Improving Autonomous Nano-Drones Performance via Automated End-to-End Optimization and Deployment of DNNs

Vlad Niculescu, Lorenzo Lamberti, *Graduate Student Member, IEEE*, Francesco Conti, *Member, IEEE*, Luca Benini, *Fellow, IEEE*, and Daniele Palossi

***Abstract*— The evolution of energy-efficient ultra-low-power (ULP) parallel processors and the diffusion of convolutional neural networks (CNNs) are fueling the advent of autonomous driving nano-sized unmanned aerial vehicles (UAVs). These sub-10 cm robotic platforms are envisioned as next-generation ubiquitous smart-sensors and unobtrusive robotic-helpers. However, the limited computational/memory resources available aboard nano-UAVs introduce the challenge of minimizing and optimizing vision-based CNNs – which to date require error-prone, labor-intensive iterative development flows. This work explores methodologies and software tools to streamline and automate all the deployment of vision-based CNN navigation on a ULP multicore system-on-chip acting as a mission computer on a Crazyflie 2.1 nano-UAV. We focus on the deployment of PULP-Dronet (Palossi *et al.*, 2019), a state-of-the-art CNN for autonomous navigation of nano-UAVs, from the initial training to the final closed-loop evaluation. Compared to the original hand-crafted CNN, our results show a 2× reduction of memory footprint and a speedup of 1.6× in inference time while guaranteeing the same prediction accuracy and significantly improving the behavior in the field, achieving: *i*) obstacle avoidance with a peak braking-speed of 1.65 m/s and improving the speed/braking-space ratio of the baseline, *ii*) free flight in a familiar environment up to 1.96 m/s (0.5 m/s for the baseline), and *iii*) lane following on a path featuring a 90 deg turn – all while using for computation less than 1.6% of the drone's power budget. To foster new applications and future research, we open-source all the software design in a ready-to-run project compatible with the Crazyflie 2.1.**

***Index Terms*— Unmanned aerial vehicle (UAV), convolutional neural network (CNN), autonomous navigation, nano-drone, ultra-low-power (ULP).**

Manuscript received May 15, 2021; revised September 10, 2021; accepted October 20, 2021. Date of publication November 8, 2021; date of current version December 13, 2021. This work was supported in part by the Autonomous Robotics Research (ARR) Center of the United Arab Emirates (UAE) Technology Innovation Institute (TII). This article was recommended by Guest Editor I. Partin-Vaisband. *(Corresponding author: Vlad Niculescu.)*

Vlad Niculescu is with the Integrated Systems Laboratory, ETH Zürich, 8092 Zürich, Switzerland (e-mail: vladn@iis.ee.ethz.ch).

Lorenzo Lamberti 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; f.conti@unibo.it).

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

Daniele Palossi is with the Integrated Systems Laboratory, ETH Zürich, 8092 Zürich, Switzerland, and also with Dalle Molle Institute for Artificial Intelligence, USI-SUPSI, 6962 Lugano, Switzerland (e-mail: daniele.palossi@idsia.ch).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/JETCAS.2021.3126259.

Digital Object Identifier 10.1109/JETCAS.2021.3126259

I. Introduction

In THE past years, unmanned aerial vehicles (UAVs) have been adopted in a wide range of applications, such as surveillance and inspection of hazardous areas [1], [2]. Nano-size UAVs, with a form factor of a few centimeters and a weight of tens of grams, are the ideal candidates for fully autonomous indoor navigation as they can safely operate near humans and reach narrow spots with their reduced dimensions [1], [3], [4]. However, these platforms have a total power envelope of a few Watts, of which only $5-15\%$ is allotted for computation, making it challenging to deploy real-time navigation pipelines directly onboard [5]. Furthermore, the small physical footprint and limited payload that can be carried by nano-UAVs constrains the battery and printed circuit board sizes. Overall, these constraints mean that onboard computing devices need to have the physical footprint, power envelope, and on-chip memory of a typical microcontroller unit (MCU).

For traditional UAVs, the classical approach for autonomous navigation is simultaneous-localization-and-mapping (SLAM), which creates a map of the environment and plans the trajectory according to it [6]. Classical SLAM is too computationally intensive to be feasible on nano-UAVs. An alternative emerging approach is to infer relevant navigation information directly from onboard sensors and cameras, using machine learning-based algorithms. In particular, deep convolutional neural networks (CNNs) have recently proved to provide good performance in autonomous navigation, at a fraction of the cost of SLAM: enough to run practical navigation tasks directly on highly resource-constrained platforms [3], [7]. Still, achieving more sophisticated navigation skills requires to deploy more complex CNNs under even stricter real-time constraints, to promptly react to challenging dynamic environments, avoid collisions, plan new routes, etc. Therefore, it is imperative to look for strategies to minimize the models' complexity and footprint while maintaining high accuracy.

Recently, low-power multi-core System-on-Chips (SoCs) have been introduced as potentially ideal devices to combine an MCU's flexibility with AI-oriented compute acceleration capabilities [8], [9]. At their peak performance, these devices deliver up to 10–100× better performance and efficiency than conventional MCUs, constituting an ideal platform for fully onboard DNN-driven autonomous navigation. However, their complex architecture, together with the non-trivial

2156-3357 © 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: ETH BIBLIOTHEK ZURICH. Downloaded on December 16,2021 at 14:16:18 UTC from IEEE Xplore. Restrictions apply.

requirements of DNN-based algorithms, requires a complex procedure including training, quantization, and a difficult hand-tuning phase to maximize performance on the final target – a critical step to achieve high frame rate and thus good in-field navigation performance.

In this work, we focus on automating the end-to-end deployment of a DNN-based neural flight controller on top of a nano-UAV employing the GreenWaves Technologies (GWT) GAP8 SoC [8] – one of the most advanced commercially available AI-oriented SoCs suitable to nano-size drones. We evaluate two distinct toolsets available for GAP8, namely the GAP*flow* provided by GWT and the open-source NEMO/DORY flow fostered by the research community [10]. Specifically, we adapt and tune these flows to automatically deploy a CNN for autonomous navigation based on the state-of-the-art (SoA) PULP-Dronet [1]. PULP-Dronet is a residual network used to drive a nano-UAV through an interior (e.g., a corridor) or exterior (e.g., street) environment, deriving a *probability of collision*, used for obstacle avoidance, and a *steering angle* to keep within a lane – implemented as a classification and a regression task, respectively.

Differently from the seminal PULP-Dronet, which relied on 16-bit fixed-point data representation, we focus on fully automated deployment, including network quantization to 8-bits, data tiling, code generation for the GAP8 SoC, evaluation of performance on the regression and classification tasks. We also improve the integration of the new PULP-Dronet with the flight controller, with a more robust approach to deal with situations where the network's output is not providing strong guidance. We compare our results in terms of accuracy to the original PULP-Dronet showing that the prediction capability is maintained ($\sim$90% for the classification despite the stronger quantization). Our results show a throughput up to 19frame/s, improved by a factor of up to 1.6$\times$ and total energy consumption of $\sim 3{-}4$mJ/frame, which is $\sim 44-58\%$ less than our baseline.

Moreover, we contribute a thorough exploration of the real-world performance of the CNN in exterior and interior environments, evaluating the drone's adherence to expected behavior in several controlled experiments performed in a room equipped with a Vicon motion capture system. We individually assess the obstacle avoidance and steering capabilities. We find that the drone can stop 0.42 m away from a dynamic obstacle that appears 1.5 m in front of the drone while flying with 1.41m/s, with a significant 25.3% improvement in the speed/braking-distance ratio vs. the original PULP-Dronet. Furthermore, we also demonstrate the capability of the drone to fly an angled narrow tunnel, and we record the trajectories for various drone velocities. We also evaluate the free-flight capabilities of the drone in a controlled indoor environment, achieving 110 m path in 56 s, which marks an improvement of $\sim 4\times$ vs. our baseline.

Our new, streamlined approach significantly improves the autonomous flight capabilities of PULP-Dronet while freeing up resources (i.e., reduced memory footprint and inference time), which allows the system to handle even more tasks (e.g., localization, detection, tracking, etc.). Also, we investigate the generalization capabilities of the drone flying in new environments that are not captured by the training dataset. Among all considered environments, we recorded the longest flight time of 171 s in an urban street.

TABLE I
UAVs TAXONOMY BY VEHICLE CLASS-SIZE [11]

Vehicle class	⊘ : Weight [cm:kg]	Power [W]	Onboard Device
standard-size [12]	$\sim 50 : \geq 1$	≥ 100	Desktop
micro-size [13]	$\sim 25 : \sim 0.5$	~ 50	Embedded
nano-size [1]	$\sim 10 : \sim 0.01$	~ 5	MCU
pico-size [5]	$\sim 2 : \leq 0.001$	~ 0.1	ULP

II. RELATED WORK

A. Standard/Micro-Sized UAVs

Customarily, we divide UAVs into four categories, shown in Table I, according to size, weight, total power consumption, and onboard processing platform. The latter two characteristics are directly linked, as the budget for onboard electronics is limited to $\sim$5-15% of the total [5]. The overwhelming majority of complex robotic perception algorithms have been demonstrated aboard standard- and micro-sized UAVs [12]–[14], which feature powerful onboard computers, often equipped with GPUs, such as NVIDIA Jetson TX1/TX2. The sophisticated functionality that can be achieved with these platforms include onboard autonomous navigation in an unstructured natural environments [12]; and search for particular objects in the field of view using semantic segmentation, for example for smart agriculture [13]. To work, these functionalities need multiple CNNs and other non-neural algorithms, such as visual SLAM, to enable multiple concurrent tasks, such as pose estimation, collision avoidance, and trajectory planning.

B. MCU-Based Nano-Sized UAVs

Highly miniaturized nano-size UAVs [1], [4], [7] have a diameter of about 10 cm, weigh only a few tens of grams, and have a total power budget of a few Watts – not enough to directly support the functionality discussed above, with only simple MCU-class devices and a few MB of memory onboard. We can distinguish three categories of solutions to enable autonomous navigation despite these limitations: restricting to limited functionality to minimize the workload [7], [15]–[19]; offloading computation to an external base-station [20]–[22]; or extending the onboard computing device either with general-purpose visual navigation engines [3], [4] or with application-specific processors [23]–[27]. From the first category, Lambert *et al.* [15] exploit the STM32F4 MCU to implement a simple DL-based *flight-controller* for hovering on a Crazyflie 2.0. Similarly, Guanya Shi *et al.* [16] apply a simple DNN-based controller ($\sim$ 27kMAC) on a swarm of nano-drones, which enables multiple drones to fly safely in close proximity. Zhao *et al.* [7] implement a CNN to improve the localization accuracy of a nano-UAV by modeling the sensor biases of the localization system. This last DL model can run at 200 Hz on the onboard ARM Cortex M4 MCU requiring about 2.7kMAC/frame, i.e., 10000$\times$ less operations than

Authorized licensed use limited to: ETH BIBLIOTHEK ZURICH. Downloaded on December 16,2021 at 14:16:18 UTC from IEEE Xplore. Restrictions apply.

existing SoA CNN-based autonomous navigation workloads, e.g., ∼40MMAC/frame in [1]. In [18], the authors propose a lightweight navigation algorithm that enables a swarm of drones to explore an indoor area while avoiding collision with the walls by exploiting four laser distance sensors on each drone. The autonomous navigation is commanded by a finite state machine, which maps the sensor output into drone control commands. The simple sensor input (i.e., four single laser beams) results in an avoidance mechanism that can only detect large and homogeneous obstacles. In contrast, our solution is based on the visual cues of the environment and, therefore, more general and robust against various types of obstacles. Reference [19] proposes an approach for navigation with obstacle avoidance using a random forest classifier. They report a classification accuracy of 90% while using a size of 229 nodes for the decision tree. However, their approach was only developed and tested with synthetic data generated with the aid of a simulator. While useful for some specific tasks, these tiny models are not a viable solution for more challenging navigation problems, like the ones we tackle in this work.

An approach to overcome the computational limitations of single-core MCUs is to offload intensive computation to off-board, wireless-connected computing resources. For example, in [22] the authors propose an autonomous navigation approach based on a CNN that uses reinforcement learning to adjust part of its parameters online. The initial values of the weights are obtained by training the whole network with synthetic images obtained from a game engine, and they also prove the in-field functionality using a 80 g drone. However, the action space of their algorithm is limited to: move 50 m forward, steer 45° and steer −45°. This results in less smooth and flexible navigation than our approach, which provides a continuous output for the steering angle and adaptive forward velocity. In [20] the authors implement a fuzzy logic position controller and vision-based position estimation by offloading all the computation to an Intel i7 processor streaming images with a 2.4 GHz radio. This class of approaches, however, suffers from several important drawbacks [28]: *i*) it introduces network-dependent latency, which prevents the drone to operate farther than few tens of meters from the remote base-station, *ii*) the noise on the transmission channel affects the reliability of the transmitted data, *iii*) security becomes a concern for eavesdropping of confidential images and data and for denial-of-service attacks on the wireless connections, and *iv*) the power consumption of the high-frequency radio transmission is significant and the wireless transceiver may dominate the power budget for control.

C. Accelerated Nano-Size UAVs

A possible solution to the limitations imposed by single-core MCUs is augmenting nano-UAVs with better compute functionality. For larger UAVs, this is a common choice – using devices such as NVIDIA GPUs, Intel Myriad, or Google Edge TPU [12], [13], [29], [30] that are both flexible and highly efficient. For nano-UAVs, however, the possibilities are more limited. Some recent works emphasize the efficacy of application-specific integrated circuits (ASIC), which suitable for autonomous navigation functionalities [23]–[27] on an low-power budget. Some of these systems have been designed to tackle specific UAV applications, such as visual-inertial odometry (VIO) [23] and simultaneous localization-and-mapping (SLAM) [24], [25], within a power envelope of few hundred mW. While extremely efficient, these systems are inflexible and they do not implement end-to-end flying functionality, but only accelerate some sub-functions, requiring anyways a mission and flight controller MCU.

The approach we follow in this work targets end-to-end mission control using a fully programmable parallel computing accelerator [31]. Parallel ultra-low-power (PULP) processors use small-scale multi-core clusters with 4-16 cores, with an enhanced RISC-V instruction set architecture (ISA), to exploit intrinsic parallelism of vision workloads, including CNNs. An example of this new generation of energy-efficient devices is the 9-core GWT GAP8 SoC, which has been already applied to nano-drones [1], [3], [4] for tasks such as obstacle avoidance, lane detection, and pose estimation using CNNs in the range of 10-100MMAC/frame. The key advantage of this approach is its flexibility and the capability to handle the end-to-end flight control task.

D. Automatic Deployment Tools

Deploying multi-MMAC CNNs on an MCU-class device requires coping with a power envelope of less than 1 W, a memory of just a few MB or less, and limited peak performance, demanding for a strict co-optimization of the algorithmic, software, and hardware components [10], [32]. Minimization of a DL model can be performed either with specific topological choices, like using depth-wise convolutions [33], [34] or using quantization as a compression technique [35], [36] from *float32* down to *int8* or less, with a net 4× reduction of model footprint. Quantization can also expose more data parallelism exploitable by packed-SIMD instructions [37], improving the final inference throughput and the energy consumption.

Moreover, given a size-optimized network, the deployment challenge must be addressed, which consists in achieving the maximum utilization of computing resources by *i*) parallelizing computation, *ii*) managing the memory hierarchy (i.e., topology-dependent tiling), and *iii*) minimizing data transfers overheads. This is a key step especially for MCU devices, where the processing units are scarce by definition [10].

General-purpose tools such as TFLite for MCUs and Larq [38]–[40], as well as vendor-locked tools like STM32 X-CUBE-AI[1] have been proposed to ease deployment on MCUs. For PULP platforms, on which we focus in this work, two deployment tools have been recently introduced: GWT's *AutoTiler*,[2] which is partially closed-source, and DORY [10] with PULP-NN backend [41], an alternative open-source academic framework.

[1]https://www.st.com/en/embedded-software/x-cube-ai.html

[2]https://greenwaves-technologies.com/manuals/BUILD/AUTOTILER/html/index.html

Authorized licensed use limited to: ETH BIBLIOTHEK ZURICH. Downloaded on December 16,2021 at 14:16:18 UTC from IEEE Xplore. Restrictions apply.

In this work, we exploit these recent advancements to bring the deployment of DL-based visual navigation on nano-drones from handcrafted and hand-tuned deployment [1] to a new streamlined, automated methodology. We leverage DNN deployment frameworks by integrating them in our flow and we achieve significantly improved performance and energy efficiency on autonomous navigation DL workloads, improving the nano-UAV in-field behavior and freeing resources for even more complex missions and tasks.

III. Background

A. Robotic Platform and Hardware

The algorithmic kernel of our application use case, i.e., PULP-Dronet V2, runs on a commercial embodiment of the PULP platform [37], the GWT GAP8 SoC [8]. GAP8 is a 1+8 general-purpose RISC-V-based multicore MCU, where the nine cores are organized in two power and frequency domains, namely the fabric controller (FC) and the cluster (CL), as shown in Figure 1. The former features one single core for control-oriented tasks, acting as an "activity supervisor" managing the interfaces to off-chip sensors/memories and orchestrating on-chip memory operations. On the other hand, the CL is designed to execute computationally intensive parallel workloads, such as vision-based CNNs, so to enable high-level energy efficiency via the parallel computational paradigm [31].

All nine cores are based on the open-source RI5CY core [37], which implements the `RV32IMC` instruction set and the `Xpulpv2` extension with digital signal processing (DSP) instructions: register-register multiply-accumulate, hardware loops, load/store operations with post-increment, packed single-instruction-multiple-data (SIMD), and specialized instructions for dot-product – the latter two operating on vectors of 8-bit or 16-bit data. The on-chip memory hierarchy is organized with 512 kB of L2 SRAM and 16 kB of L1 on the FC, while the CL has a 64 kB shared L1 as a tightly-coupled data memory (TCDM). The TCDM is organized in sixteen banks, providing an aggregate bandwidth of 11.2GB/s@175 MHz. It is connected to the cores by means of a fully combinational logarithmic interconnect, which guarantees 0-wait state access from the cores in the absence of collisions – in which case, one of the colliding accesses is stalled for one cycle. To enable data transfer, the GAP8 SoC features two DMA engines: the first one, called μDMA, is in charge of data exchange with external I/O peripherals (e.g., DRAM, cameras, etc.) across a wide range of interfaces (e.g., QSPI, HyperBus, etc.). Inside the CL domain, the second DMA controller is connected on one side to the TCDM logarithmic interconnected; on the other side to the AXI interconnect. The CL DMA can be used to transfer data between the L2 and the L1 TCDM memory at up to 8B/cycle in either direction, guaranteeing data availability for the eight CL cores.

The robotic platform we employ in this work is the COTS open-source Crazyflie 2.1 nano-quadrotor from Bitcraze.[3]

[3]https://www.bitcraze.io/products/crazyflie-2-1

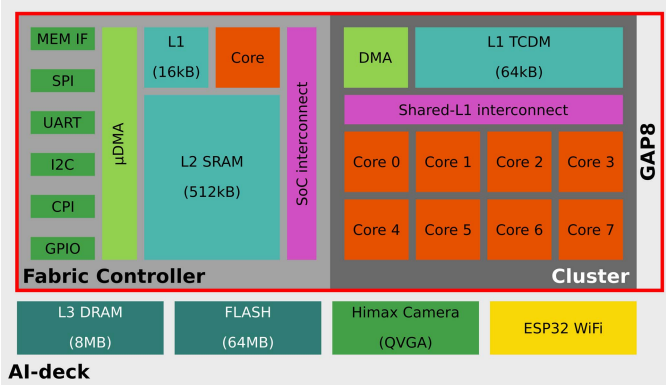


Fig. 1. AI-deck diagram and the GAP8 System-on-Chip architecture.

This tiny UAV weighs 27 g, has a diameter of ∼10 cm, and a total payload of ∼ 15 g. The underlying robotic platform is built around the STM32F405 MCU, in charge of all low-level flight controller functionalities, such as sensors' interfacing, state estimation, and low-level control.

In this work, we employ a configuration that extends the basic nano-drone with two commercially available open-source, pluggable printed circuit boards (PCBs): the Flow-deck and the AI-deck.[4] The former features a low-resolution down-looking optical-flow camera coupled with a time-of-flight (ToF) sensor, enabling the drone to detect motions in any direction and providing a distance measurement from the ground, respectively. The latter is the commercially supported version of the PULP-Shield research prototype, introduced in [3]. This board, shown in Figure 1, extends the nano-drone's onboard capabilities with an energy-efficient GAP8 processor, off-chip DRAM, and Flash memory (8 MB and 64 MB, respectively), a QVGA resolution low-power gray-scale camera (i.e., Himax HM01B0 sensor), and a versatile ESP32-based WiFi module.[5]

Combining the STM32 MCU with the GAP8 SoC enables the heterogeneous architectural paradigm at the ultra-low-power scale [31], enabling onboard execution of sophisticated vision-based algorithms. In this *host-accelerator* context, the STM32 represents the *host*, handling control-oriented tasks (i.e., flight controller), while the GAP8 offers general-purpose parallel computation capabilities, acting as the *accelerator* for compute-intensive perception and navigation tasks.

B. PULP-Dronet CNN

Dronet [42] is a vision-based end-to-end autonomous drone navigation CNN, deployed on a nano-drone, for the first time, in the seminal PULP-Dronet project [1]. This shallow NN is based on three consecutive ResNet [43] blocks that branch the last layer to produce two outputs: a probability of collision (classification problem) and a steering angle (regression problem). The CNN was originally developed using 16-bit fixed-point arithmetic as the result of a quantization-aware training

[4]https://store.bitcraze.io/products/ai-deck

[5]We remind the reader that, in this work, the Wi-Fi radio has been used only for debugging and showcasing video-streaming purposes.

Authorized licensed use limited to: ETH BIBLIOTHEK ZURICH. Downloaded on December 16,2021 at 14:16:18 UTC from IEEE Xplore. Restrictions apply.

process. Images used in the original training/validation/testing, and also in this work, are partitioned in three disjoint sets:

- **Udacity**: ∼39.1K high-resolution images labeled only with steering angle.
- **Bicycle**: ∼32.2K high-resolution images labeled only with collision probabilities.
- **Himax**: ∼1.3K low-resolution images collected from the same camera aboard our target nano-drone and labeled only with collision probabilities.

The union of Udacity and Bicycle sets results in the so-called *Original* dataset that we use to train our PULP-DroNet V2 in PyTorch (100 epochs) and to select the models that minimize both regression and classification error on the validation set.

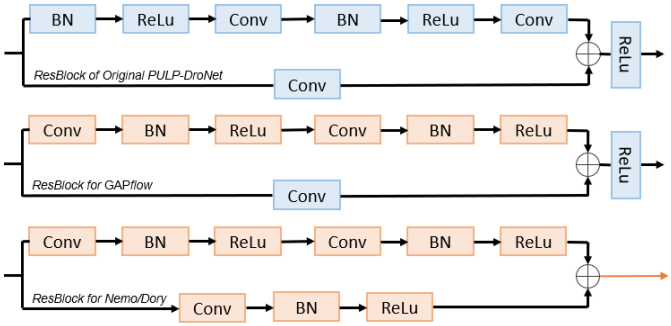


Fig. 2. ResBlocks of PULP-DronetV1, PULP-DronetV2 GAP*flow* and PULP-DronetV2 NEMO/DORY (top-bottom order).

IV. Deployment Automation Flow

The development of AI-based algorithms on MCU-class processors, aboard a nano-drone, is a complex multi-objective optimization problem that must take into account: *i*) memory availability, *ii*) power envelope, *iii*) hardware limitations (e.g., no FPU), and *iv*) throughput. Therefore, to enable the execution of PULP-Dronet on GAP8 under these constraints, we assemble and streamline a flow of automated tools that divide the process into two main stages: *i*) quantization of the neural network, and *ii*) hardware-aware deployment of the quantized model.

A. Quantization

This stage, remapping the CNN's numerical representation, e.g., from `float32` to `int8`, enables efficient integer computation on the underlying hardware. From a mathematical viewpoint, the tools we consider in this work focus on *uniform affine quantization*: all tensors $\mathbf{t}$ (typically inputs $\mathbf{x}$, outputs $\mathbf{y}$ or weights $\mathbf{w}$) are first restricted to a known range $[\alpha_t, \beta_t)$, then they are mapped to N-bit purely integer tensors $\widehat{\mathbf{t}}$ by means of a bijection:

$$\mathbf{t} = \alpha_{\mathbf{t}} + \varepsilon_{\mathbf{t}} \cdot \widehat{\mathbf{t}}\,, \tag{1}$$

where $\varepsilon_{\mathbf{t}} = (\beta_{\mathbf{t}} - \alpha_{\mathbf{t}})/(2^N - 1)$. $\varepsilon_{\mathbf{t}}$ is often called the *scaling factor* used to scale tensors from their floating-point to their integer representation. Quantization flows enforce the representation quantized tensors of all waits and part of the data tensors in the network – the latter typically together with ReLU activation functions.

NNTOOL is the NN mapping flow developed by GWT, included in the GAP*flow*, that converts a TFLite topology graph into a new custom representation. It is distributed as part of the GAP8 software development kit.[6] NNTOOL performs "layer-fusion", post-training calibration and quantization (8/16-bit), and folds batch normalization (BN) into the convolution layer that precedes it, avoiding costly intermediate buffers and saving a small amount of memory traffic (i.e., 1.792 kB in Dronet). On the other hand, NEMO is the quantization tool used by the open-source pipeline NEMO/DORY [10], which provides both post-training quantization (i.e., quantizing the model without further re-training, using only lightweight calibration) and quantization-aware training (i.e., quantization at training-time, to mitigate potential accuracy loss). NEMO does not fold BN layers, but instead, it converts them into fully integer channel-wise scaling operations [32].

For our application, we apply post-training quantization at 8-bit for both NEMO and NNTOOL, which is – to date – the most commonly adopted quantized bit-width and is supported by both flows. Specifically, NNTOOL employs a signed *int8* format for both activations and weights, whereas NEMO employs *uint8* for activations and *int8* for weights. Both tool-sets require a *Conv-BN-ReLU* pattern for all main branches of each ResBlock. This simplifies both quantization and deployment: the accumulated tensor at the output of the Conv operation naturally requires a finer grain representation than that of inputs and weights – both flows employ 32 bits. Integer scaling and ReLU can be applied to a single element at a time, meaning that there is no need to materialize a full tensor of 32 bits elements – rather, each element is produced at 32 bits by Conv but immediately reduced to 8 bits by ReLU or BN+ReLU. The baseline version of the NEMO flow does not support the quantization of data that is not at the output of a ReLU; as a consequence, we introduce a further modification by pushing the final ReLU of the ResBlocks back to the residual branch. Figure 2 summarizes the minor modifications that were used in the two flows with respect to PULP-DronetV1 – establishing two new NN topologies, namely, PULP-DronetV2 GAP*flow* and PULP-DronetV2 NEMO/DORY.

B. Hardware-Aware Deployment

The deployment goal is to enable and exploit the hardware platform by generating C code that: *i*) maximizes the parallel execution over all available cores, and *ii*) minimizes the data transfers overhead. On GAP8, the main challenge is the limited L1 memory (64 kB), that forces the deployment tools to solve an optimization problem, partitioning the tensors into smaller chunks of data, called tiles, to be moved between L2 and the L1 memory.

Both GAP*flow* and NEMO/DORY partition this problem in two separate parts: *i)* a set of optimized kernels operating exclusively on L1 data tiles, and *ii)* a tiling solver to define the optimal size for tiles and generate the code for the

[6]https://github.com/GreenWaves-Technologies/gap_sdk

Authorized licensed use limited to: ETH BIBLIOTHEK ZURICH. Downloaded on December 16,2021 at 14:16:18 UTC from IEEE Xplore. Restrictions apply.

related data transfers between L2 and L1, including double buffering for all tensors. As optimized primitives, GAP*flow* relies on a set of open-source kernels available within the GAP SDK, with the possibility of defining custom ones. NEMO/DORY uses the open-source PULP-NN library[7] [41]. The tiling solver employed by the GAP*flow* is a proprietary tool called AutoTiler, whereas NEMO/DORY employs DORY, an open-source flow [10].

The two primitive libraries exploit different data layouts, affecting the final performance on the Conv layers. PULP-NN, employed by NEMO/DORY, exploits the height-width-channel (HWC) layout, where the data along the channels' dimension is stored with a stride of one, while the data along the width dimension is stored with a stride equal to the number of channels. NNTOOL uses the channel-height-width (CHW) format, reverting the previous order. The convolutional layer can be performed either as a *direct convolution* or as a *matrix-matrix multiplication*, optimized for CHW and HWC layouts, respectively. Both implementations have pros and cons. Direct convolution uses a sliding window with a masking/shuffling mechanism, while in the matrix multiplication case, we need to pay some extra overhead to rearrange the input data to a single-dimension tensor (i.e., `im2col`, image-to-column) so that the convolution can be computed as matrix multiplication. On the other hand, matrix multiplication is a more regular operation than convolution, it is essentially identical for any filter size, and it does not require any data shuffling. In general, the HWC layout and the matrix-matrix multiplication become more convenient when the feature map of the convolved layer has many input channels. Conversely, the CHW data layout used by the GAP*flow* is most advantageous with direct convolutions on Conv layers with spatial dimensions much larger than the number of input channels.

The tiling solver employed by GAP*flow* is the AutoTiler, whereas the open-source flow employs DORY [10]. AutoTiler is a proprietary partially closed-source tool. It can automatically promote full tensors from L3 to L2 and to L1 or tile them in order to maximize performance using the GAP*flow* backend primitives. On the other hand, DORY specifies tiling as two separate problems – one for L3/L2, the other for L2/L1 transfers. To promote data from L3 to L2, DORY uses a set of simple heuristics, such as looking at the known-good solution first (e.g., copying the full weights for the next layer in L2 while the current one is being run; keep all activations in L2) and revert to less optimal ones when the former ones are not feasible (e.g., move part of the activations in L3). For the L2/L1 transfers, insisting on a much smaller L1 size (64 KB), tiling is specified as a constrained optimization problem with the objective to maximize L1 utilization, and at the same time maximize a few hardware-aware heuristics (e.g., favor tiles that are better parallelized due to their specific sizes).

Overall, we observe that for our CNN the AutoTiler finds a better solution for layers that are spatially large and without many input channels, such as the first convolutional layer; DORY, on the other hand, performs better for layers where the number of input channels is high. By inspection of generated code, we also notice that the AutoTiler is able to fuse consecutive layers (e.g., convolutions, max-pooling, and ReLU) and apply multiple operations directly on the same L1 tile, avoiding an intermediate copy to L2. This is the case for the first part of PULP-Dronet (i.e., Conv5×5 + MaxPool), where the AutoTiler merges the first two layers, while DORY executes them one after the other, as it can not store in the L1 memory all the needed parameters required by the HWC layout. Both GAP*flow* and NEMO/DORY implement, whenever is possible, pipelined memory/computation phases by means of the GAP8's μDMA (L3-L2).

[7]https://github.com/pulp-platform/pulp-nn

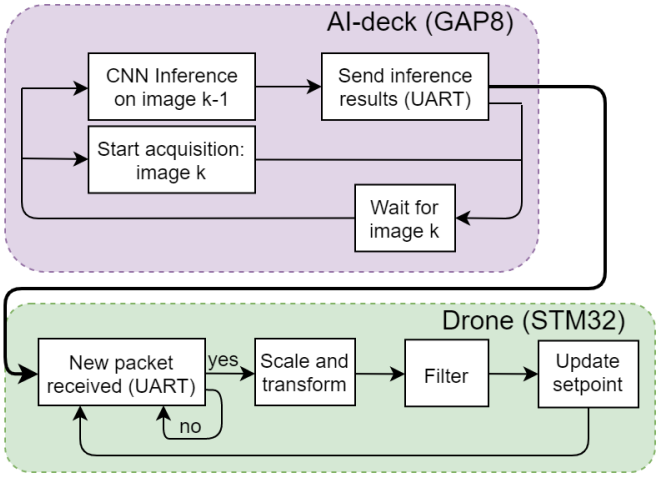


Fig. 3. Overview of the main acquisition and control loops. The AI-deck is in charge of image acquisition and perception, and the drone's MCU runs the application that interprets the perception results and transforms them into flight commands.

C. Platform Integration & Low-Level Control

To enable autonomous navigation on the nano-drone, the inference results of the CNN running in the AI-deck have to be communicated to the flight controller, running on the Crazyflie's main board. This controller is in charge of running control algorithms that drive the drone and run on top of the STM32F405 MCU. Communication between the Crazyflie flight controller and the AI-deck happens via UART communication at 115200 baud. Figure 3 shows the stages of the perception and control. In the AI-deck side (purple), whenever a new inference is started for the current image (k) in the CL, the FC also commands the acquisition of the next image (k+1), which will serve as input for the next inference. Every time a new inference result is available, the AI-deck sends this data via UART. When using GAP*flow*, the outputs of the deployed CNN are also quantized at 8 bits: the inference result, therefore, simply consists of 2 bytes - one for the probability of collision and one for the steering rate. When using NEMO/DORY, the accumulated values after the final layer – represented as 32 bit integers – are directly used as outputs. In this case, the inference result is sent as a packet of 8 bytes.

The program that allows receiving the data from the AI-deck and processing it to drive the drone is integrated as a new task in the drone's firmware. To achieve a computational-efficient

Authorized licensed use limited to: ETH BIBLIOTHEK ZURICH. Downloaded on December 16,2021 at 14:16:18 UTC from IEEE Xplore. Restrictions apply.

```
while 1:
  if uart_data_available:
      # reset the flag
      uart_data_available ← 0
      # scale data
      p_col ← data[0] * c_scale0
      ω_steer ← data[1] * c_scale1
      # compute the integral
      I(k) ← I(k+1) + (p_col − 0.3)
      I(k) ← clip_[0,3)(I(k))
      # transform: compute forward velocity
      p_col ← p_col + w · I(k)
      v_unfilt ← v_target(k) · (1 − p_col)^2
      # filter forward velocity
      v_set(k) ← α1 · v_unfilt + (1 − α1) · v_set(k − 1)
      # filter steering rate
      ω_set(k) ← α2 · ω_steer + (1 − α2) · ω_set(k − 1)
      # command the drone
      command(v_set(k), ω_set(k))
```

Listing 1. The listing describes the data processing that is applied to the raw output of the CNN to obtain the setpoint that is communicated to the drone's commander.

data exchange, the drone's MCU uses a DMA mechanism to receive the UART data from the AI-deck. The DMA is configured to trigger an interrupt whenever a certain number of bytes has been received – which is two in our case. When the interrupt is triggered, a binary flag (i.e., 0 or 1) that indicates new data available from UART is set. The main loop of the application evaluates the value of this flag every 5 ms, and in case it is set, it reads the two received bytes and then resets the flag back to 0.

The main overview of the inference post-processing stages is given in Figure 3, and each step (scale, transform, filter) is detailed by Listing 1. First, the two pieces of data ($data[0]$ and $data[1]$) associated with the output of the CNN are dequantized by multiplying them by the scaling constants resulting from the quantization process. The scaling constants are programmed in the drone's MCU firmware. Next, $I(k)$ is computed, which is an integral term that is added to the probability of collision (p_{col}), and it is meant to penalize the lasting effect of the obstacles in the field view. We noticed that the CNN is sometimes unsure about particular frontal obstacles, and the probability of collision oscillates from values > 0.8 to values below 0.3. Thanks to the addition of the integral term, when the CNN is indicating an obstacle with a $p_{col} > 0.3$, I will increase over time, building up the drone's confidence that it is facing an actual obstacle. When the CNN's inference indicates an obstacle-free horizon ($p_{col} < 0.3$) for a longer time again, $p_{col} - 0.3$ is negative and therefore the integral decreases. We clip the value of $I(k)$ to the interval $[0, 3)$ as negative confidence is meaningless (on the lower side), and excessive confidence could result in a windup effect. We scale $I(k)$ by a scaling constant w that establishes how much impact the integral has on the final value of the probability of collision. In our experiments, we set this value to 0.2.

To convert the probability of collision p_{col} into forward velocity (v_{unfilt}), we use a simple square low – penalizing velocity quadratically with respect to p_{col}. Furthermore, to reduce the high-frequency noise associated to v_{unfilt}, this value is filtered using a first-order, low-pass infinite impulse response (IIR) filter defined by the coefficient α_1 (we use $\alpha_1 = 0.6$. The same type of filter (defined by α_2) is also used for the steering rate ω_{steer}. We use $\alpha_2 = 0.7$; we observed experimentally that a lower value results in an increased delay and in low-frequency oscillations around the navigation path. The filtered values for the forward velocity and the steering rate (v_{set} and ω_{set}) represent the new flight setpoint, which is transmitted to the drone's flight controller.

TABLE II
REGRESSION & CLASSIFICATION–IN **BOLD** OUR BEST FIXED8 SCORES

Training		Testing			
NN topology	Dataset	Precision	*Original Dataset*		*Himax Dataset*
			RMSE	Acc	Acc
V1	Original	Float32	0.105	0.945	0.845
		Fixed16	0.097	0.935	0.873
	Original +Himax	Float32	0.109	0.964	0.900
		Fixed16	0.110	0.977	0.891
V2 GAP*flow*	Original	Float32	0.126	0.915	0.831
		Fixed8	**0.124**	0.916	0.840
	Original +Himax	Float32	0.136	0.925	0.881
		Fixed8	0.135	**0.925**	**0.886**
V2 NEMO/ DORY	Original	Float32	0.146	0.902	0.841
		Fixed8	0.143	**0.903**	0.836
	Original +Himax	Float32	0.118	0.893	0.905
		Fixed8	**0.120**	0.892	**0.900**

V. RESULTS

In this section, we present three main classes of results: *i*) regression and classification capability of the proposed PULP-Dronet V2 CNNs; *ii*) onboard power analysis and inference performance; *iii*) in-field closed-loop control accuracy and the real-time performance.

A. Regression & Classification Performance

In Table II, the NNs quality metrics are reported as accuracy for the classification problem and root-mean-squared error (RMSE) for the regression one, using the Original and Himax datasets for both training and testing. We also evaluate the impact of quantization w.r.t. floating-point calculation for each model proposed and each training set. Particular attention should be given to the scores achieved on the Himax testing set as it maps the type of images available on our flying drone.

1) Testing on Original Dataset: When training on the Original dataset with a `float32` format, both the proposed models show a lower performance w.r.t. the PULP-Dronet V1. The drop is between 0.02 and 0.04 in RMSE and up to $\sim 4\%$ of accuracy. This small difference can be ascribed to *i*) the differences in the NNs topologies and quantization factors (8-bits vs. 16-bits), and *ii*) a weak CNN's convergence. We attribute this weak convergence to the disjoint training datasets for the two problems (i.e., classification and regression); the relatively large unified CNN front-end, resulting in shared weights for two very different tasks up to the very last layer, may also contribute. Our models reveal a different behavior when training on the Original+Himax dataset (`float32`) than the

Authorized licensed use limited to: ETH BIBLIOTHEK ZURICH. Downloaded on December 16,2021 at 14:16:18 UTC from IEEE Xplore. Restrictions apply.

equivalent models trained on the Original set. The GAP*flow* model (similarly to the V1 baseline) slightly improves the classification performance ($\sim +1\%$ accuracy) at the price of a small reduction in the regression capability ($\sim +0.01$ RMSE); instead, the NEMO/DORY model shows the opposite behavior, i.e., accuracy $\sim -1\%$ and RMSE ~ -0.02. The small differences are mainly because the two pipelines use slightly different topologies (Figure 2), due to the different approach to batch normalization in the quantized regime.

Our four 8-bit quantized models show a minimal variation in both RMSE and accuracy metrics (within 0.003 and 0.1%, respectively) compared to the respective `float32` version, which is not the case for the V1 baseline as it improves the RMSE of 0.008 and drops 1% of accuracy. The lower variance is the consequence of the different quantization schemes adopted. In fact, in contrast to the quantization-aware training used in V1, we do not need to retrain the NNs to change the numerical domain due to our post-training quantization. Moreover, post-training quantization does not require the model's training dataset, enabling a faster and straightforward process for producing the quantized model.

2) Testing on Himax Dataset: When training on the Original dataset with a `float32` format, all the three NN topologies score a similar accuracy of $\sim 83-84\%$. Instead, training on the Original+Himax dataset, the accuracy of all models increases up to $\sim 88-90\%$, proving the beneficial effect of the dataset extension. Finally, the 8-bit quantization of the V2 models does not affect the accuracy, keeping the same maximum (90%) achieved by the 16-bit quantized baseline (V1), which again shows higher variance. Ultimately, the proposed PULP-Dronet V2 models achieve the same accuracy of the V1 baseline on the in-field-collected Himax dataset, despite the reduced data-type (8-bit vs. 16-bit), leading to a highly desirable $2\times$ reduction in the memory footprint. Including the Himax dataset in the training process does not aim at mitigating any quantization effect, but to ensure a better tuning between the CNN model and the onboard camera. In conclusion, regardless of the testing dataset, the 8-bit quantization preserves the CNN's accuracy unaltered w.r.t to the `float32` representation and allows us to deploy and run our model on the target platform successfully.[8]

B. Onboard Performance

1) Power Consumption & Inference Performance: We evaluate the execution time and power traces of the proposed models running them on the GAP8 and using a RocketLogger data logger [44] (64ksps). For these experiments, the SoC's operating points are FC@50 MHz, CL@100 MHz, VDD@1 V, as the most energy-efficient configuration [1], and FC@250 MHz, CL@175 MHz, VDD@1.2 V to push the system at its maximum performance. The GAP*flow* model processes one frame in 1.05Mcycle, while the NEMO/DORY model needs 11% fewer cycles. Since our models use almost the same topology of PULP-Dronet V1, they all compute $\sim$ 41MMAC/frame, as the original baseline. We mention, however, that each MAC in PULP-Dronet V2 is an 8×8-bit MAC rather than a 16×16-bit MAC as in V1.

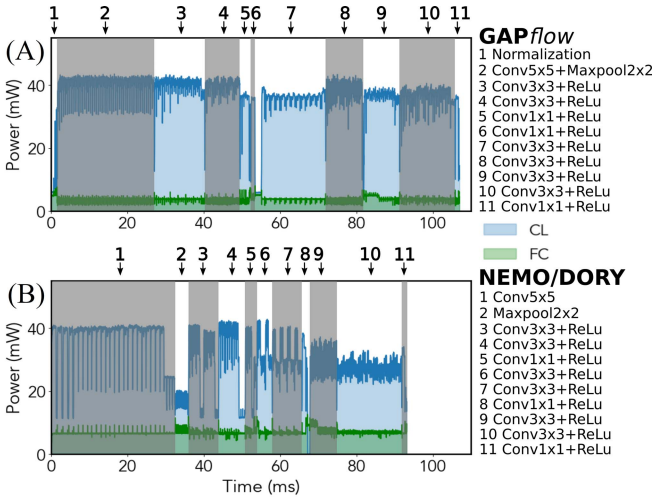


Fig. 4. GAP8's power waveforms for: FC@50 MHz, CL@100 MHz, VDD@1 V, i.e., the most energy efficient configuration.

Figure 4 shows the power consumption for one frame inference for both models (most energy-efficient configuration) and highlights the time intervals associated with the execution of each CNN's layer. The GAP*flow* model (Figure 4-A) shows an initial extra stage to normalize the 8-bit input data-range to [−127,+128], as well as some cluster idleness at the begin of layer 7, due to a μDMA transfer wait. A major difference between the two models is visible in the first two layers (i.e., conv5 $\times$ 5 and max-pool), as the GAP*flow* model achieves better performance merging them (see Section IV). Nevertheless, NEMO/DORY outperforms its counterpart during the Conv+ReLU pattern, present in each ResNet block.

In the most energy-efficient configuration, the GAP*flow* and NEMO/DORY models achieve similar performance for one frame inference, as $\sim$ 9frame/s @ $\sim$ 40 mW and $\sim$ 10frame/s @ $\sim$ 35 mW, respectively, improving the throughput vs. PULP-Dronet V1 ($40\% - 60\%$). Running the same test, with the SoC's maximum frequencies, the GAP*flow* model scores $\sim$ 17frame/s @ $\sim$ 119 mW, while the NEMO/DORY one peaks at $\sim$ 19frame/s @ $\sim$ 102 mW. Even if the GAP*flow* model exposes a more balanced utilization of the available cores, i.e., almost constant CL power consumption, it pays the overhead for the CWH layout applied to small WH. Conversely, DORY, even with a less balanced parallel workload, i.e., scattered profile inside layers 3, 4, 5, 6, and 7, reduces overheads due to its HWC layout.

Ultimately, quantization is a key-enabler technique to fully deploy a DNN model on resource-constrained COTS MCUs, which usually lack floating-point units (e.g., the GAP8 SoC). For the same reason, it is hard to precisely compare the execution performances of a full-precision model vs. a quantized one on such a processor. An option is represented by soft-float emulation of all floating-point operations; although, this approach would introduce a major execution overhead.

[8]The model's memory footprint could be further reduced with stronger quantization, e.g., 4/2/1-bit; however, this approach is not guaranteed to be sufficient to maintain the full precision regression/classification performance as shown by our work and by the SoA when adopting 8-bit quantization [35], [36].

Authorized licensed use limited to: ETH BIBLIOTHEK ZURICH. Downloaded on December 16,2021 at 14:16:18 UTC from IEEE Xplore. Restrictions apply.

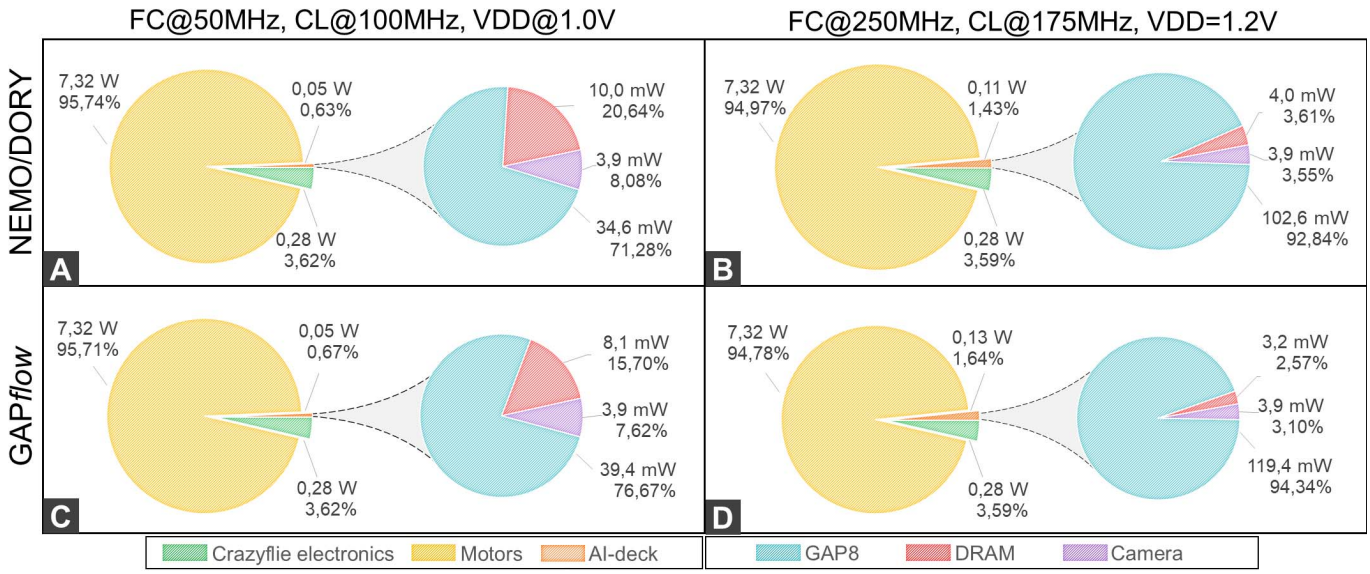


Fig. 5. The nano-drone's power envelope break-down, with AI-deck zoom-in. A/B) NEMO/DORY, and C/D) GAP*flow* framework. SoC running at FC@50 MHz, CL@100 MHz (A/C) and FC@250 MHz, CL@175 MHz (B/D), the most energy efficient and maximum performance configurations, respectively.

Therefore, we show how quantization improves memory footprint and inference throughput by comparing the proposed 8-bit model to the quantized V1 baseline (16-bit). On the one hand, the 2× reduction in the data-type format halves the total size of parameters from 0.64 MB to 0.32 MB. On the other hand, it allows for efficient exploitation of the GAP8's SIMD instructions, resulting in a throughput speedup of 1.5-1.6× w.r.t. the baseline. This mismatch in speedups (i.e., memory footprint and throughput gain) can be ascribed to multiple factors, such as *i)* non-MAC and non-accelerable operations, and *ii)* non-idealities, e.g., imbalanced workload and marshaling overheads.

2) State-of-the-Art Comparison: We validate the two proposed GAP8's pipelines, comparing their performance against one of the most popular CNN libraries for MCUs: CMSIS-NN [40]. CMSIS-NN peaks at 0.71MAC/cycle on 8-bit data convolutions [40] on a CNN's layer similar to our 3 × 3 convolution, for which we achieve at best 0.81MAC/cycle/core and 1.0MAC/cycle/core for the GAP*flow* and NEMO/DORY, respectively. Considering all the inevitable non-idealities, such as sub-optimal load balancing, we yield a weighted performance, for the entire CNN, of 3.9MAC/cycle (GAP*flow*) and 4.7MAC/cycle (NEMO/DORY), employing all the GAP8's cores. To concertize this comparison, we consider the CMSIS-NN on top of a high-performance Cortex-M7-based single-core STM32H723VE, which can achieve up to 390MMAC/s @ 203 mW, running at 550 MHz. The GAP8, at the maximum frequency of CL@175 MHz, achieves 1.44GMAC/s @ 102 mW and 1.14GMAC/s @ 119 mW with NEMO/DORY and GAP*flow*, respectively. This analysis shows that, whether using the GAP*flow* or the NEMO/DORY pipeline, the GAP8 outperforms the Cortex-M7-based MCU with CMSIS-NN by more than 1.7× in power consumption and by more than 3× in throughput. The throughput is of high importance because it has a significant impact on the navigation capabilities, as showed and discussed in Section V-C.

3) Power Break-Down: This section analyzes the power-breakdown of the entire nano-UAV running both PULP-Dronet V2 pipelines – i.e., NEMO/DORY and GAP*flow*. We frame this investigation also considering – for each PULP-Dronet V2 version – the two GAP8's operating points introduced in Section V-B, called *most energy efficient* and *maximum performance*, respectively running at FC@50 MHz CL@100 MHz, and FC@250 MHz CL@175 MHz.

The analysis in Figure 5 refers to three main parts: *i*) the nano-drone's motors, *ii*) its basic electronics running the stock flight controller, and *iii*) the AI-deck executing our visual-workloads. The electronics slice accounts for both flight controller MCUs (i.e., STM32 and nRF51) and all the basic platform's sensors (e.g., IMU, barometer). Additionally, for all four configurations, we also report a break-down zoom-in on the three main AI-deck's components: *i*) the GAP8 SoC, *ii*) the off-chip DRAM, and *iii*) the ULP camera. The reader should note that the DRAM is considered active at full speed only for the time required to copy the CNN's parameter from L3 to L2 and otherwise turned off. Similarly, Flash memory is not considered in this power break-down evaluation, as it is necessary only for the system initialization (i.e., data movement from Flash to DRAM) and then never again used during the drone's mission.

Figure 5 shows how the four motors consume the vast majority of the total power budget, i.e., 7.3 W, while the rest of the drone's electronics accounts for 277 mW in all four configurations, as they are always kept at the same operative conditions. Conversely, the power consumption for the AI-deck changes depending on both exploration parameters, but it is never higher than 1.64%, resulting in a motors' power consumption between 94.8% and 96.0% of the total budget.

Authorized licensed use limited to: ETH BIBLIOTHEK ZURICH. Downloaded on December 16,2021 at 14:16:18 UTC from IEEE Xplore. Restrictions apply.

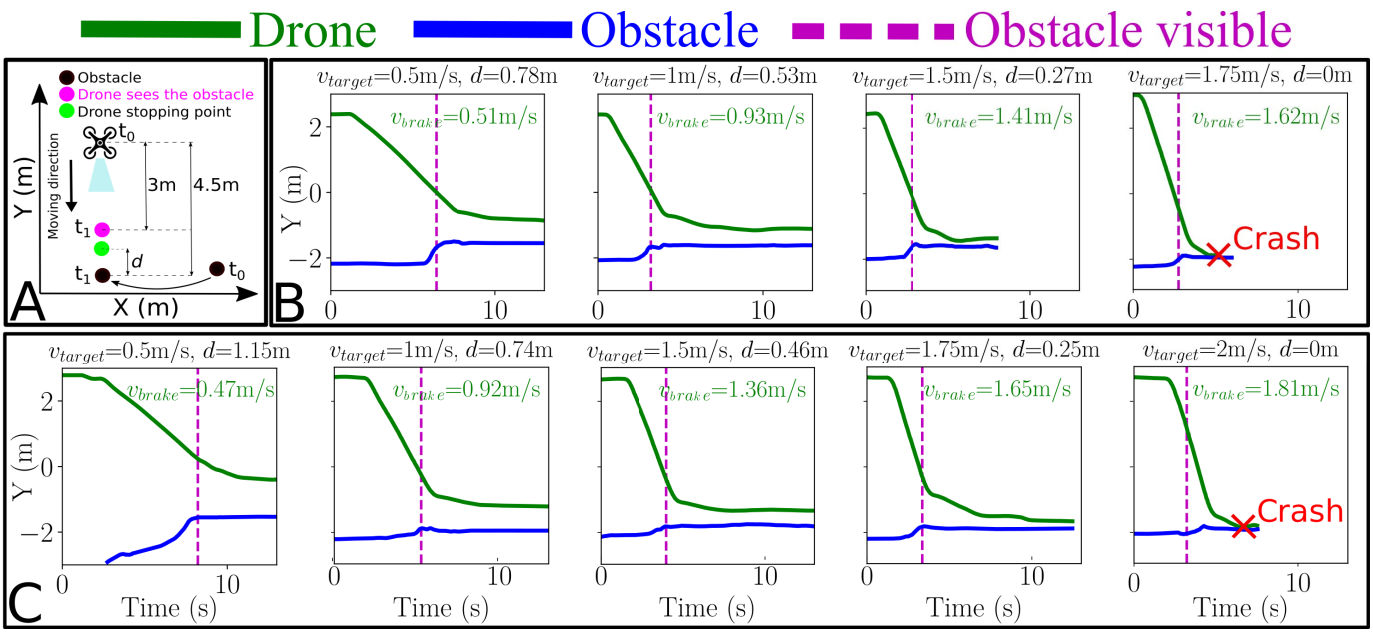


Fig. 6. (A) Experimental setup for the *obstacle avoidance* evaluation. In-field tests are carried sweeping v_{target}, for both the most energy efficient (B) and the maximum performance (C) SoC's configurations.

Focusing on the comparison between the two PULP-Dronet V2 versions, we can identify two main behaviors: *i*) the version developed using the NEMO/DORY framework always shows higher average power consumption for the DRAM and *ii*) the GAP*flow*-based version always has a marginally higher average power consumption for the GAP8 computation. In the most energy efficient configuration, the NEMO/DORY implementation accounts for the 0.63% of the system's power consumption (Figure 5-A), while GAP*flow* accounts for the 0.67% (Figure 5-C). In fact, as shown in Section V-B.1, the GAP*flow*-based implementation brings, on average, to a slightly higher power consumption compared to NEMO/DORY one, 39 mW and 34 mW, respectively. Moreover, due to the L2 data pre-loading during the initialization stage, the GAP*flow* version accesses the DRAM fewer times than its counterpart, resulting in a lower DRAM power consumption w.r.t. NEMO/DORY, i.e., 8 mW and 10 mW, respectively.

Moving to a comparison on the SoC's operating points, the max performance configuration gives a slight advantage to the NEMO/DORY version of PULP-Dronet, which makes the AI-deck consuming 1.43% of the total system's power (Figure 5-B), while using GAP*flow* the percentage becomes 1.64% (Figure 5-D). This small advantage comes from the fact that the NEMO/DORY version, requiring more L3-L2 data transfer w.r.t. the GAP*flow* version, can benefit more from the increased FC's frequency of the max performance configuration. This difference results in a minimal reduction of the AI-deck's power consumption for the NEMO/DORY-based version, as much as 0.04% and 0.21% compared to the GAP*flow*, for the most energy-efficient configuration and the maximum performance one, respectively. Therefore, from a practical viewpoint, both PULP-Dronet V2 versions perform with a very similar power envelope when deployed on our nano-drone. Ultimately, in all four configurations, we can remark how the addition of the AI workload to our quadrotor only accounts for the smallest portion of the power consumption of the entire system, never higher than 1.64%. Such a small impact on the whole system's power budget demonstrates the capability of running the PULP-Dronet V2 at the highest performance point, only marginally impacting the quadrotor lifetime. This enables the possibility to further extend the onboard intelligence with additional tasks (e.g., tracking, detection, localization), aiming at more complex mission objectives.

C. In-Field Closed-Loop Evaluation

In the following, we perform the in-field evaluation of the navigation capabilities of the PULP-Dronet V2, using the implementation generated by the GAP*flow* framework, and deploying it on a Crazyflie 2.1 nano-drone equipped with an additional AI-deck, as illustrated in Section III. We focus on four key aspects to assess the performances of our closed-loop nano-UAV: *i*) the obstacle avoidance task; *ii*) the lane following task; *iii*) the longest flight distance in a familiar environment; *iv*) the generalization capability, testing the autonomous navigation in never-seen-before environments.

1) Obstacle Avoidance Task: One of the two outputs of the Dronet CNN is the probability of collision used to predict a potential obstacle in the path followed by the nano-drone. In this set of experiments, we assess the drone's robustness in avoiding dynamic obstacles by stressing the closed-loop system with an ad-hoc setup, as shown in Figure 6-A. The drone flies a straight trajectory of 4.5 m where a dynamic obstacle (i.e., 0.7×0.7 m cardboard sheet) appears at the end of the path, leaving only 1.5 m for braking and avoiding the collision – i.e., *braking-space*. The straight flight is enforced

Authorized licensed use limited to: ETH BIBLIOTHEK ZURICH. Downloaded on December 16,2021 at 14:16:18 UTC from IEEE Xplore. Restrictions apply.

by silencing the steering angle output of the CNN – i.e., always 0. We perform and record all experiments in a room equipped with a mm-precise motion capture system @ 50 Hz (i.e., Vicon) to analyze the drone's behavior in post-processing.

We investigate this scenario by sweeping two key parameters: *i*) the drone's *target forward velocity* (v_{target}), i.e., a software parameter representing the forward velocity the drone tries to reach if no obstacle is detected, and *ii*) the CNN's inference throughput by means of the two SoC's configurations, introduced in Section V-B, named *most energy-efficient* and *max performance*. This evaluation is depicted in Figure 6-B for most energy-efficient configuration (FC@50 MHz, CL@100 MHz) peaking at 8.7frame/s, and in Figure 6-C for the maximum performance one (FC@250 MHz, CL@175 MHz) up to 12.8frame/s.

We stress the system with a growing velocity v_{target} that takes the following values: 0.5m/s, 1.0m/s, 1.5m/s, and 1.75m/s in Figure 6-B, and 0.5m/s, 1.0m/s, 1.5m/s, 1.75m/s, and 2.0m/s in Figure 6-C. As the v_{target} is a software parameter, we also report, for each test, the actual peak velocity when the drone begins braking (v_{brake}) recorded with the Vicon. We stop this incremental procedure once we reach the limit for which the nano-drone can not prevent the collision anymore. Additionally, in all plots, we highlight a dashed vertical line marking the time when the moving obstacle appeared in the drone's view.

The system proves to be fully working up to $v_{brake} = 1.41$m/s and $v_{brake} = 1.65$m/s, in the most energy-efficient configuration and the maximum performance one, respectively. We can notice how the maximum performance configuration provides a similar safety distance ($d =\sim 0.25$ m) w.r.t. its counterpart, despite their different v_{brake}, thanks to the higher inference throughput (i.e., 12.8frame/s vs. 8.7frame/s). Lastly, referring to the PULP-Dronet baseline [3]; for the same FC@50 MHz, CL@100 MHz configuration, we score a 25.3% higher v_{brake}*/braking-space* ratio, confirming not only the successful deployment of our PULP-Dronet V2 on the COTS Crazyflie nano-drone but also increased promptness of the system.

2) Lane Detection Task: In the following experiments, we assess the PULP-Dronet V2 capability to predict the correct steering angle under two controlled curvature scenarios: a smooth turn of 45°, and a more challenging 90° (i.e., sharp turn). The setup consists of a path 4.5 m long and 1.3 m wide with a left-side turn in the middle, as depicted in Figure 7-A/B with the dotted lines indicating the boundaries of the path. We explore two parameters *i*) the target forward velocity (v_{target}) and *ii*) the CNN's throughput, like in the previous obstacle avoidance experiments, utilizing the two SoC's configurations introduced in Section V-B. The software parameter v_{target} is swept with a granularity of 0.25m/s for the 45° case, and a smaller 0.1m/s growing-step for the 90° setup, due to its higher complexity.

Starting from $v_{target} = 0.5$m/s for the 45° setup (Figure 7-A) and $v_{target} = 0.3$m/s for the 90° one (Figure 7-B), we keep increasing the v_{target} until the system reaches its limit (i.e., collision). This exploration defines the actual maximum average velocity (v_{avg}) – including the initial acceleration phase – for which the nano-drone can complete this lane detection task, resulting in:

- scenario 45°, configuration FC @ 50 MHz CL @ 100 MHz: maximum $v_{avg} = 1.29$m/s;
- scenario 45°, configuration FC @ 250 MHz CL @ 175 MHz: maximum $v_{avg} = 1.47$m/s;
- scenario 90°, configuration FC @ 50 MHz CL @ 100 MHz: maximum $v_{avg} = 0.49$m/s;
- scenario 90°, configuration FC @ 250 MHz CL @ 175 MHz: maximum $v_{avg} = 0.59$m/s;

The increased throughput of the *maximum performance* configuration (12.8frame/s vs. 8.7frame/s), enables higher flight speed in both testing scenarios. As expected, the higher complexity of the 90° scenario is confirmed by a lower maximum v_{avg} compared to the 45° counterpart.

TABLE III
STEERING ANGLE RMSE AND ACTUAL AVERAGE VELOCITY

			SoC configuration			
			FC@50/CL@100 MHz		FC@250/CL@175 MHz	
		v_{target} [m/s]	RMSE [m]	v_{avg} [m/s]	RMSE [m]	v_{avg} [m/s]
Turn curvature	45°	0.5	**0.07**	0.63	0.26	0.62
		0.75	0.12	1.07	0.21	0.78
		1	**0.07**	0.92	0.22	1.32
		1.25	0.17	**1.29**	**0.20**	1.36
		1.5	collision	1.57	0.23	**1.47**
	90°	0.3	**0.14**	0.36	**0.11**	0.34
		0.4	0.17	**0.49**	0.16	0.44
		0.5	collision	0.63	0.20	**0.59**

The 2D trajectories of all tests are reported in Figure 7, where we define as *ground truth* (dashed lines) the trajectory an ideal nano-drone would follow, keeping the path center for the entire flight. Comparing the actual trajectories with the ground truth one, in Table III, we report the root mean squared error (RMSE) for all tests. For both scenarios, we can see a general trend where the higher the actual velocity is (v_{avg}) the more the RMSE grows, ranging between 0.07-0.26 m and 0.11-0.20 m, for the 45° and 90° case, respectively.

A second interesting trend can be seen in the variation of the RMSE between the two SoC configurations of the 45° scenario. At a first look, it seems that a higher throughput penalizes the system's capability, increasing the RMSE. However, by looking at the drone's trajectories in Figure 7-A (right plot), it is clear how the successful tests are clustered into two groups:

- tests v_{target} 0.5 and 0.75 (yellow and blue curves) tend to follow a shorter path trajectory;
- all the other curves exhibit a right-hand drive policy fostered by the dataset labeled with steering angles (i.e., Udacity samples are collected in the US).

In both cases, the ultimate trajectory is slightly away from the ideal central ground truth, increasing the RMSE but still accomplishing the mission.

3) Longest Flight Distance: In this set of experiments, we want to assess our closed-loop system's autonomous navigation capability in a free-flight mission, exploring a "friendly" environment. For this purpose, we select as mission field

Authorized licensed use limited to: ETH BIBLIOTHEK ZURICH. Downloaded on December 16,2021 at 14:16:18 UTC from IEEE Xplore. Restrictions apply.

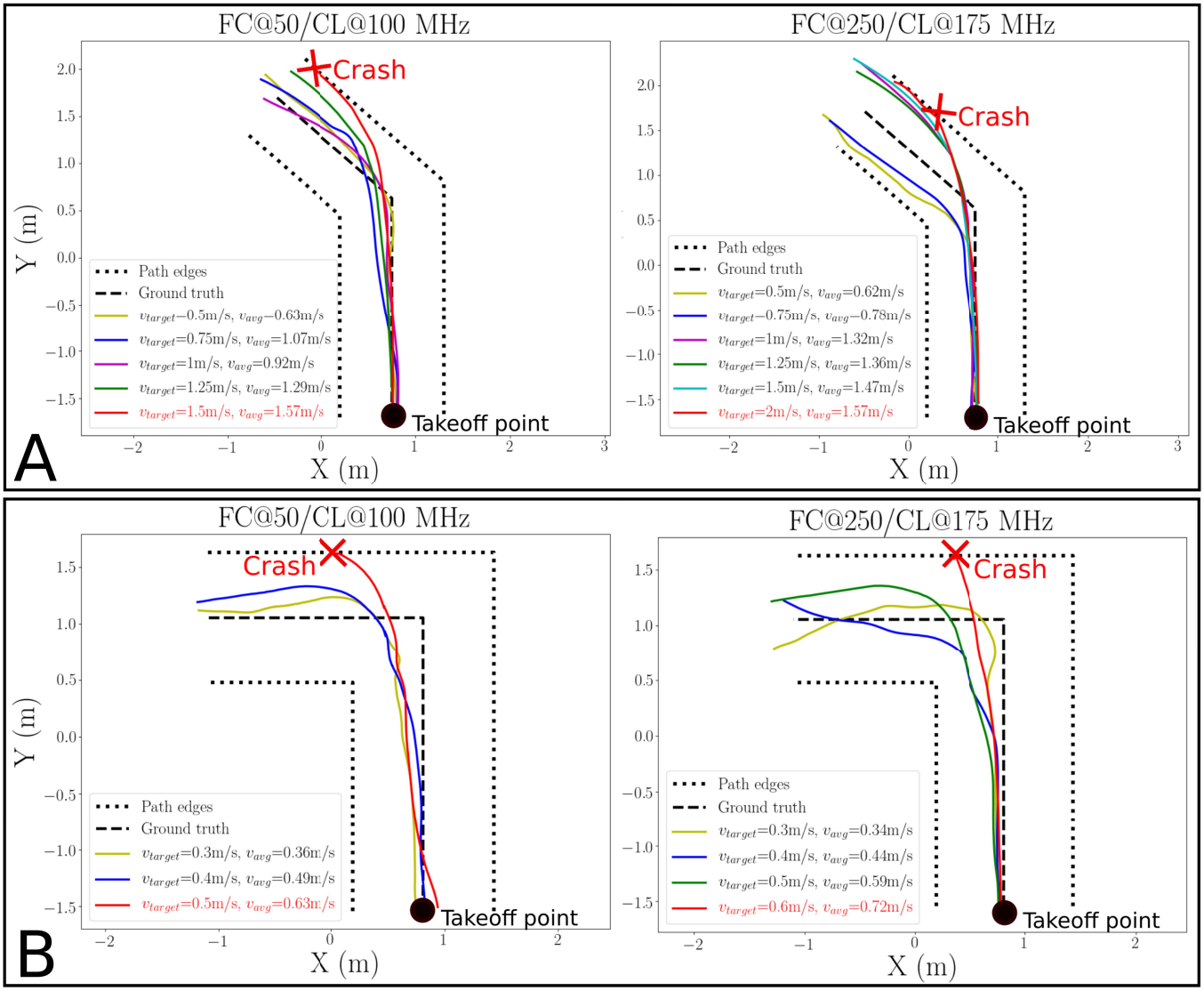


Fig. 7. Lane detection task evaluation. We assess the CNN's capability of predicting the correct (ground truth) steering angle in a scenario featuring a left-side turn–45° (A) 90° (B)–at the center of the path. We sweep the forward target velocity (v_{target}) identifying the limit of our system–in red failing configurations.

the same 110 m-long corridor (U-shape) used for collecting part (16%) of the Himax dataset images. The mission field presents only static obstacles (e.g., walls, doors, and furniture), where we perform 25 tests, sweeping the v_{target} parameter. We employ a growing-step of 0.5m/s, from $v_{target} = 0.5$m/s to 2.5m/s, testing each configuration 5 times. All these experiments are made by selecting the maximum performance SoC's configuration (FC@250 MHz, CL@175 MHz), able to deliver up to 12.8frame/s inference.

In Table IV, we summarize all the experiments for each v_{target} configuration, reporting the average flight time over the successful runs and highlighting in bold the peak performances. We achieve the maximum success-rate (5 success out of 5 tests, per configuration), with an actual mean flight velocity (v_{avg}) from 0.51 to 1.72m/s. Increasing the v_{avg} to 1.96m/s, lowers the success-rate to 80%, defining the performance upper bound of our system, as increasing even further the v_{avg} to 2.29m/s the success-rate quickly drops to 20%. With such high velocity also comes an increased acceleration the drone applies to reach the desired v_{target}, resulting in a high positive pitch. Therefore, the camera mostly captures the floor, which turns in a minimal time to react to the obstacles, flying at high speed. These results prove a superior performance compared to PULP-Dronet V1, which for the same flown distance reports an average velocity of 0.5m/s.

TABLE IV

EVALUATION OF THE FLIGHT TIME AND AVERAGE VELOCITY WHEN THE DRONE FLIES THROUGH A 110 m LONG CORRIDOR

v_{target} [m/s]	v_{avg} [m/s]	Average time [s]	Distance [m]	Success rate
0.5	0.51	216	110	**5/5**
1	0.98	112	110	**5/5**
1.5	1.72	64	110	**5/5**
2	1.96	56	110	4/5
2.5	**2.29**	**48**	110	1/5

Authorized licensed use limited to: ETH BIBLIOTHEK ZURICH. Downloaded on December 16,2021 at 14:16:18 UTC from IEEE Xplore. Restrictions apply.

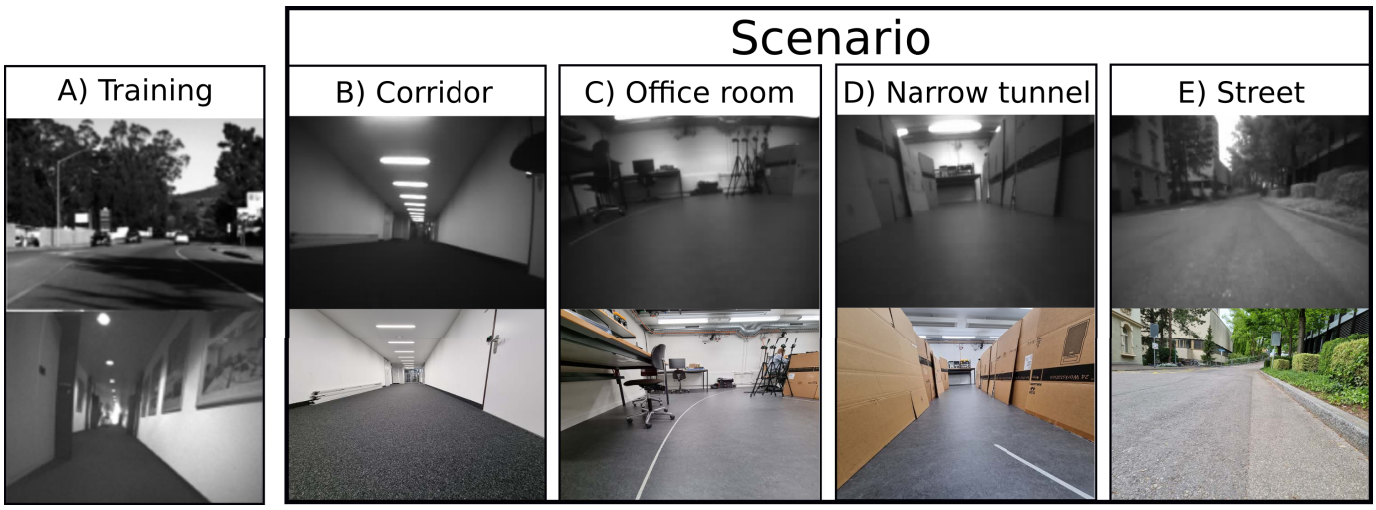


Fig. 8. Samples of: A) the training images (both Udacity and Himax dataset); B) working-place corridor; C) office room with furniture; D) narrow pipeline-like tunnel; E) public street.

While PULP-Dronet V1 used a linear mapping between the probability of collision and the forward velocity, we extended this mechanism with a quadratic relation. This results in a more significant reduction in the forward velocity while approaching an obstacle, which gives the drone more time to steer when flying at high velocities (prior to steering).[9] The main limitation of this experiment is that the corridor is "known" to PULP-Dronet V2 as the training set contains images of the corridor: in the next section, we explore the network's capability to generalize to never-seen-before testing environments.

4) Generalization Capability: As the last part of our in-field evaluation, we present a set of functional experiments aiming at demonstrating the robustness of the PULP-Dronet V2 CNN, testing the closed-loop system in different deployment scenarios. As reported in Table V, and showcased in Figure 8, we explore four different application scenarios, namely:

1) **corridor:** a working-place corridor that is not part of the Himax dataset;
2) **office room:** a room with tables and lab equipment;
3) **narrow tunnel:** a narrow pipeline-like tunnel ($\sim$5 m long and $\sim$1.2 m wide) made of cardboard;
4) **street:** a public street, with road signs and cars.

Orthogonal to the four scenarios, we also consider a second testing criterion: the presence/absence of obstacles in the path the nano-drone should follow. For the cases *corridor*, *office room*, and *street*, the obstacle is represented by a person that can be either standing still in the center of the path (i.e., static obstacle) or moving and crossing the trajectory of the drone or stopping in front of it (i.e., dynamic obstacle). Regardless of the type of obstacle, the goal is to adjust the moving direction, avoiding the obstacle. Instead, for the *narrow tunnel* case, we employ a small cardboard panel as an obstacle, still differentiating between a static and dynamic configuration. All these experiments refer to a setup with a drone's mean target velocity $v_{target} = 0.5\text{m/s}$.

[9]As supplementary material, we make available video footage of one run, with $v_{target} = 2.0\text{m/s}$, available at https://youtu.be/41IwjAXmFQ0.

TABLE V
IN-FIELD EVALUATION SCENARIOS OVER MULTIPLE RUNS WITH AND WITHOUT OBSTACLES (MEAN FLIGHT TIME: SUCCESS-RATE)

Scenario		Obstacle		
		None	Static	Dynamic
	1. Corridor	144 s : 6/6	137 s : 3/6	63 s : 4/6
	2. Office room	86 s : 5/6	78 s : 4/6	67 s : 5/6
	3. Narrow tunnel	12 s : 5/6	0 s : 0/6	19 s : 4/6
	4. Street	171 s : 6/6	148 s : 6/6	148 s : 6/6

Considering the training datasets shown in Figure 8-A, the selected deployment fields introduce a significant difference between the visual cues present in the in-field images and those the CNN has been trained with. The main two sources of difference between the training and deployment can be ascribed to *i*) environmental conditions (e.g., absence of road lane signs in the street) *ii*) photometric/geometric differences between the cameras used to acquire the vast majority of the training dataset and the one available on the mission drone (e.g., field-of-view and resolution).

In Table V, we report the results in terms of mean flight time (across multiple runs) and success-rate, for each case. For scenarios 1, 2, and 4, we consider the test successful if the drone follows the path, with no crashes, for at least 60 s. Instead, for scenario 3, we define as success criteria the capability of the nano-drone to complete the exploration of the entire narrow tunnel. Among all successful cases, the mean flight time spans from 12 s to 171 s, for scenario 3 with no obstacles and scenario 4 with no obstacles, respectively.

The PULP-Dronet V2 sample application shows a high success rate for all configurations except for the *narrow tunnel* with static obstacles, in which case the nano-drone gets stuck in the obstacle proximity, slowly drifting towards it, until it suddenly crashes. As introduced in Section III, the training datasets have disjoint labels, with the Udacity set providing only steering labels and both Bicycle and Himax sets coupled with only collision ones. Therefore, the actual tasks learned by the CNN can be defined as *"predict the presence of obstacles"*

Authorized licensed use limited to: ETH BIBLIOTHEK ZURICH. Downloaded on December 16,2021 at 14:16:18 UTC from IEEE Xplore. Restrictions apply.

and *"predict the steering to follow the lane"*, but not explicitly *"predict the steering to prevent a collision"*. This limit of the Dronet CNN (all versions) could be mitigated in the future by either introducing a new training dataset providing both labels for each image sample or introducing an additional level of intelligence between the CNN and the low-level control.

However, this group of tests aims at assessing the generalization capability of the PULP-Dronet V2 baseline model without introducing any additional deep learning technique, such as deep domain adaptation [45], dataset augmentation [4] or continual learning [46], as they would be out of the scope of this work. All the nine configurations presented in this evaluation are provided of video footage of the experiments available at https://youtu.be/Cd9GyTl6tHI.

VI. Conclusion

The recent progress in deep learning research opens new possibilities for using vision-based end-to-end neural networks to enable nano-drone autonomous navigation. The MCUs found in nano-drones have limited memory and computational resources, and therefore they are unable to run complex CNN models in their original form. However, the available solutions for complexity reduction of the CNNs used to facilitate navigation mainly involve hand-crafted modifications and typically require multiple iterations. This paper fills this gap by analyzing and integrating tools and methodologies that automate this optimize-and-deploy process, automating fine-grained hardware-aware tuning of the CNN. We perform an extensive experimental evaluation of the proposed flow, using a SoA CNN for autonomous nano-drone navigation [1], achieving $\sim 3{-}4\text{mJ/frame}$ inference and reducing the memory requirements by $2\times$ and improving the throughput by $1.6\times$ while preserving the same $\sim 90\%$ classification accuracy of the original implementation. Furthermore, we perform an in-field evaluation of the navigation capabilities of a nano-drone, considering the collision avoidance, steering capabilities, maximum flown distance, and generalization in never-seen-before environments. We record a maximum indoor flight distance of 110 m and an average velocity of 1.96m/s, $4\times$ higher than our PULP-Dronet baseline. We foster the research community releasing as open-source our code and models: https://github.com/pulp-platform/pulp-dronet.

Acknowledgment

The authors would like to thank C. Cioflan, P. Mayer, and M. Nikolov for their assistance in recording the supplementary videos. Open-source code and dataset are available at: https://github.com/pulp-platform/pulp-dronet. In-field experiments video footage at: https://youtu.be/41IwjAXmFQ0, https://youtu.be/Cd9GyTl6tHI.

References

[1] D. Palossi, A. Loquercio, F. Conti, E. Flamand, D. Scaramuzza, and L. Benini, "A 64-mW DNN-based visual navigation engine for autonomous Nano-drones," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 8357–8371, Oct. 2019.

[2] A. Gomez, M. Magno, M. F. Lagadec, and L. Benini, "Precise, energy-efficient data acquisition architecture for monitoring radioactivity using self-sustainable wireless sensor nodes," *IEEE Sensors J.*, vol. 18, no. 1, pp. 459–469, Jan. 2018.

[3] D. Palossi, F. Conti, and L. Benini, "An open source and open hardware deep learning-powered visual navigation engine for autonomous nano-UAVs," in *Proc. 15th Int. Conf. Distrib. Comput. Sensor Syst. (DCOSS)*, May 2019, pp. 604–611.

[4] D. Palossi *et al.*, "Fully onboard AI-powered human-drone pose estimation on ultra-low power autonomous flying nano-UAVs," 2021, *arXiv:2103.10873*.

[5] R. J. Wood *et al.*, "Progress on 'Pico' air vehicles," in *Robotics Research*, H. I. Christensen and O. Khatib, Eds. Cham, Switzerland: Springer, 2017, pp. 3–19.

[6] G. Loianno, D. Scaramuzza, and V. Kumar, "Special issue on high-speed vision-based autonomous navigation of UAVs," *J. Field Robot.*, vol. 35, no. 1, pp. 3–4, Jan. 2018.

[7] W. Zhao, A. Goudar, J. Panerati, and A. P. Schoellig, "Learning-based bias correction for ultra-wideband localization of resource-constrained mobile robots," 2020, *arXiv:2003.09371*.

[8] E. Flamand *et al.*, "GAP-8: A RISC-V SoC for AI at the edge of the IoT," in *Proc. IEEE 29th Int. Conf. Appl.-Specific Syst., Archit. Processors (ASAP)*, Jul. 2018, pp. 1–4.

[9] *Artificial Intelligence Microcontroller with UltraLow-Power Convolutional Neural Network Accelerator*, Maxim Integrated, San Jose, CA, USA, May 2021.

[10] 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.

[11] D. Palossi, A. Marongiu, and L. Benini, "Ultra low-power visual odometry for nano-scale unmanned aerial vehicles," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 1647–1650.

[12] N. Smolyanskiy, A. Kamenev, J. Smith, and S. Birchfield, "Toward low-flying autonomous MAV trail navigation using deep neural networks for environmental awareness," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2017, pp. 4241–4247.

[13] I. Sa *et al.*, "WeedNet: Dense semantic weed classification using multispectral images and MAV for smart farming," *IEEE Robot. Automat. Lett.*, vol. 3, no. 1, pp. 588–595, Jan. 2018.

[14] B. Bodin *et al.*, "SLAMBench2: Multi-objective head-to-head benchmarking for visual SLAM," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2018, pp. 3637–3644.

[15] N. O. Lambert, D. S. Drew, J. Yaconelli, S. Levine, R. Calandra, and K. S. J. Pister, "Low-level control of a quadrotor with deep model-based reinforcement learning," *IEEE Robot. Autom. Lett.*, vol. 4, no. 4, pp. 4224–4230, Oct. 2019.

[16] G. Shi, W. Honig, Y. Yue, and S.-J. Chung, "Neural-swarm: Decentralized close-proximity multirotor control using learned interactions," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2020, pp. 3241–3247.

[17] O. Andersson, M. Wzorek, and P. Doherty, "Deep learning quadcopter control via risk-aware active learning," in *Proc. 31st AAAI Conf. Artif. Intell.*, San Francisco, CA, USA: AAAI Press, 2017, pp. 3812–3818.

[18] K. N. McGuire, C. De Wagter, K. Tuyls, H. J. Kappen, and G. C. H. E. de Croon, "Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment," *Sci. Robot.*, vol. 4, no. 35, pp. 1–43, Oct. 2019.

[19] I. Daramouskas, I. Perikos, I. Hatzilygeroudis, V. J. Lappas, and V. Kostopoulos, "A methodology for drones to learn how to navigate and avoid obstacles using decision trees," in *Proc. 11th Int. Conf. Inf., Intell., Syst. Appl. (IISA*, Jul. 2020, pp. 1–4.

[20] F. Candan, A. Beke, and T. Kumbasar, "Design and deployment of fuzzy PID controllers to the nano quadcopter crazyflie 2.0," in *Proc. Innov. Intell. Syst. Appl. (INISTA)*, Jul. 2018, pp. 1–6.

[21] O. Dunkley, J. Engel, J. Sturm, and D. Cremers, "Visual-inertial navigation for a camera-equipped 25g nano-quadrotor," in *Proc. IROS Aerial Open Source Robot. Workshop*, 2014, pp. 1–2.

[22] A. Anwar and A. Raychowdhury, "Autonomous navigation via deep reinforcement learning for resource constraint edge nodes using transfer learning," *IEEE Access*, vol. 8, pp. 26549–26560, 2020.

[23] A. Suleiman, Z. Zhang, L. Carlone, S. Karaman, and V. Sze, "Navion: A 2-mW fully integrated real-time visual-inertial odometry accelerator for autonomous navigation of nano drones," *IEEE J. Solid-State Circuits*, vol. 54, no. 4, pp. 1106–1119, Apr. 2019.

Authorized licensed use limited to: ETH BIBLIOTHEK ZURICH. Downloaded on December 16,2021 at 14:16:18 UTC from IEEE Xplore. Restrictions apply.

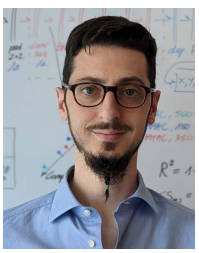
[24] Z. Li *et al.*, "An 879GOPS 243 mW 80fps VGA fully visual CNN-SLAM processor for wide-range autonomous exploration," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2019, pp. 134–136.
[25] J.-H. Yoon and A. Raychowdhury, "31.1 A 65nm 8.79TOPS/W 23.82 mW mixed-signal oscillator-based NeuroSLAM accelerator for applications in edge robotics," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2020, pp. 478–480.
[26] N. K. Manjunath, A. Shiri, M. Hosseini, B. Prakash, N. R. Waytowich, and T. Mohsenin, "An energy efficient EdgeAI autoencoder accelerator for reinforcement learning," *IEEE Open J. Circuits Syst.*, vol. 2, pp. 182–195, 2021.
[27] M. Hosseini and T. Mohsenin, "Binary precision neural network manycore accelerator," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 17, no. 2, pp. 1–27, Apr. 2021.
[28] F. Meneghello, M. Calore, D. Zucchetto, M. Polese, and A. Zanella, "IoT: Internet of threats? A survey of practical security vulnerabilities in real IoT devices," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 8182–8201, Oct. 2019.
[29] S. Bakshi and L. Johnsson, "A highly efficient SGEMM implementation using DMA on the intel/movidius myriad-2," in *Proc. IEEE 32nd Int. Symp. Comput. Archit. High Perform. Comput. (SBAC-PAD)*, Sep. 2020, pp. 321–328.
[30] L. A. Libutti, F. D. Igual, L. Pinuel, L. De Giusti, and M. Naiouf, "Benchmarking performance and power of USB accelerators for inference with MLPerf," in *Proc. 2nd Workshop Accelerated Mach. Learn. (AccML)*, Valencia, Spain, 2020, pp. 1–15.
[31] F. Conti, D. Palossi, A. Marongiu, D. Rossi, and L. Benini, "Enabling the heterogeneous accelerator model on ultra-low power microcontroller platforms," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2016, pp. 1201–1206.
[32] M. Rusci, A. Capotondi, and L. Benini, "Memory-driven mixed low precision quantization for enabling deep network inference on microcontrollers," in *Proceedings of Machine Learning and Systems*, vol. 2, I. Dhillon, D. Papailiopoulos, and V. Sze, Eds. 2020, pp. 326–335.
[33] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4510–4520.
[34] A. Howard *et al.*, "Searching for MobileNetV3," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 1314–1324.
[35] J. Choi, S. Venkataramani, V. V. Srinivasan, K. Gopalakrishnan, Z. Wang, and P. Chuang, "Accurate and efficient 2-bit quantized neural networks," in *Proc. MLSys*, vol. 1, A. Talwalkar, V. Smith, and M. Zaharia, Eds. 2019, pp. 348–359.
[36] B. Jacob *et al.*, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 2704–2713.
[37] M. Gautschi *et al.*, "Near-threshold RISC-V core with DSP extensions for scalable IoT endpoint devices," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 10, pp. 2700–2713, Oct. 2017.
[38] R. David *et al.*, "TensorFlow lite micro: Embedded machine learning on TinyML systems," 2020, *arXiv:2010.08678*.
[39] L. Geiger and P. Team, "Larq: An open-source library for training binarized neural networks," *J. Open Source Softw.*, vol. 5, no. 45, p. 1746, Jan. 2020.
[40] L. Lai, N. Suda, and V. Chandra, "CMSIS-NN: Efficient neural network kernels for arm cortex-M CPUs," 2018, *arXiv:1801.06601*.
[41] 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," *Phil. Trans. Roy. Soc. A, Math., Phys. Eng. Sci.*, vol. 378, no. 2164, Feb. 2020, Art. no. 20190155.
[42] 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.
[43] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
[44] L. Sigrist, A. Gomez, R. Lim, S. Lippuner, M. Leubin, and L. Thiele, "RocketLogger: Mobile power logger for prototyping IoT devices: Demo abstract," in *Proc. 14th ACM Conf. Embedded Netw. Sensor Syst. (CD-ROM)*, Nov. 2016, pp. 288–289.
[45] B. Sun and K. Saenko, "Deep coral: Correlation alignment for deep domain adaptation," in *Proc. Eur. Conf. Comput. Vis.* Amsterdam, The Netherlands: Springer, 2016, pp. 443–450.
[46] H. Ren, D. Anicic, and T. Runkler, "TinyOL: TinyML with online-learning on microcontrollers," 2021, *arXiv:2103.08295*.

Vlad Niculescu received the master's degree in robotics, systems, and control from ETH Zürich in 2019, where he is currently pursuing the Ph.D. degree in electrical engineering with the Integrated Systems Laboratory. During the bachelor's and master's degrees, he competed in more than ten international student competitions, and he was the Electrical Lead of the student project Swissloop, which won the Second Place and the Innovation Award in the SpaceX Hyperloop Pod Competition 2019. His research is now focused on developing localization and autonomous navigation algorithms that target ultra-low-power platforms which can operate onboard nano-drones.

Lorenzo Lamberti (Graduate Student Member, IEEE) received the bachelor's and master's degrees (Hons.) from the University of Bologna, Italy, in 2016 and 2019, respectively, where he is currently pursuing the Ph.D. degree in electronic engineering with the Energy-Efficient Embedded Systems Laboratory. Previously, he has been an Intern with the Fermi National Accelerator Laboratory, Chicago, USA, and the Datalogic's Artificial Intelligence Laboratory, Pasadena, USA. His research is currently targeted at autonomous navigation for nano-size unmanned aerial vehicles. In particular, he focuses on neural architecture search, training, and in-field deployment of deep neural network-based navigation tasks on low-power MCUs.

Francesco Conti (Member, IEEE) received the Ph.D. degree in electronic engineering from the University of Bologna, Italy, in 2016. He is currently an Assistant Professor with the DEI Department, University of Bologna. From 2016 to 2020, he held a research grant at the DEI Department, University of Bologna, and a position as a Post-Doctoral Researcher with the Integrated Systems Laboratory, Digital Systems Group, ETH Zürich. His research focuses on the development of deep learning-based intelligence on top of ultra-low power, ultra-energy efficient programmable systems-on-chip. His research work has resulted in more than 60 publications in international conferences and journals and has been awarded several times, including the 2020 IEEE TCAS-I Darlington Best Paper Award.

Luca Benini (Fellow, IEEE) received the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, USA, in 1997. He has served as the Chief Architect for the Platform2012/STHORM Project with STMicroelectronics, Grenoble, France, from 2009 to 2013. Currently, he holds the Chair of digital circuits and systems with ETH Zürich, Zürich, Switzerland, and is a Full Professor with the University of Bologna, Bologna, Italy. He has published more than 1000 peer-reviewed articles and five books. His current research interest includes energy-efficient computing systems' design from embedded to high performance. He is a fellow of ACM and a member of the Academia Europaea. He was a recipient of the 2016 IEEE CAS Mac Van Valkenburg Award and the 2019 IEEE TCAD Donald O. Pederson Best Article Award.



Daniele Palossi received the Ph.D. degree in information technology and electrical engineering from ETH Zürich, Zürich, Switzerland, in 2019. He is currently a Post-Doctoral Researcher with the Dalle Molle Institute for Artificial Intelligence (IDSIA), USI-SUPSI, Lugano, Switzerland, and the Integrated Systems Laboratory (IIS), ETH Zürich. His work has resulted in more than 20 publications in international conferences and journals. His research focuses on the embedded domain with special emphasis on energy-efficient ultra-low-power platforms, algorithms for autonomous navigation, and resource-constrained small-sized cyber-physical systems. He was a recipient of the Swiss National Science Foundation (SNSF) Spark Grant and the Second Prize at the Design Contest held at the ACM/IEEE ISLPED 2019.

Authorized licensed use limited to: ETH BIBLIOTHEK ZURICH. Downloaded on December 16,2021 at 14:16:18 UTC from IEEE Xplore. Restrictions apply.