 35 g.

The videos of the experiments are accessible through the link provided in the supplementary material section

$$v_{\text{filter}}(k) \leftarrow \alpha_1 \cdot v_{\text{unfilt}} + (1-\alpha_1) \cdot v_{\text{filter}}(k-1) \quad (1)$$

$$\omega_{\text{filter}}(k) \leftarrow \alpha_2 \cdot \omega_{\text{unfilt}} + (1-\alpha_2) \cdot \omega_{\text{filter}}(k-1). \quad (2)$$

A. U-Shaped Path With Static Obstacles

We conduct the first set of experiments with all static obstacles. Table VII outlines the success rate for each CNN tested along with the average speed (v_{avg}) of the drone. If the drone fails to complete the entire path, v_{avg} is noted as N/A. If it fails to complete a segment, we do not start the drone again on subsequent segments, and their success rates are noted as "—" when they are never attempted. PULP-Dronet v2 never succeeds at any speed point to pass segment S1 of the path. The two obstacles of segment S1 visually obstruct the entire corridor's width, resulting in a consistently high-collision probability, while the CNN's steering output keeps the drone in the center of the lane defined by the surrounding walls. As a result, the drone either does not move forward due to the CNN's high-collision probability, or it slowly drifts against obstacle 1, ultimately crashing. This outcome aligns with a similar scenario described in [5], where PULP-Dronet v2 encountered a 100% failure rate in tackling a narrow tunnel scenario with static obstacles.

Moving to our CNNs, we plot the trajectories of PULP-Dronet v3 in Fig. 6(a)–(c), and the trajectories of Tiny-PULP-Dronet v3 in Fig. 6(d)–(f). First, we analyze the results

TABLE VII
SUCCESS RATE OF OUR CLOSED-LOOP SYSTEM IN THE U-SHAPED PATH WITH STATIC OBSTACLES ONLY. WE REPORT THE SUCCESS RATE FOR THE S1-S2-S3 SEGMENTS, AND THE v_{avg} OVER THE COMPLETE PATH FOR PULP-DRONET V2, PULP-DRONET V3, AND TINY-PULP-DRONET V3

$\mathbf{v_{target}}$ [m/s]	PULP-Dronet v2 [5]				PULP-Dronet v3 (ours)				Tiny-PULP-Dronet v3 (ours)			
	success rate				success rate				success rate			
	(S1)	(S2)	(S3)	v_{avg} [m/s]	(S1)	(S2)	(S3)	v_{avg} [m/s]	(S1)	(S2)	(S3)	v_{avg} [m/s]
0.5	0/5	—	—	N/A	5/5	5/5	4/5	0.39	5/5	5/5	5/5	0.45
1	0/5	—	—	N/A	5/5	4/5	3/5	0.24	5/5	0/5	—	N/A
1.5	0/5	—	—	N/A	5/5	0/5	—	N/A	5/5	0/5	—	N/A

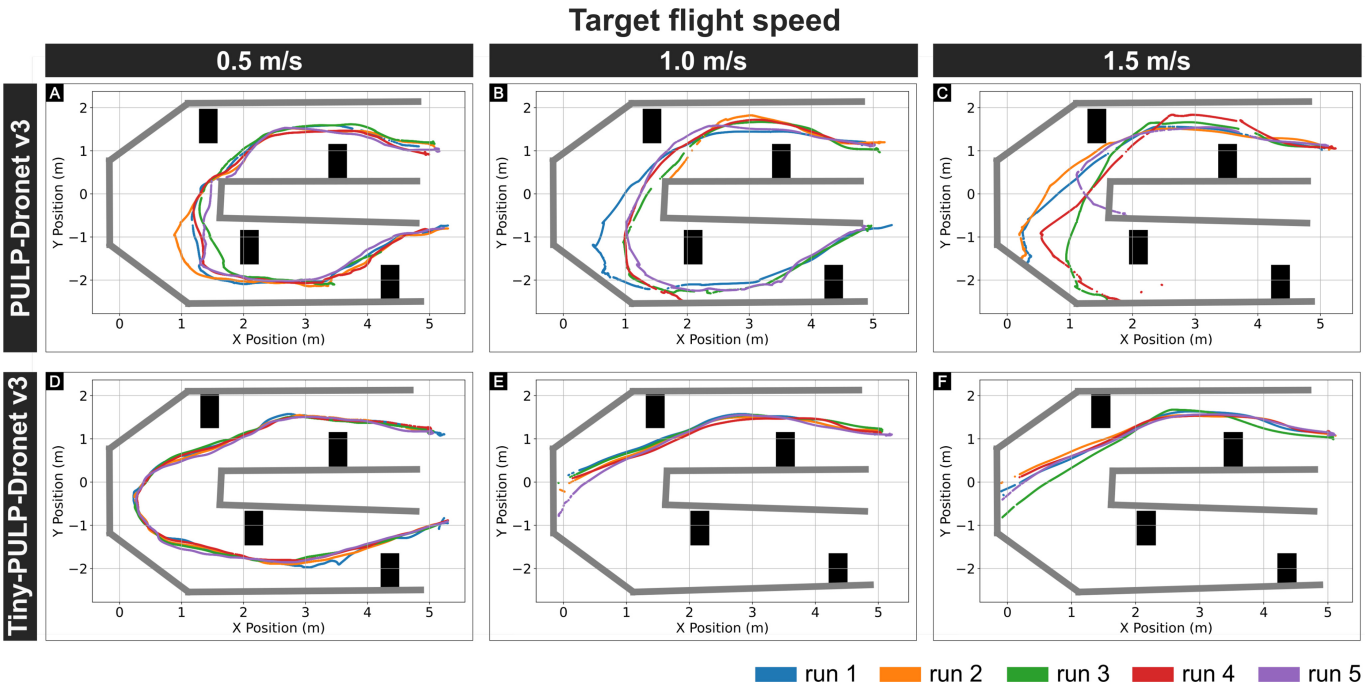


Fig. 6. In-field experiments trajectories of PULP-Dronet v3 (A-B-C) and Tiny-PULP-Dronet v3 (D-E-F) for three target speeds $v_{\text{target}} = [0.5, 1.0, 1.5]$ m/s, tested with static obstacles only (represented as black rectangles).

with a target speed of 0.5 m/s. PULP-Dronet v3 and Tiny-PULP-Dronet v3 fly through the whole U-shaped path with a single failure and no failures, respectively. The only time that PULP-Dronet v3 fails, it crashes against obstacle 4, still successfully completing 66% of the corridor.

Analyzing the @1 m/s target speed configuration, PULP-Dronet v3 successfully navigates the entire corridor three times while crashing one time in segment S2 and once in segment S3. On the other hand, Tiny-PULP-Dronet v3 reliably succeeds 5/5 times only in segment S1 of the path, always crashing during the turn of segment S2. This outcome is attributed to Tiny-PULP-Dronet v3's lower RMSE performance compared to PULP-Dronet v3, as detailed in Section VI-D. Analyzing the @1.5 m/s target speed configuration, none of the CNNs tested succeeded in completing the turn, outlining the speed upper limit of our closed-loop systems in this scenario, but still succeeding 100% of the times in avoiding the obstacles in segment S1.

In conclusion, the static obstacle avoidance success rate of PULP-Dronet v3 and Tiny-PULP-Dronet v3 is inversely proportional to the target flight speed. While PULP-Dronet v3 and Tiny-PULP-Dronet v3 show 80% and 100% success rates at 0.5 m/s, their success rates over the whole path lowers at higher speeds. PULP-Dronet v3 reduces its success rate to 60% at 1 m s, while Tiny-PULP-Dronet v3 and PULP-Dronet v3 never succeed at 1 m s and 1.5 m s, respectively. These results mark an improvement with respect to PULP-Dronet v2, showing that 1) the data set we collected with joint labels successfully trains CNNs that can tackle both obstacle avoidance and steering tasks together and 2) our Tiny-PULP-Dronet v3, with only 2.9 k parameters, can enable static obstacle avoidance while being 168× smaller than the SoA model.

B. U-Shaped Path With Static and Dynamic Obstacles

We conduct the second set of experiments stressing dynamic obstacle avoidance. We use the same setup as Section VII-A, but now obstacle 2 appears in the center of the corridor after the drone passes obstacle 1. Table VIII outlines the success rate of the three CNNs tested. Starting from the SoA CNN, PULP-Dronet v2 never succeeds in navigating any of the three segments (S1, S2, S3) of our path for the same reason described in Section VII-A, getting stuck in front of obstacle 1 and eventually crashing into it.

TABLE VIII
SUCCESS RATE OF OUR CLOSED-LOOP SYSTEM IN THE U-SHAPED PATH WITH THREE STATIC OBSTACLES AND A DYNAMIC ONE. WE REPORT THE SUCCESS RATE FOR THE S1-S2-S3 SEGMENTS, AND THE v_{avg} OVER THE COMPLETE PATH FOR PULP-DRONET V2, PULP-DRONET V3, AND TINY-PULP-DRONET V3

v_{target} [m/s]	PULP-Dronet v2 [5]				PULP-Dronet v3 (ours)				Tiny-PULP-Dronet v3 (ours)			
	success rate				success rate				success rate			
	(S1)	(S2)	(S3)	v_{avg} [m/s]	(S1)	(S2)	(S3)	v_{avg} [m/s]	(S1)	(S2)	(S3)	v_{avg} [m/s]
0.5	0/5	—	—	N/A	5/5	5/5	3/5	0.57	5/5	5/5	3/5	0.44
1	0/5	—	—	N/A	5/5	5/5	5/5	0.98	5/5	5/5	3/5	0.82
1.5	0/5	—	—	N/A	4/5	3/5	2/5	1.33	2/5	0/5	—	N/A

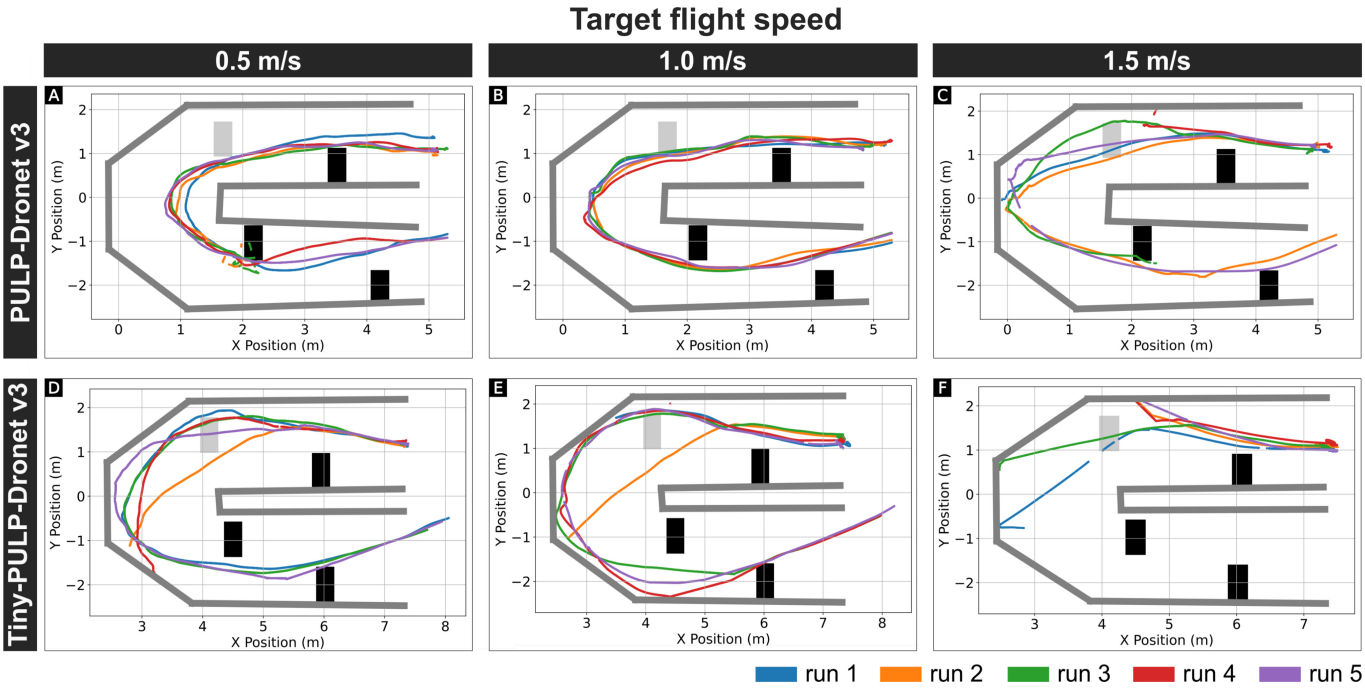


Fig. 7. In-field experiments trajectories of PULP-Dronet v3 (A-B-C) and Tiny-PULP-Dronet v3 (D-E-F) for three target speeds $v_{target} = 0.5$, 1.0, and 1.5 m/s, tested with three static obstacles (represented as black rectangles) and a dynamic one (represented as a light gray rectangle).

Moving to our CNNs, we plot the trajectories of PULP-Dronet v3 in Fig. 7(a)–(c) and the trajectories of Tiny-PULP-Dronet v3 in Fig. 7(d)–(f). PULP-Dronet v3 shows the highest success rate among all the CNNs tested. It completes the whole path with a success rate of 60%, 100%, and 40% when flying at 0.5, 1.0, and 1.5m/s, respectively. Remarkably, PULP-Dronet v3 only crashed once against the dynamic obstacle while flying at the highest speed (1.5 m/s), always succeeding in passing the S1 segment in all other cases. On the other hand, the Tiny-PULP-Dronet v3 completes the whole corridor with a 60% success rate when flying at both 0.5 m/s and 1 m/s. However, it always fails at a higher speed of 1.5 m/s, either crashing on the right wall of segment S1 or failing to complete the turn in S2. Nevertheless, Tiny-PULP-Dronet v3 avoided the collision against the dynamic obstacle of segment S1 two times when flying at the highest speed.

In conclusion, our experiments demonstrate that for PULP-Dronet v3 and Tiny-PULP-Dronet v3, the dynamic obstacle avoidance rate is inversely proportional to the drone's flight speed when it exceeds 1 m/s. Both CNNs reliably avoid collisions with the dynamic obstacle in Section S1 100% of the time at speeds up to 1 m/s. However, at the highest speed of 1.5 m/s, the success rates decrease to 80% for PULP-Dronet v3 and 40% for Tiny-PULP-Dronet v3, respectively.

VIII. CONCLUSION

Nano-sized UAVs are ideal candidates as ubiquitous flying IoT nodes. In this article, we distill a novel family of CNNs for autonomous navigation on nano-drones, i.e., the Tiny-PULP-Dronet v3 CNNs. Compared to the SoA, our models reduce memory footprint by up to 168× (down to 2.9 kB) and achieve an inference rate of up to 139 frame/s. We create a new open-source 66 k image data set for autonomous nano-UAV navigation. We compare with in-field tests both SoA and our CNNs on a COTS nano-UAV. Our tiny CNN succeeds in navigating a challenging path with static and dynamic obstacles and a 180° turn at speeds of up to 1 m/s. In contrast, the baseline consistently fails despite having 168× more parameters.

For future development, we aim to exploit the compute and memory budget we have freed with Tiny-PULP-Dronet v3 by introducing additional tasks, e.g., executing multiple CNNs onboard [18]. Additionally, efficiency can be further improved

by implementing a more aggressive quantization scheme [23], e.g., 4-bit or 2-bit, coupled with quantization-aware training, to speed up computation and reduce the memory footprint while maintaining accuracy.

ACKNOWLEDGMENT

The authors thank the Autonomous Robotics Research Center of the Technology Innovation Institute, Abu Dhabi, UAE, for granting them access to their indoor flying arena. They thank Michał Barciś and Daniel Ribien for their support in data set collection and Elia Cereda for his support and assistance.

SUPPLEMENTARY MATERIAL

Supplementary video at: http://youtu.be/ehNlDyhsVSc.

Open-source code and data set: http://github.com/pulp-platform/pulp-dronet.

REFERENCES

[1] J. E. Kooi and R. Babuška, "Inclined quadrotor landing using deep reinforcement learning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, 2021, pp. 2361–2368.

[2] G. Shi, W. Hönig, X. Shi, Y. Yue, and S.-J. Chung, "Neural-Swarm2: Planning and control of heterogeneous multirotor swarms using learned interactions," *IEEE Trans. Robot.*, vol. 38, no. 2, pp. 1063–1079, Apr. 2022.

[3] E. Cereda et al., "Deep neural network architecture search for accurate visual pose estimation aboard Nano-UAVs," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2023, pp. 6065–6071.

[4] L. Lamberti, L. Bompani, V. J. Kartsch, M. Rusci, D. Palossi, and L. Benini, "Bio-inspired autonomous exploration policies with CNN-based object detection on nano-drones," in *Proc. Design, Autom. Test Europe Conf. Exhibit. (DATE)*, 2023, pp. 1–6.

[5] V. Niculescu, L. Lamberti, F. Conti, L. Benini, and D. Palossi, "Improving autonomous Nano-drones performance via automated end-to-end optimization and deployment of DNNs," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 11, no. 4, pp. 548–562, Dec. 2021.

[6] N. S. Labib, M. R. Brust, G. Danoy, and P. Bouvry, "The rise of drones in Internet of Things: A survey on the evolution, prospects and challenges of unmanned aerial vehicles," *IEEE Access*, vol. 9, pp. 115466–115487, 2021.

[7] Z. Wei et al., "UAV-assisted data collection for Internet of Things: A survey," *IEEE Internet Things J.*, vol. 9, no. 17, pp. 15460–15483, Sep. 2022.

[8] J. Higgins, N. Mohammad, and N. Bezzo, "A model predictive path integral method for fast, proactive, and uncertainty-aware UAV planning in cluttered environments," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, 2023, pp. 830–837.

[9] E. Cereda et al., "Improving the Generalization capability of DNNs for ultra-low power autonomous Nano-UAVs," in *Proc. 17th Int. Conf. Distrib. Comput. Sensor Syst. (DCOSS)*, 2021, pp. 327–334.

[10] M. J. Anderson, J. G. Sullivan, J. L. Talley, K. M. Brink, S. B. Fuller, and T. L. Daniel, "The 'Smellicopter,' a bio-hybrid Odor Localizing Nano air vehicle," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, 2019, pp. 6077–6082.

[11] N. H. Motlagh, T. Taleb, and O. Arouk, "Low-altitude unmanned aerial vehicles-based Internet of Things services: Comprehensive survey and future perspectives," *IEEE Internet Things J.*, vol. 3, no. 6, pp. 899–922, Dec. 2016.

[12] H. Shakhatreh et al., "Unmanned aerial vehicles (UAVs): A survey on civil applications and key research challenges," *IEEE Access*, vol. 7, pp. 48572–48634, 2019.

[13] N. Zhang, F. Nex, G. Vosselman, and N. Kerle, "End-to-end Nano-drone obstacle avoidance for indoor exploration," *Drones*, vol. 8, no. 2, p. 33, 2024.

[14] D. Hanover et al., "Autonomous drone racing: A survey," *IEEE Trans. Robot.*, vol. 40, pp. 3044–3067, 2024.

[15] B. Habas, J. W. Langelaan, and B. Cheng, "inverted landing in a small aerial robot via deep reinforcement learning for triggering and control of rotational maneuvers," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2023, pp. 3368–3375.

[16] A. Garofalo, M. Rusci, F. Conti, D. Rossi, and L. Benini, "PULP-NN: Accelerating quantized neural networks on parallel ultra-low-power RISC-V processors," *Philosoph. Trans. Roy. Soc. A*, vol. 378, no. 2164, 2020, Art. no. 20190155.

[17] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," 2019, *arXiv:1801.04381*.

[18] L. Lamberti et al., "Tiny-PULP-Dronets: Squeezing neural networks for faster and lighter inference on multi-tasking autonomous nano-drones," in *Proc. IEEE 4th Int. Conf. Artif. Intell. Circuits Syst. (AICAS)*, 2022, pp. 287–290.

[19] J. Dupeyroux, R. Dinaux, N. Wessendorp, and G. De Croon, "A novel obstacle detection and avoidance dataset for drones," in *Proc. DroneSE RAPIDO Syst. Eng. Constrained Embedded Syst.*, 2022, pp. 8–13.

[20] A. Loquercio, A. I. Maqueda, C. R. Del-Blanco, and D. Scaramuzza, "DroNet: Learning to fly by driving," *IEEE Robot. Autom. Lett.*, vol. 3, no. 2, pp. 1088–1095, Apr. 2018.

[21] Y. Chang, Y. Cheng, J. Murray, S. Huang, and G. Shi, "The HDIN dataset: A real-world indoor UAV Dataset with multi-task labels for visual-based navigation," *Drones*, vol. 6, no. 8, p. 202, 2022.

[22] M. Navardi, E. Humes, and T. Mohsenin, "E2EdgeAI: Energy-efficient edge computing for deployment of vision-based DNNs on autonomous tiny drones," in *Proc. IEEE/ACM 7th Symp. Edge Comput. (SEC)*, 2022, pp. 504–509.

[23] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, "Pruning and quantization for deep neural network acceleration: A survey," *Neurocomputing*, vol. 461, pp. 370–403, Oct. 2021.

[24] F. Conti, "Technical report: NEMO DNN Quantization for deployment model," 2020, *arXiv:2004.05930*.

[25] A. Burrello, A. Garofalo, N. Bruschi, G. Tagliavini, D. Rossi, and F. Conti, "DORY: Automatic end-to-end deployment of real-world DNNs on low-cost IoT MCUs," *IEEE Trans. Comput.*, vol. 70, no. 8, pp. 1253–1268, Aug. 2021.

[26] S. S. Saini and P. Rawat, "Deep residual network for image recognition," in *Proc. IEEE Int. Conf. Distrib. Comput. Elect. Circuits Electron. (ICDCECE)*, 2022, pp. 1–4.

[27] A. G. Howard et al., "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.

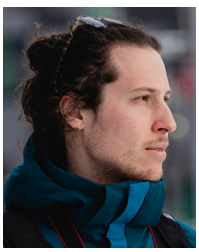
Lorenzo Lamberti (Member, IEEE) received the Ph.D. degree in electronic engineering from the University of Bologna, Bologna, Italy, in 2024,

He is currently a Researcher with the University of Bologna. His research encompasses artificial intelligence, miniaturized robotics, ultralow-power embedded systems, and neuromorphic vision. His work has resulted in over 12 publications in peer-reviewed international conferences and journals.

Dr. Lamberti was the Technical Lead of the winning team in the "Nanocopter AI Challenge," hosted at the 2022 IMAV International Conference in TU Delft.

Lorenzo Bellone received the B.Sc. degree from the Faculty of Industrial Engineering from the Politecnico di Torino, Turin, Italy, in 2018, and the M.Sc. degree from the Faculty of Telecommunication Engineering, Politecnico di Torino in 2021.

He is currently a Researcher with the Autonomous Robotics Research Center, Technology Innovation Institute, Abu Dhabi, UAE. His research interests focus on deep learning for communication networks, UAV-aided networks, and deployment of ML solutions for aerial multiagent systems.



Luka Macan received the B.Sc. and M.Sc. degrees from the Faculty of Electrical Engineering and Computing, University of Zagreb, Zagreb, Croatia, in 2017 and 2019, respectively. He is currently pursuing the Ph.D. degree with the University of Bologna, Bologna, Italy.

His research interests include machine learning on embedded systems and hardware accelerators.

Enrico Natalizio (Senior Member, IEEE) received the master's degree *(magna cum laude)* and the Ph.D. degree in computer engineering from the University of Calabria, Arcavacata, Italy, in 2000 and 2005, respectively.

He is currently the Chief Researcher of the Autonomous Robotics Research Center, Technology Innovation Institute, Abu Dhabi, UAE, and a Full Professor with the LORIA Laboratory, Université de Lorraine, Metz, France. In October 2010, he was a Postdoctoral Researcher with POPS Team, Inria Lille—Nord Europe, Villeneuve-d'Ascq, France, and from 2012 to 2018, he was an Associate Professor with the Université de technologie de Compiègne, Compiègne, France, and a Full Professor with the Université de Lorraine from September 2018. His research interests include UAV communications and networking, robot and sensor communications with applications in networking technologies for disaster management and infrastructure monitoring, and IoT privacy and security.

Dr. Natalizio has been ranked in the Top 2% Worldwide Scientists in Stanford University's Bibliometric Study of 2021. He is currently an Associate Editor of *Vehicular Communications* (Elsevier) and *Computer Networks*.

Francesco Conti (Member, IEEE) received the Ph.D. degree in electronic engineering from the University of Bologna, Bologna, Italy, in 2016.

He is currently a Tenure-Track Assistant Professor with the DEI Department, University of Bologna. From 2016 to 2020, he held a Research Grant position with the University of Bologna and a position as a Postdoctoral Researcher with ETH Zürich, Zürich, Switzerland. His research is centered on hardware acceleration in ultralow power and highly energy-efficient platforms, with a particular focus on System-on-Chips for artificial intelligence applications. His research work has resulted in more than 90 publications in international conferences and journals.

Dr. Conti's research work was awarded several times, including the 2020 IEEE Transactions on Circuits and Systems I: Regular Papers Darlington Best Paper Award.

Daniele Palossi received the Ph.D. degree in information technology and electrical engineering from ETH Zürich, Zürich, Switzerland.

He is currently a Senior Researcher with Dalle Molle Institute for Artificial Intelligence, USI-SUPSI, Lugano, Switzerland, where he leads the Nano-Robotics Research Group, and with the Integrated Systems Laboratory, ETH Zürich. His research stands at the intersection of artificial intelligence, ultralow-power embedded systems, and miniaturized robotics. His work has resulted in more than 45 peer-reviewed publications in international conferences and journals.

Dr. Palossi was a recipient of the Swiss National Science Foundation Spark Grant, the Second Prize at the Design Contest held at the ACM/IEEE ISLPED'19, several Best Paper Awards, and Team Leader of the Winning Team of the First "Nanocopter AI Challenge" hosted at the IMAV'22 International Conference.

Luca Benini (Fellow, IEEE) received the Ph.D. degree from Stanford University, Stanford, CA, USA, in 1997.

He holds the Chair of Digital Circuits and Systems with ETH Zürich, Zürich, Switzerland, and is a Full Professor with the Università di Bologna, Bologna, Italy. His research interests are in energy-efficient parallel computing systems and machine learning hardware.

Dr. Benini is the recipient of the 2016 IEEE CAS Mac Van Valkenburg Award, the 2020 EDAA Achievement Award, the 2020 ACM/IEEE A. Richard Newton Award, and the 2023 IEEE CS E.J. McCluskey Award. He is a Fellow of the ACM and a member of the Academia Europaea.

Open Access funding provided by 'Alma Mater Studiorum - Università di Bologna' within the CRUI CARE Agreement