Fully Onboard SLAM for Distributed Mapping With a Swarm of Nano-Drones

Carl Friess, Vlad Niculescu, Tommaso Polonelli, *Member, IEEE*, Michele Magno, *Senior Member, IEEE*, and Luca Benini, *Fellow, IEEE*

***Abstract*—The use of unmanned aerial vehicles (UAVs) is rapidly increasing in applications ranging from surveillance and first-aid missions to industrial automation involving cooperation with other machines or humans. To maximize area coverage and reduce mission latency, swarms of collaborating drones have become a significant research direction. However, this approach requires open challenges in positioning, mapping, and communications to be addressed. This work describes a distributed mapping system based on a swarm of nano-UAVs, characterized by a limited payload of 35 g and tightly constrained onboard sensing and computing capabilities. Each nano-UAV is equipped with four 64-pixel depth sensors that measure the relative distance to obstacles in four directions. The proposed system merges the information from the swarm and generates a coherent grid map without relying on any external infrastructure. The data fusion is performed using the iterative closest point algorithm and a graph-based simultaneous localization and mapping algorithm, running entirely onboard the UAV's low-power ARM Cortex-M microcontroller with just 192 kB of memory. Field results gathered in three different mazes with a swarm of up to four nano-UAVs prove a mapping accuracy of 12 cm and demonstrate that the mapping time is inversely proportional to the number of agents. The proposed framework scales linearly in terms of communication bandwidth and onboard computational complexity, supporting communication between up to 20 nano-UAVs and mapping of areas up to 180 m^2 with the chosen configuration requiring only 50 kB of memory.**

***Index Terms*—Mapping, nano-drone, simultaneous localization and mapping (SLAM), swarm intelligence, unmanned aerial vehicle (UAV).**

I. Introduction

UNMANNED aerial vehicles (UAVs) have emerged as attractive solutions for several applications that require high maneuverability and scalability, such as distributed inspection and surveillance [1]. In particular, as defined in [2], nano-UAVs have proven to be safe to operate near people due to their reduced weight, i.e., below 50 g, which makes

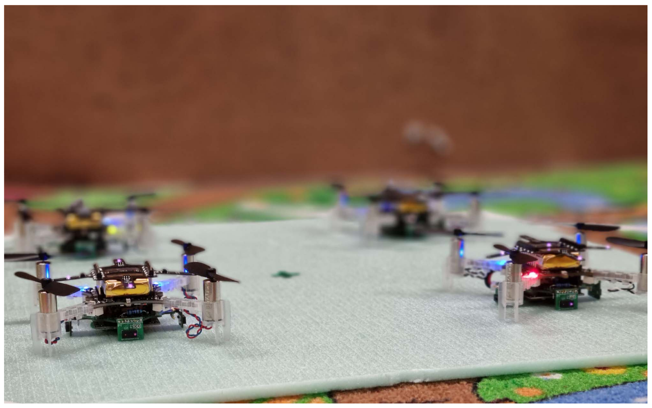


Fig. 1. Swarm of nano-UAVs before take-off. The platform is based on a commercial Crazyflie 2.1 extended with four VL53L5CX ToF ranging sensors and the flow-deck v2. Total weight at the take-off is 34.8 g.

them an excellent choice for navigating through indoor or cramped environments [3]. Furthermore, they are agile, small enough to fit in the palm of a hand, and their cost-effective hardware facilitates swarm formations. An Internet of Robotic Things (IoRT) swarm [4] allows for a decreased latency and a higher probability of reaching the mission objective due to the intrinsic redundancy of having more than one drone [5]. Finding gas leaks [6], localizing survivors in mines [7], remote health monitoring of COVID-19 patients [4], or machine-to-machine cooperation with the Internet of Things (IoT) devices [8] are only a few examples where nano-UAVs and IoRT [9], with a single agent or a swarm, can be employed. Within such applications, UAVs have to perceive the environment and compute their next movements to enable optimal mission strategy [10]. However, enabling optimal planning requires good knowledge of the map of the environment as demonstrated in [10].

Simultaneous localization and mapping (SLAM) is a class of mapping algorithms that can actively correct for both odometry and mapping errors [3]. In particular, graph-based SLAM is widely used due to its high accuracy and ability to store the whole robot trajectory [11], [12]. The algorithm models the drone poses as graph nodes, the odometry measurements as edges, and consists of two main elements: 1) the loop-closure detection, which identifies if the drone has returned to a previously visited location and 2) the graph optimization, which creates an additional edge as the result of the loop closure, optimizing the previously added poses. Moreover, since mapping requires accurate sensing capabilities that can

Manuscript received 27 November 2023; revised 19 January 2024; accepted 7 February 2024. Date of publication 19 February 2024; date of current version 9 October 2024. This work was supported in part by the "TinyTrainer" Project that receives funding from the Swiss National Science Foundation under Grant 207913. *(Corresponding author: Vlad Niculescu.)*

Carl Friess, Vlad Niculescu, Tommaso Polonelli, and Michele Magno are with the Department of Information Technology and Electrical Engineering, ETH Zürich, 8092 Zürich, Switzerland (e-mail: vladn@ethz.ch).

Luca Benini is with the Department of Information Technology and Electrical Engineering, ETH Zürich, 8092 Zürich, Switzerland, and also with the Department of Electrical, Electronic and Information Engineering, University of Bologna, 40126 Bologna, Italy.

Digital Object Identifier 10.1109/JIOT.2024.3367451

measure the depth of the environment, light detection and ranging (LiDAR) and stereo cameras [13] are among the most popular approaches that support SLAM on UAVs today.

Even though the SLAM algorithm paired with LiDARs is widely used [14], [15], it requires computationally intensive and memory-hungry processes, which are not feasible on nano-UAVs due to their limited payload and constrained resources [16]. As an alternative, miniaturized, low-resolution, and energy-efficient Time of Flight (ToF) sensors weighing only 42 mg have been released on the market in the last few years, opening new applications in the field of nano-UAVs [17]. Consequently, recent works have exploited these sensors and enabled SLAM with nano-UAVs. However, due to the low-sensor resolution (i.e., one depth pixel per sensor), their systems are only capable of mapping simple-geometry environments (e.g., long corridors) [3]. Furthermore, they offload the computation to an external base station via a wireless link, which increases the power consumption and restricts the operating area to the radio range [3]. Unlike previous works, we take advantage of the novel VL53L5CX 8×8 ToF matrix sensor, which provides a 64-pixel depth map and offers a Field of View (FoV) of 45°, already characterized in [17]. By mounting four such sensors on the nano-UAV (i.e., front, rear, left, right), we increase the FoV and achieve superior loop-closure performance compared to the previous depth-based solutions for nano-UAVs and micro-robots [3], [18].

By projecting the depth measurements acquired over a short time frame into the coordinate system of the drone, we obtain a small map that we call a *scan*. When the drone revisits a location, it acquires another scan and determines the rigid-body transformation with respect to the scan acquired when it first visited that location. This type of loop-closure detection is called *scan-matching*. Furthermore, in this work, we implement and use an optimized version of the iterative closest point algorithm (ICP), a State-of-the-Art (SoA) in scan-matching that works with an arbitrary environment geometry. In addition, we integrate the ICP with graph-based SLAM, which runs entirely onboard the nano-UAV and achieves full SLAM functionality.

This article proposes a precise system for distributed indoor mapping with a swarm of nano-UAVs exploiting novel and low-power depth sensors. The entire computation runs onboard the nano-UAVs without relying on any external computer. Our contributions can be summarized as follows.

1) An optimized implementation and evaluation of the ICP algorithm. It runs entirely onboard a resource-constrained microprocessor in about 500 ms for the input size used in our evaluation with an expected translation accuracy of about 3 cm. Due to the combination with the 8×8 ToF matrix sensors, this is the first work that enables ICP onboard nano-UAVs.
2) A distributed and autonomous exploration policy with obstacle avoidance that allows multiple drones to explore an unknown environment with different flight paths and moving obstacles.
3) The computationally lightweight integration between ICP and SLAM, and its extension to support a swarm of drones. Furthermore, we design and implement a reliable IoRT radio communication protocol that orchestrates how the drones in the swarm exchange poses and scans with each other.
4) An extensive field evaluation that demonstrates the mapping capabilities of our system with a swarm of four nano-UAVs visible in Fig. 1. We prove how our optimized SLAM, combined with ICP, corrects the odometry errors by up to 55% and aligns the world frames of individual drones to generate coherent maps. To our knowledge, this is the first work that enables onboard mapping with a swarm of nano-UAVs, where robots are working in an IoRT network to collect, analyze, and transmit data.

II. Related Work

Standard-size UAVs differ from micro-aerial vehicles (MAVs) and nano-UAVs in terms of size, weight, total power consumption, and onboard processing capabilities. The latter two are directly linked as the budget for onboard electronics, including sensing and processing, is about 1/10 of the total motor power [16]. Today, most new advancements in the SoA regarding robotic perception and mapping have been demonstrated on standard-size UAVs and MAVs, which feature a power budget between 50 and 100 W and a total mass $\geq$1 kg, as reported in [19]. Hence, they feature powerful onboard computing platforms, often equipped with graphics processing units (GPUs) and several gigabytes of memory [19]. On the other hand, this work focuses on nano-UAVs in particular. They weigh less than 50 g with a total power budget on the order of 5-10 W, of which only 500 mW–1 W remain for the sensors and micro controller units (MCUs) [16]. Moreover, low-power MCUs mostly support a limited amount of memory, in general between 100 kB and 500 kB, a stringent limitation for visual-based perception and mapping. This article tackles these unsolved research challenges, improving the mapping capabilities of nano-UAVs in single agent or swarm formation.

Previous works on MAVs and UAVs have commonly relied on miniature, conventional 360° LiDAR sensors [20] or depth stereo camera [19] for mapping purposes. In particular, this article [21] integrates single-layer LiDAR sensors with inertial measurement units (IMUs) for indoor mapping tasks. The platform used is the commercial DJI Phantom 3 drone with an additional desktop-class Intel i5 processor required onboard. The LiDAR sensor used is 62 mm × 62 mm × 87.5 mm in size and weighs 210 g, while nominally consuming 8.4 W of power. Despite being effective, the setup in [21] is unrealistic for a nano-UAV, as depicted in Fig. 1, which features a total power budget below 10 W. Using a similar configuration, the study in [22] integrates a multilayer LiDAR sensor to allow 3-D mapping of indoor environments while also relying on a desktop-class Intel i7 processor. Although the LiDAR sensor in this case only consumes 8 W of power, its footprint is larger (103.3 mm × 103.3 mm × 71.7 mm) and weighs 509 g. On the other hand, Fang et al. [23] leveraged an RGB-D camera combined with a particle filter for navigating obstructed and visually degraded shipboard environments. The used platform is 58 cm × 58 cm × 32 cm in size, carries more than 500 g of instrumentation, and operates on a high-performance octa-core ARM processor. Again, the payload requirements and computational demands do not fit this article's scope. SoA mapping strategies are also investigated in the UAV field, as reported in Table I, in terms of sensors, mapping accuracy, swarm size, and power consumed by the computing platforms. Causa et al. [24] proposed a cooperative mapping strategy based on LiDAR and global navigation satellite system (GNSS), relying on a standard size UAV of 3.6 kg and off-board processing. Shen et al. [25] brought the intelligence fully onboard, relying on a power-hungry (i.e., 30 W) Nvidia Xavier and a VLP-16 LiDAR. Also, in [15], the mapping algorithm and onboard processing are entrusted to a Jetson TX2 featuring a multicore central processing unit (CPU) and a GPU, as well as 8 GB of memory. Chang et al. [14] proposed a robust multirobot SLAM system designed to support robot swarms; however, results are validated offline on an Intel i7-8750H processor. Although these SoA approaches provide good mapping capabilities in the range of 5 to 20 cm, they involve large and heavy sensors that require processing platforms consuming a few tens of watts. These sensors and processors are too power hungry to operate onboard nano-UAVs and have an unsustainable total weight. Thus, it is clear that standard SoA approaches for autonomous navigation and mapping are not computationally optimized and cannot be deployed in the nano-UAV field [26], which is the scope of this article. However, in recent years, lightweight alternative sensing solutions, which are more appropriate for nano-UAV platforms, have become available [17], [26]. Indeed, despite implementing only a basic obstacle avoidance strategy, Müller et al. [26] exploited an ultralightweight (42 mg) depth sensor and an optimized navigation algorithm running entirely onboard a nano-UAV. The concept behind [26] has been extended for the scope of this article—enabling distributed mapping on miniaturized robotic platforms.

In [27], simple ToF ranging sensors facing forward, backward, left, and right on a Crazyflie drone have been used to implement mapping. Although this work implements basic obstacle avoidance, manual piloting from a ground station is required, and no method for compensating odometry drift is implemented. Using the same hardware configuration, Duisterhof et al. [7] implemented obstacle avoidance for source seeking using deep reinforcement learning. However, since this method does not map the environment, the path is often suboptimal. Mapping using these simple ranging sensors is possible, but the low resolution of the acquired data means that longer flight times are required to approach the mapping fidelity of traditional drone-based systems while still suffering from odometry drift. Mapping methods can be improved by applying SLAM algorithms to correct pose estimation errors, which is the main scope of this article.

A mapping pipeline typically requires two stages: first, the robot must close the loop using scan-matching and then use the scan-matching information to correct the trajectory. ICP represents the SoA for performing scan-matching, and multiple variations were proposed in the literature. Pan et al. [29] proposed MULLS ICP, which formulates the scan-matching as a weighted least squares problem. Although it provides accuracy in the order of a few tens of centimeters, it requires tuning several parameters, and thus, it was discarded for this work. Dellenbach et al. [30] proposed CT-ICP, a particular form of ICP that also corrects the measurement distortion introduced by robot motion. They achieve a mean translation error of 0.55 m, but their approach requires knowing the robot motion profile. Furthermore, they determine the correspondences between the scan points using the point-to-plane approach, which is computationally demanding for a nano-UAV onboard MCU. Recently, Vizzo et al. [31] released Kiss-ICP, which overcomes the issues associated with the previous ICP works and achieves a translation error of 0.49 m. Furthermore, they use the point-to-point correspondence between the scan points, which is computationally faster than the point-to-plane or point-to-line alternatives. Therefore, this approach was employed for the scope of this article. All the mentioned scan-matching works use ICP paired with LiDARs to enable odometry estimation while the robot is moving. In our work, we use ICP paired with the same correspondence mechanism as in [31], but only perform scan matching when a loop closure is detected. Furthermore, within this article scenario, the drone is stationary while acquiring the scans, leading to a translation error of less than 10 cm despite the sparse input information provided by the ToF sensors. The execution time of the SoA is in the range of 26–83 ms [29], [30], [31], when running on commodity computers with general-purpose CPUs, such as the Intel i7-7700HQ. On the other hand, our system requires an execution time of about 500 ms with a power consumption that is two orders of magnitude smaller with respect to an Intel i7 processor.

The subfield of SLAM problems known as "SLAM with sparse sensing" [32] explores more challenging applications where robots receive data points with a lower frequency and accuracy, as is the case for sensing solutions suitable for nano-UAVs [33]. For instance, the work in [34] proposes to solve this problem by extracting line features as landmarks and applying a Rao–Blackwellized particle filter (RBPF) [35]. However, particle filters are not a favorable approach for larger maps since large numbers of particles are required to maintain accuracy [36]. With this in mind, results in [3] demonstrate the efficient offline use of graph-based SLAM on sparse data with a novel front-end for scan matching. Although this method uses an Intel i7 desktop-class processor, it was also evaluated using data collected from a Crazyflie drone using single-zone ToF sensors, a setup similar to this article. Following up on [27], the method presented in [18] shows a similar approach to applying SLAM to compensate for odometry drift; however, it is still being computed offline. The approach is further limited by the single-zone sensors, which are mounted on a larger drone (182 mm × 158 mm × 56 mm) based on the Crazyflie Bolt.

Works, such as [37], used ranging sensors to implement SLAM onboard differential wheeled robot platforms using embedded application processors. Furthermore, Zhu et al. [38] used an application processor and cameras to implement visual SLAM onboard a wheeled consumer robot platform, while in [39] a visual SLAM algorithm optimization is discussed to

TABLE I
SYSTEM AND PERFORMANCE COMPARISON BETWEEN THIS ARTICLE AND THE SOA WORKS PRESENT IN THE LITERATURE. ONBOARD PROCESSING, SENSING ELEMENTS, MAPPING ACCURACY, AND SYSTEM SETUPS ARE COMPARED

Work	Onboard processing	Sensor	ToF Pixels	Map accuracy	Field test	Multi-robot Swarm	Power Consumption	System Weight
				Nano-UAV and MAV				
This work	**Yes (Cortex-M4)**	**4× ToF VL53L5CX**	**256**	**10-15 cm**	**Yes (4 drones)**	**Yes (up to 20 drones)**	**240 mW**	**34.8 g**
[27]	No	4× ToF VL53L1x	4	10-20 cm	Yes	No	-	401 g
[7]	Yes (Cortex-M4)	4× ToF VL53L1x	4	-	Yes	No	240 mW	31.7 g
[3]	No (Intel i7 station)	4× ToF VL53L1x	4	5-15 cm	No	No	-	-
[18]	No	4× ToF VL53L1x	4	4.7 cm	Yes	No	-	401 g
[26]	Yes (Cortex-M4)	1× ToF VL53L5CX	64	no map	Yes	No	320 mW	35 g
[28]	Yes (GAP9)	4× ToF VL53L5CX	256	8-10 cm	Yes	No	350 mW	44 g
				Standard-size UAV				
[24]	No	LIDAR	-	5-20 cm	No	Yes	-	3.6 kg
[25]	Yes (Xavier)	VLP-16 LiDAR	-	2.14 m	Yes	No	30 W	>2 kg
[15]	Yes (Jetson TX2)	RP-LIDAR	-	-	Yes	Yes up to 50	7.5 W	1.8 kg
[14]	No (Intel i7 station)	LIDAR	-	15-20 cm	No	Yes	-	-
[19]	Yes (Jetson TX2)	Intel RealSense D435	-	-	Yes	Yes	7.5 W	1.3 kg

exploit better the resources of an open multimedia application platform (OMAP) processor. On the other hand, Beevers and Huang [40] showed an MCU-based solution using a particle filter-based approach [34] that relies on extracting line features. Similarly, a different method called orthogonal SLAM is proposed in [41] and executed on an MCU onboard a drone in [42]. However, this method assumes that the extracted line features are always orthogonal. In contrast, our approach makes use of a fully onboard, highly optimized implementation of graph-based SLAM [43] with scan matching using ICP [31] adapted for a novel sensing solution on a 34.8 g nano-UAV.

Although visual and sparse-distance SLAM mapping is a well-known topic in the robotics community, the computational load and the related processing latency are still a concern in the UAV field [10]. Moreover, the current research trend pushes for collaborative mapping among a group of agents [44], also referred to as IoRT [4], often composed of a heterogeneous set of flying robots and sensing elements [19]. As shown by Table I, only a few works in the literature propose a distributed mapping solution based on a UAV swarm that is successfully implemented in a real experiment [15], in particular without relying on any external infrastructure [14], [19]. In the nano-UAV field, lightweight methodologies and field tests demonstrating mapping capabilities are scarce [3], [7]. A recent publication implementing onboard SLAM with nano-UAVs is titled NanoSLAM [28], which reaches 10 cm mapping accuracy. However, despite the relevant results regarding system integration and latency, NanoSLAM supports only one agent and necessitates a supplementary co-processor, which increases the total weight at take-off to 44 g as shown in Table I, thus decreasing the flight time concerning the solution proposed in this article. Moreover, Niculescu et al. [28] showed how the mapping coverage is directly proportional to the nano-UAV battery lifetime, flying at relatively low speed (below 2 m/s). Therefore, since the flight time is bounded by the technology limitation of the battery capacity and the frame weight, the alternative to increase the mapping coverage is to parallelize the task over a swarm of nano-UAVs. To the best of our knowledge, no works investigating a mapping system based on a swarm of nano-UAVs are present in the literature. Thus, this article proposes the first study, implementation, and field results enabling fully onboard and distributed mapping for the nano-UAV ecosystem. We demonstrate the possibility of supporting up to 20 agents relying only on a 34.8 g hardware platform for sensing, processing, and communication. The achieved accuracy is aligned with the SoA for MAVs and standard-size UAVs, with an average error on the order of 12 cm. The proposed system, including the lightweight perception framework implemented in this article, paves the way for enhancing the autonomous capabilities of nano-UAVs by improving optimal path planning and IoRT multiagent collaboration.

III. SYSTEM SETUP

This section describes the nano-UAV platform selected for this work, deepened in Section III-A, and the supplementary commercial and custom sensors, in Section III-B, used as the basis for our experiments. The onboard computational and sensing capabilities are discussed, as well as the hardware's total weight.

A. Crazyflie Nano-UAV

The Crazyflie 2.1 is an open nano-UAV platform from Bitcraze commonly used in research. Its mainboard also acts as the airframe and includes an IMU, a barometer, a radio (Nordic nRF51822), and an STM32F405 MCU (168 MHz, 196-kB RAM) that handles sensor readout, state estimation, and real-time control. The drone features extension headers that can be used to add commercially available decks (i.e., plug-in boards) to improve its sensing capabilities. In this work, the drone was equipped with the commercial Flow-deck v2, featuring a downward-facing optical flow camera (i.e., PMW3901) and single-zone ToF ranging sensor (i.e., VL53L1CX), which improve velocity and height measurements through sensor fusion by the onboard extended Kalman filter (EKF). The drone's MCU and the mentioned sensors consume about 100 mW. The Crazyflie was further equipped with a custom deck designed and presented in this work featuring four

VL53L5CX ToF ranging sensors—each sensor consuming an additional power of 100 mW. The total weight at take-off is 34.8 g, including all the hardware used for the scope of this article. The final system setup is depicted in Fig. 1. With a 350-mAh battery, the drone can fly for about 6 min.

B. Custom Quad ToF Deck

The VL53L5CX is a multizone 64-pixel ToF sensor featuring an extremely lightweight package, with a total weight of 42 mg and an accuracy of 4 cm [26]. Its performance in the field of nano-UAVs was characterized in [17]. The maximum ranging frequency for the full resolution of 8×8 pixel is 15 Hz, and the FoV is 45°. Moreover, the VL53L5CX provides a pixel validity matrix paired with the 64-pixel measurement matrix, which automatically flags noisy or out-of-range measurements. To enable the use of the multizone ranging sensors with the Crazyflie, we have created a custom deck designed specifically for the VL53L5CX ToF sensor. It can be used simultaneously with the Flow-deck v2 and incorporates four VL53L5CX sensors that face forward, back, left, and right, allowing for the detection of obstacles in all directions. Although lower than in conventional LIDARs, the 8×8 resolution is enough to enable accurate scan-matching. Unlike LIDARs, the novel multizone ToF sensor is a perfect tradeoff between resolution, accuracy, weight, and power consumption for use onboard nano-UAVs. Moreover, we incorporate a 4-MB serial peripheral interface (SPI) flash storage device to store the acquired measurements. The memory type was chosen based on the interfaces available onboard the drone (i.e., SPI). Among the available memory ICs with an SPI interface, flash memory typically has more capacity than SRAM for the same size and much lower latency than a microSD card. The final design of the custom ToF deck only weighs 5.1 g.

IV. Algorithms

This section provides the theoretical background of the lightweight algorithms we implement onboard and how they fit together to solve the mapping problem. Generating an accurate map requires good accuracy of the drone's position estimate, which can be impacted by odometry drift, leading to warped maps. We show how we leverage and combine the information from multiple ToF multizone sensors to achieve scan matching using the ICP algorithm. While ICP can correct the odometry errors in the revisited locations, we show how it can be combined with SLAM to correct the whole past trajectory and therefore, enable accurate mapping. Moreover, ICP is also used to combine data collected from different nano-UAVs in the swarm.

A. Onboard State Estimation

The state estimation relies entirely on onboard sensors, leveraging the IMU, the optical flow camera, and the downward-pointing single-pixel ToF sensor. The IMU provides 3-D acceleration and angular velocity, which are crucial for attitude estimation. The downward-pointing ToF sensor provides absolute height measurements, and its combination with the optical-flow camera enables body-frame velocity

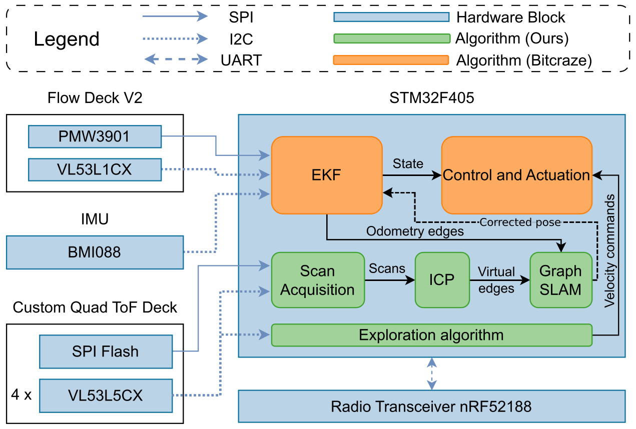


Fig. 2. Overview of the system architecture, including a hierarchical description of the algorithmic pipeline and data interconnections.

measurements. All the sensor information is fused by the onboard EKF, which provides complete state estimation, i.e., position, velocity, and orientation. The EKF and the state controller are functionalities implemented in the open-source Crazyflie firmware. Fig. 2 shows an overview of the system architecture, where the orange software blocks belong to the base firmware. The blocks illustrated in green represent our additions, implemented as separate FreeRTOS tasks and discussed in this section.

B. Scan Frames and Scans

In this work, we tackle the mapping problem in 2-D, and therefore, the localization and mapping are performed in one plane. Since each ToF sensor provides an 8×8 distance matrix, we need to reduce it to an 8-element array that is compatible with our 2-D scenario. Hence, we derive a single measurement from each column of the ToF matrix. To this end, we only consider the center four pixels from each column, discard any invalid pixels, and take the median of the remaining values. Should none of the pixels be valid, the entire column is discarded. This occurs, for example, when there is no obstacle within the 4 m range of the sensor. The mechanism of selecting the median pixel from each row provides a low-pass filtering effect, which mitigates the effect of the high-frequency noise in the nano-UAV's motion. Furthermore, each sensor is polled before the mission starts to ensure the proper functionality and avoid possible installation issues.

Let $\boldsymbol{x}_k = (x_k, y_k, \psi_k)$ be the state of the drone (i.e., *pose*) at timestamp k expressed in the world coordinate frame—in practice, the state information is provided by the onboard EKF. Furthermore, let $i \in \{1, 2, 3, 4\}$ describe index among sensors and $j \in \{1, 2, \dots 8\}$ the index among the zones of each sensor. Thus, d_k^{ij} represents the distance the ith sensor provides for the jth zone. Equation (1) provides the function that projects a distance measurement d_k^{ij} acquired at pose $\boldsymbol{x}_k$ into the world coordinate frame. Fig. 3(a) provides a graphical representation of the variables in (1), and the world and the drone's body frames are represented in black and green, respectively. The heading angle ψ_k represents the rotation between the drone's body frame and the world's frame, $\beta_i \in \{0°, 90°, 180°, 270°\}$

represents the fixed rotation of each sensor with respect to the drone's body frame and $\boldsymbol{R}$ is the 2-D rotation matrix. Furthermore, o_x^i and o_y^i represent the offset of each ToF sensor i with respect to the drone's center O expressed in the body frame of the drone. d_{ij} is the projection of a distance measurement on the OX axis of the ToF sensor's coordinate frame [marked with blue in Fig. 3(a)], and the sensor directly provides it. The y-coordinate of the measurement in the same coordinate frame is calculated as $\tan(\theta_j) \cdot d^{ij}$, where θ_j is the angle of the zone

$$\boldsymbol{s}\left(\boldsymbol{x}_k, d_k^{ij}\right) = \begin{pmatrix} x_k \\ y_k \end{pmatrix} + \boldsymbol{R}_{(\psi_k+\beta_i)} \begin{pmatrix} d_k^{ij} + o_x^i \\ \tan\left(\theta_j\right) \cdot d_k^{ij} + o_y^i \end{pmatrix}. \quad (1)$$

We define the *scan frame* as the set containing the 2-D projection of each distance measurement for a particular timestamp k. The size of a scan frame is, at most, 32 points—4 sensors × 8 zones—as some distance measurements might be invalid and, therefore, not included in the scan frame. A scan frame is therefore obtained by applying (1) for each of the 32 points. The projection of the distance measurements is required for the scan-matching algorithm. However, the 32 points in a scan frame are still too sparse to compute accurate scan matching. We solve this problem by stacking 15 scan frames in a set that we call a *scan*. Specifically, once the nano-UAV decides to acquire a scan, it appends a new scan frame with a frequency of 7.5 Hz until it reaches the count of 15. To avoid synchronization problems, the four ToF sensors start data acquisition simultaneously. During the mission, the time misalignment between the sensors is about 9 ms, which would result in subcentimeter errors given the low speed of nano-UAVs. Furthermore, to mitigate these effects even more, we command the drone to stop while acquiring a scan.

The four ToF sensors have a cumulative FoV of 180°–45° per sensor. To maximize the scan coverage, the drone also rotates in place by 45° on the yaw axis while acquiring the scan, resulting in a 360° coverage. We empirically determined that the odometry errors introduced by spinning the drone during scan acquisition are negligible, leading to an increase of about 2 cm and 1.5° for the positioning and heading errors, respectively. In conclusion, we can define a scan as $\boldsymbol{S}_k = \{\boldsymbol{s}(\boldsymbol{x}_{\tilde{k}}, d_{\tilde{k}}^{ij}) \mid i \leq 4; j \leq 8; k \leq \tilde{k} < k + 15; i, j, \tilde{k} \in \mathbb{N}^*\}$ and each scan $\boldsymbol{S}_k$ has an associated pose $\boldsymbol{x}_k$, acquired right before the scan acquisition starts. The resulting size of a scan is, at most, 480 points. This setting was empirically selected based on the tradeoff between the need to have sufficient points for ICP-based scan-matching to produce accurate solutions and the memory footprint.

C. Iterative Closest Point

Scan matching is the process of aligning two scans, such that overlapping features line up correctly. The result of scan matching is the optimal rotation and translation that is applied to one scan, such that its features overlap with the other scan. Since each scan has an associated pose, the transformation between two scans is the same as the transformation between their associated poses. This is critical for correcting errors in odometry estimations that lead to the misalignment of poses that were acquired at different times or locations. We choose the ICP algorithm for scan matching and describe the theoretical study of ICP optimized for a practical implementation onboard nano-UAVs.

We define two scans $P = \{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_N\}$ and $Q = \{\boldsymbol{q}_N, \ldots, \boldsymbol{q}_M\}$ and omit the time index in the notation for simplicity. Scan matching can be formulated as a least squares problem, and finding the optimal transformation between P and Q is equivalent to solving the optimization problem in (2) [31]. However, this equation assumes known data associations, i.e., which point in scan Q corresponds to each point in scan P. Under this assumption, solving (2) leads to the optimal solution without requiring an initial guess. In the ideal case, when the scans are identical, rotating each scan point Q using $\boldsymbol{R}^*$ and then adding the translation factor $\boldsymbol{t}^*$ should result in a perfect overlap with scan P and the same holds for the poses $\boldsymbol{x}_Q$ and $\boldsymbol{x}_P$ that are associated with the two scans

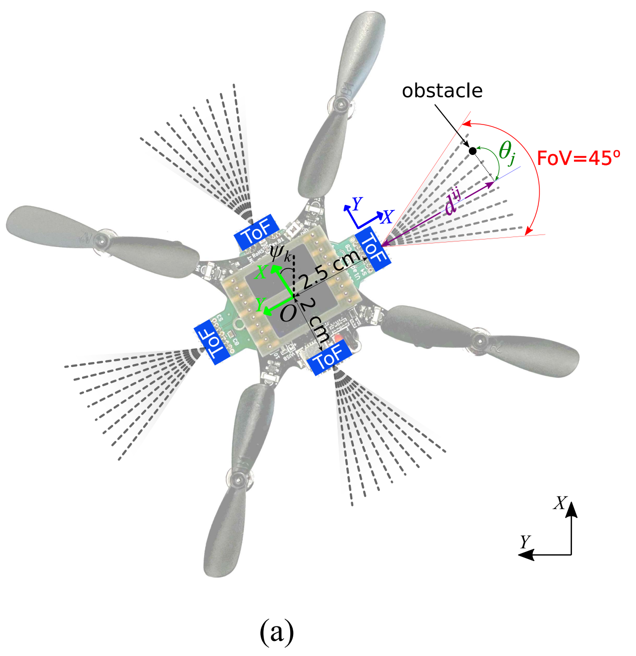
$$\boldsymbol{R}^*, \boldsymbol{t}^* = \arg\min_{\boldsymbol{R},\boldsymbol{t}} \sum \|\boldsymbol{q}_i - \left(\boldsymbol{R}\boldsymbol{p}_i + \boldsymbol{t}\right)\|^2. \quad (2)$$

However, no prior knowledge of correspondences is available in real-world applications, and a heuristic is required. A common method for establishing the correspondences for a given point in a scan is to find the closest point in the other scan in terms of Euclidean distance [31]. In practice, this step is performed using a double loop, and it, therefore, comes with a quadratic complexity. Furthermore, it typically accounts for more than 90% of the total ICP execution time. Once the correspondences are calculated, (2) determines the optimal transformation. The iterative process between calculating the correspondences and computing the optimal transformation, until the mean Euclidean distances between the correspondence pairs reach the minimum, is called ICP. Its accuracy is mainly impacted by how precisely the correspondences are determined.

D. Simultaneous Localization and Mapping

Producing an accurate map requires precise robot position estimation. In most indoor scenarios (including ours), the position and heading of the drone are computed by integrating velocity and angular velocity measurements, respectively, typically performed by the onboard EKF. However, these measurements are affected by sensor noise, and integrating noisy data over time results in drift. We employ SLAM to correct the errors in the trajectory of the robot caused by imprecise odometry measurements. However, this approach is also valid in the case of absolute range-based positioning systems [8].

We employ the algorithmic approach to graph-based SLAM introduced in [43] and implement an optimized version that allows the algorithm to run onboard resource-constrained nano-UAVs. Within this approach, the UAV's poses $\boldsymbol{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$ are modeled as graph nodes, and the odometry measurements $\boldsymbol{z}_{ij}$ as graph edges. An odometry measurement $\boldsymbol{z}_{ij} = (\Delta_x, \Delta_y, \Delta_\psi)$ is expressed in the coordinate frame of $\boldsymbol{x}_i$ and represents the relative transformation between poses $\boldsymbol{x}_i$ an $\boldsymbol{x}_j$. The work in [43] formulates the SLAM as a least-squares optimization problem that determines the optimal poses given



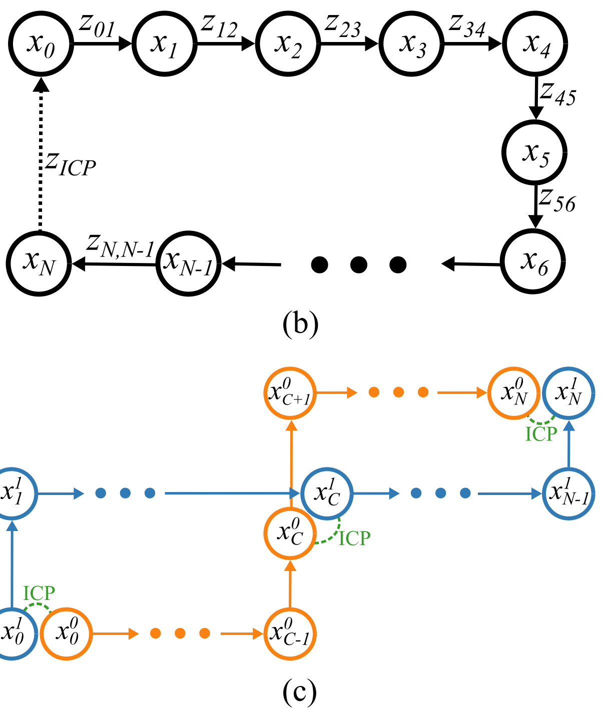

x_0 z_{01} x_1 z_{12} x_2 z_{23} x_3 z_{34} x_4 z_{45} x_5 z_{56} x_6 x_{N-1} $z_{N,N-1}$ x_N z_{ICP}
(b)
x^0_{C+1} x^0_N x^1_N ICP x^1_1 x^1_C x^1_{N-1} x^0_C ICP x^1_0 x^0_0 ICP x^0_{C-1}
(c)

Fig. 3. Illustrations of the four ToF sensors (a) and the pose graphs (b) and (c).

the pose $\boldsymbol{x}_0$ and a set of edges z_{ij}. The optimization problem is given by (3), where the number of terms in the sum is equal to the number of edges in the directed graph. Ω is the diagonal information matrix, which weighs the importance of each term in the sum

$$\boldsymbol{e}_{ij} = z_{ij} - \hat{z}_{ij}(\boldsymbol{x}_i, \boldsymbol{x}_j)$$
$$\boldsymbol{X}^* = \arg\min_{\boldsymbol{X}} \sum_{i,j} \boldsymbol{e}_{ij}^T \Omega \boldsymbol{e}_{ij}. \quad (3)$$

Fig. 3(b) shows a simple pose graph example, but representative of any real-world scenario. Since any node is connected to the previous one by an odometry measurement, at least one path exists between any two given nodes. The optimization solution given by [43] to the problem in (3) is based on the Gauss-Newton iterative method, and an initial guess is required for the poses, which we obtain from the nano-UAV's odometry measurements. Since the drone directly obtains the odometry measurements $z_{01}, z_{12}, \ldots z_{N-1,N}$ from the state estimator, it is straightforward to compute the initial guess of the poses $\boldsymbol{x}_1, \ldots \boldsymbol{x}_N$ by forward integration with respect to $\boldsymbol{x}_0$. This is the best guess that we have, which is quite close to the real values assuming that the odometry drift is in bounds—typically below 0.5 m. So far, solving the optimization problem in (3) for the measurements z_{ij} would lead to no change in the poses because the pose values after the forward integration are already in agreement with the measurements. Assuming that poses $\boldsymbol{x}_N$ and $\boldsymbol{x}_0$ are close enough in terms of Euclidean distance, one can use the ICP algorithm introduced in Section IV-C to generate a direct relation between the two poses—which we call a *virtual edge* or a *constraint*. The virtual edges take part in the optimization process just as the other edges, as additional terms to the sum in (3). However, since the scans are very accurate, so is the result of the ICP, and, therefore, the virtual edge measurements are more precise than the edges associated with odometry measurements. Consequently, the information matrix Ω associated with the virtual edges takes the value $20\boldsymbol{I}$, while the information matrix for the odometry edges is $\boldsymbol{I}$. These values are determined empirically. Fig. 2 shows the interaction between the EKF, ICP, and SLAM. The EKF provides the raw poses as input to the SLAM algorithm, which are used to derive the odometry edges. Furthermore, by acquiring scans in revisited places, the ICP block derives relative constraints between poses, which are also communicated to the SLAM block as virtual edges. The SLAM block optimizes the poses using the odometry and virtual edges and provides a new set of optimized poses. Optionally, the most recent optimized pose can be used to correct the accumulated odometry drift by overwriting the current estimate of the EKF.

So far, we have presented how we integrate the graph-based algorithm presented in [43] with ICP to optimize a single drone agent's trajectory (i.e., poses). In the following, we show how we apply SLAM when using multiple drones. Fig. 3(c) illustrates the pose graphs of two drones following two independent trajectories. For the sake of readability, we omit the notation for the edge measurements and use the superscript in the pose notation to indicate which drone the pose is associated with. Within the SLAM formulation introduced so far, the two trajectories would result in two disconnected graphs. However, knowing the locations where the trajectories intersect, the drones can use ICP to determine how closely located poses (e.g., $\boldsymbol{x}_0^0$ and $\boldsymbol{x}_0^1$) relate and therefore create a connection between the two graphs for each intersection point. This approach enables both loop closure and alignment of

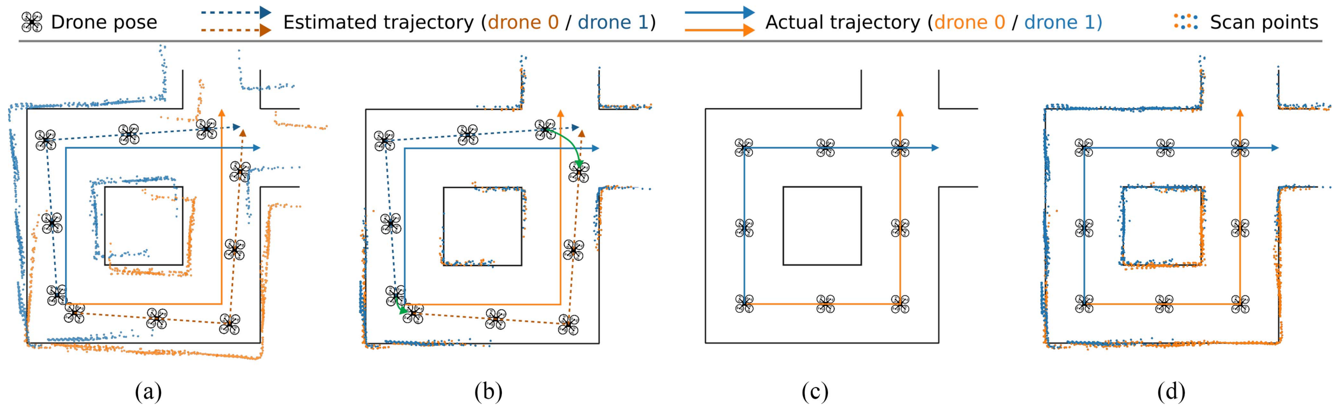

Drone pose
Estimated trajectory (drone 0 / drone 1)
Actual trajectory (drone 0 / drone 1)
Scan points
(a)
(b)
(c)
(d)

Fig. 4. Steps involved in the process of collecting and optimizing the poses and scans to generate a coherent map. This example uses two drones, and their estimated and actual trajectories are represented with dashed and solid lines, respectively. The walls of the environment are illustrated with black solid lines, while the ICP constraints are shown as green arrows. (a) Fetching all poses and scans. (b) Deriving constraints with ICP. (c) Optimizing the poses. (d) Correcting the scans.

the trajectories of multiple drones in a common coordinate system. While the scenario in Fig. 3(c) only refers to two drones, the approach scales for any number of drones in the swarm. Putting together the pose-graphs associated with multiple drones results in a larger graph that we call *the global graph*.

Fig. 4 illustrates a simplified example of how the information from multiple drones is merged to optimize each drone's trajectory and generate a collective map. The map is constructed by using the scans acquired by all drones and applying the SLAM optimization computed on one drone, which we call *the main drone*. Thus, in the first step the main drone—*drone 0* in this example—collects the scans and poses from *drone 1*, which together constitute the raw data, as shown in Fig. 4(a). We recall that every pose has an associated scan because whenever the drone logs a new pose, it also spins by 45° to obtain the scan. However, the scan is not directly acquired, but computed by projecting the ToF measurements in the world frame using (1)—this is performed in a distributed fashion, to reduce the computational load of the main drone.

Consequently, the data points in Fig. 4(a) are obtained by merging the five scans from drone 0 (in orange) with the five scans obtained from drone 1 (in blue). Due to odometry drift, the maps provided by each drone are not aligned. Next, the main drone derives virtual edges between the overlapping scans using ICP—represented with green arrows in Fig. 4(b). In this example, the overlapped scans belong to different drones, but as shown earlier in this section they could also belong to the same drone when a location is revisited. Fig. 4(b) shows only the overlapped scan pairs, acquired in the bottom left and top right corners. The base-poses and virtual edges represent the prerequisites for optimizing the pose-graph using SLAM, which leads to the optimized set of poses that are shown in Fig. 4(c). The same corrections applied to each pose can subsequently be applied to the corresponding scans, facilitating the creation of a coherent global map as shown in Fig. 4(d).

Since the size of the resulting global graph represents the sum of all subgraph sizes, there is no difference in computation and memory requirements between a swarm of m drones with N poses each and a single drone with $m \cdot N$ poses. Since the graph optimization is only executed on a single drone at a time, the main limitation of the system is the maximum number of poses that the SLAM algorithm can optimize within an acceptable time. However, this limitation could be further relaxed by dividing the global graph. For example, in the situation depicted in Fig. 3(c), optimizing the graph $\boldsymbol{x}_0^{1,0}, \boldsymbol{x}_1^{1,0}, \ldots \boldsymbol{x}_C^{1,0}$ and then using the optimized value of $\boldsymbol{x}_C^{1,0}$ as a constraint for optimizing $\boldsymbol{x}_C^{1,0}, \boldsymbol{x}_{C+1}^{1,0}, \ldots \boldsymbol{x}_N^{1,0}$ would result in very similar results to the case of optimizing the whole global graph at once.

E. Autonomous Exploration

Given that the environment is unknown, an autonomous exploration policy is needed to guide the nano-UAVs through the unfamiliar environment. The policy is deliberately designed to choose different paths for each drone, assuming sufficient distinct paths exist. To simplify the evaluation, we assume an environment consisting of perpendicular walls and equal-width corridors. For this purpose, we develop a simple exploration policy that can drive the drone along the x or y axis. While it can still be used in environments with nonperpendicular walls, this would lead to suboptimal "zig-zag" trajectories. However, minor adjustments could enable the exploration strategy to cope with arbitrary geometry environments if desired. The autonomous exploration policy is based on following walls or corridors and relies on the ToF sensor information to avoid obstacles. More in detail, it computes the minimum distance from the matrix in each direction and uses this information to make decisions. The motion commands are separated along two axes: 1) primary and 2) secondary. The primary axis is the direction the drone aims to explore. The secondary axis is perpendicular, and motion along this axis should be minimal.

The motion along the *secondary axis* is commanded so that the drone maintains a constant distance from the walls. A proportional velocity controller with a of gain $v_{\text{stab}} = 2\,\text{s}^{-1}$ determines the velocity set point based on the distance to the walls on either side, i.e., d_L and d_R. When the drone detects that it is located in a corridor (i.e., walls on both sides), it attempts to center itself by targeting a velocity $v_{\text{sec}} = d_L - 0.5 \cdot (d_L + d_R)) \cdot v_{\text{stab}}$. Otherwise, if there is a wall within 1 m of either side of the drone, it attempts to hold a target distance of $d_{\text{wall}} = 0.5\,\text{m}$ to the wall, by applying $v_{\text{sec}} = (d_{\text{wall}} - d_{L,R}) \cdot v_{\text{stab}}$. These values are chosen assuming that the width of corridors is approximately 1 m, but the parameters can be adapted for different environments. The wall-following is effective for scan matching since walls and corners typically provide feature-rich scans while avoiding frequent out-of-range measurements associated with large open spaces.

The *primary axis* describes the direction of exploration. Its control is based on waypoints. The navigation policy continuously seeks the next waypoint, which is always located about 1 m away from the previous one in the direction of motion. In each waypoint, the drone adds a new pose to its graph and acquires and stores a new scan. Equation (4) describes the proportional velocity control along the primary axis, where v_{exp} is the nominal exploration velocity, d_w is the remaining distance to the next waypoint and $d_{\text{slow}} = 0.75\,\text{m}$ determines the distance to a waypoint where the drone starts slowing down. Additionally, increases in velocity are limited to an acceleration of $a_{\text{exp}} = 0.5\,\text{m/s}^2$, to smooth the drone's motion. When the drone faces an obstacle in the direction of exploration before reaching the next waypoint, it immediately chooses a new perpendicular direction for exploration. The primary and secondary axes are switched, and exploration continues. If the only option is to go back the way the drone came (i.e., dead-end), it lands and finishes exploring. The choice of the new direction depends on the physical environment and a predefined preference that is set differently across individual drones in order to prevent multiple drones from exploring the same path

$$v_{\text{pri}}(d_w) = \begin{cases} \left(\frac{d_w}{d_{\text{slow}}} + 0.1\right) v_{\text{exp}}, & \text{if } d_w < d_{\text{slow}} \\ v_{\text{exp}}, & \text{otherwise.} \end{cases} \quad (4)$$

Given that we are interested in operating a swarm of drones, and in most situations, they start from the same location, it would be suboptimal if all drones had identical flight paths. At the same time, for testing purposes, individual drones' trajectories should be predictable and repeatable. In addition to alternating the steering priority (i.e., left or right) when facing an obstacle as mentioned above, we customize the exploration policy by choosing different initial headings for each drone. This changes how the primary axis is initially selected.

V. Swarm Coordination

This section discusses the methodology used to enable swarm mapping and interdrone radio communication. In particular, it introduces the communication protocol and the types of messages the drones exchange with each other. Furthermore, it explains how the drones coordinate within the swarm, the mapping mission, and how they exchange poses and scans to create a map. Lastly, a scalability study is provided.

A. Communication—Physical Layer

The physical layer is provided by the Crazyflie peer-to-peer (P2P) interface enabled through its onboard radio transceiver. The P2P application programming interface (API) provides basic packet-based communication primitives with best effort delivery. No routing is implemented on this layer, meaning all packets are broadcast without any form of acknowledgment. Further, there is neither media access control nor flow control over the radio link. Since the layer provides completely unreliable package delivery, all link, network, and transport layer functionality must be implemented. Given the limited capabilities of nano-UAVs, it is desirable to leverage the existing hardware already available onboard rather than adding more complex communication transceivers that would result in more weight and additional power consumption. Therefore, we rely on the available P2P communication and develop a robust and scalable communication protocol, which we present later in this section. Since we also need to capture data from the swarm in a base-station computer (for monitoring and evaluation purposes only), we configure an additional *bridge drone* that remains grounded, but relays all received packets to the computer over the built-in USB link.

B. Communication—Transport Layer

To forgo the complexity and resource requirements of, for example, a TCP/IP stack, we designed a lightweight transport layer protocol to provide reliable message-based networking with broadcast and unicast capabilities.

In order to simplify the protocol, we make some assumptions about the communication and impose some limitations. Most importantly, each transmitter may only attempt to transmit a new packet when all intended recipients have acknowledged the previous packet. This significantly reduces the complexity of the receiver and provides some rudimentary flow control. Further, we assume that all drones are within communication range of each other. This allows us to forego the need to implement a mesh networking scheme that would be out of the scope of this work. Since synchronization between drones in our method is event-driven and no long-lasting data streams between drones are necessary, we recognize that message-based networking is more appropriate than connection-oriented networking. In particular, most messages are short, and large messages are transmitted infrequently and are not broadcast. Message-based communication in this form reduces complexity, as less state needs to be maintained.

The basic principles of our proposed protocol are derived from ALOHAnet. However, to reduce the memory footprint of each layer, the protocol merges the link, network, and transport layers, which allows packets to be evaluated without copying individual sections. Furthermore, this enables us to merge all headers for these layers into a 16-bit wide structure and increase the utilization of physical layer packets.

Since the physical layer packets carry at most 60 bytes of user data, messages must be split into individual packets and reassembled at the receiver. Given the first limitation we imposed above, reassembly is trivial. We use the 16-bit wide packet header of the following structure to implement the protocol: 4-bit *source* and *destination* field, *acknowledge* bit, *end* bit, *tag* field, and *sequence number* field. The source and destination refer to the addresses of the transmitter and receiver drone. A destination field of address `0xF` indicates a broadcast, while the source field must not contain the broadcast address. The sequence number of each packet is an increasing unsigned integer that is used to deduplicate packets that have been retransmitted. When receiving a packet from another drone, the receiver responds with an empty packet with the acknowledge flag set. Once the sender receives the acknowledgment (ACK) packet from all targeted drones, it may send another message. Message boundaries are indicated by setting the end flag on the last packet of a message. The application layer uses the tag field to help identify the type of message received. No length field is included in the header since a lower layer header already provides this information.

C. Communication—Message Types

In the application layer, we distinguish between four different types of messages that are each identified using the tag field in the protocol header. First, there is the *pose update message* (PUM)—16 bytes: this message is used to synchronize the pose graph between drones. Each message contains a pose data structure and is broadcast to the entire swarm. The same message type is used both to register new poses and update existing poses. Since each node is identified by a globally unique identifier, these two scenarios can be distinguished by checking whether the pose identifiers already exist. The next message type is the *ToF scan request* (TSR)—4 bytes: this is a unicast message that is sent by the main drone to request the transfer of a ToF scan. The body of this message is a pose identifier associated with the scan that is being requested. Paired with the TSR, we introduce the *ToF scan response* (SR)—1146 bytes: this message is sent in response to a TSR message. The body contains the 2-D points in the requested scan. Lastly, we have the *control messages*—16 bytes: this class of messages contains control flow instructions, such as the takeoff or landing commands, sent by the base station through the bridge drone.

D. Mission Coordination

Even if identical in terms of hardware, the drones can be classified by their functionality. First, we have the bridge drone that we introduced above. Then, we have the flying drones in the swarm that explore the environment and acquire scans and poses. Among the flying drones in the swarm, one drone is elected as the leading main drone. Although all drones contribute to the mapping task, the main drone collects relevant scans, performs the ICP and SLAM computation to optimize the poses of the global graph, and propagates the results back to the swarm. Despite applying a centralized approach to execute the SLAM algorithm, the architecture does not, in fact, require the main drone to store all of the data acquired by the other drones. Memory-intensive data, such as the individual ToF scans, captured by other drones, can be loaded from the swarm on demand. This is important to ensure that this approach can scale with the size of the swarm, given the tight memory constraints of individual drones and bandwidth constraints of the entire swarm, and to increase the overall swarm robustness, avoiding single points of failure. The pose graph is the only data shared and synchronized between all drones in the swarm formation, which is lightweight and scales linearly with the total length of all flight paths in the swarm. Given that the pose graph is shared between all drones and all other data can be loaded on-demand, the architecture provides fault tolerance for the main drone. Should the main drone fail, any of the remaining drones are able to assume its role.

Before the mapping mission starts, all drones are placed in known positions. When the bridge drone broadcasts the take-off message, all drones start exploring the environment according to the exploration policy introduced in Section IV-E. Whenever a drone reaches a way-point (i.e., every 1 m), it adds a new pose in the graph and then broadcasts it to the swarm. Furthermore, a new scan is acquired and stored in the external flash memory. For every new pose, the main drone checks whether it is within proximity of another pose, indicating that there should be sufficient overlap between the corresponding ToF scans to perform scan matching. Should this be the case, the main drone will request the relevant scan data from the appropriate drone, execute ICP, and add a virtual edge to the pose graph. At the end of the mission, the main drone can execute the SLAM algorithm and broadcast corrected pose information back to the swarm. The pose graph is optimized at the end of the mission to simplify the evaluation, but due to the relatively low-execution time of SLAM (i.e., several seconds) with respect to the mission time (i.e., several minutes), the optimization can also happen multiple times during the mission. After SLAM is run, the optimized poses are used to correct the scans, and the map is obtained by merging all scans together.

E. Scalability

A performance and scalability analysis of the presented communication protocol is proposed. In particular, we consider the required bandwidth for broadcast communications with respect to the number of swarm agents, normalized with the interpose distance $d = 1$ m (also 1 m in our work). Hence, we define the required swarm bandwidth $B_s(N, d)$ as a function of the number of agents N and the pose-to-pose distance d as shown in (5). As introduced earlier in this section, *PUM*, *TSR*, and *SR* are 16 bytes, 4 bytes, and 1146 bytes, respectively. Furthermore, *ACK* is 2 bytes since an acknowledge requires sending a complete header without any payload. P_{sm} defines the probability two drones fly through the same location, i.e., probability to require the scan-matching. Normally, $\chi_{\text{upd}} \cdot (PUM_{\text{tot}} + f_{\text{map}} \cdot (ACK \cdot N))$ should also be added to $B_s(N, d)$ in (5) and represents the overhead for broadcasting the map from the main drone to the swarm. However, χ_{upd} defines the map update rate, which is broadcast to all drones after running SLAM, and it is zero in our work since we only run SLAM once at the end of the mission. f_{map} and f_{scan} are the packet fragmentation ratios, i.e., the minimum integer number of 60 byte packets to send a map and a scan, respectively.

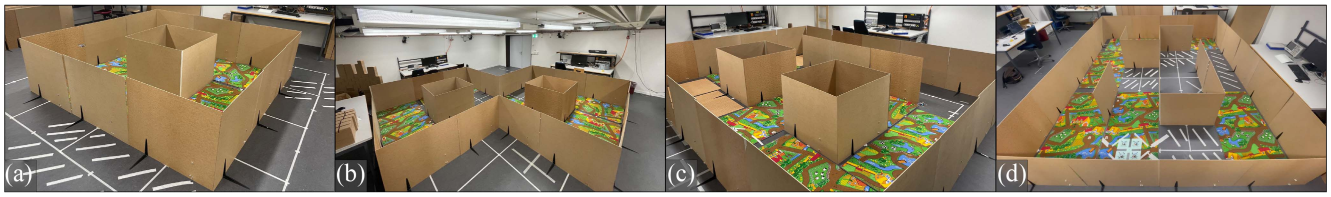


Fig. 5. Images taken in-field showing our evaluation environments. (a) Experimental setup for the experiment presented in Section VI-B. The mazes shown in (b)–(d) correspond to Experiments 1, 2, and 3 from Section VI-C.

Since the size of a scan is 1146 bytes, $f_{\text{scan}} = 20$, while f_{map} depends on the environment, but it is bounded by the size of the flash

$$B_s(N, d) = \underbrace{\frac{N}{d}(\text{PUM} + (\text{ACK} \cdot (N-1)))}_{\text{pose broadcasting}} + \underbrace{N \cdot P_{sm} \cdot (\text{TSR} + SR + f_{\text{scan}} \cdot \text{ACK})}_{\text{scan broadcasting}}. \quad (5)$$

Considering the practical experiments reported in Section VI with 2 or 4 nano-UAVs, the required bandwidth scanning a new pose every 1 m is 4.05 kbit/s and 8.24 kbit/s, respectively. Notably, the bandwidth requirement scales approximately linearly. Considering the Crazyflie P2P interface, the maximum measured bitrate in our swarm is 64.1 kbit/s, which would be able to support a swarm composed of 20 agents. However, if a 6 Mbit/s UWB communication would be used instead [8]—commonly used in nano-UAV platforms—the maximum swarm size could reach 500 with $d = 1$ m.

VI. Results

In this section, we provide a quantitative analysis of ICP and SLAM in terms of scan-matching accuracy, trajectory correction, and mapping accuracy. All experiments are performed in our testing arena, equipped with a Vicon Vero 2.2 motion capture system (mocap) for ground-truth measurements of the drones' poses. To assess the localization and mapping capabilities of our swarm, we build several mazes out of 100 cm × 80 cm chipboard panels. Fig. 5 shows in-field acquired images with the mazes used in our evaluation.

A. ICP Results

In the following, we evaluate the scan matching accuracy in terms of rotation error and translation error. In this scope, we place a drone in two nearby locations and acquire a scan in each. Then, we execute onboard ICP and compute the relative transformation between the poses associated with the two. Furthermore, with the aid of the mocap, we also compute the ground truth transformation since the mocap precisely measures the position and heading of the two poses. To calculate the error, we use a method similar to the one proposed by Pomerleau et al. [45]. We express the ground truth transformation and the one provided by ICP in homogenous coordinates, which we note $\boldsymbol{T}_{GT}$ and $\boldsymbol{T}_{ICP}$. Since homogenous coordination allows unifying rotation and translation in the same transformation, we calculate the ICP error using (6).

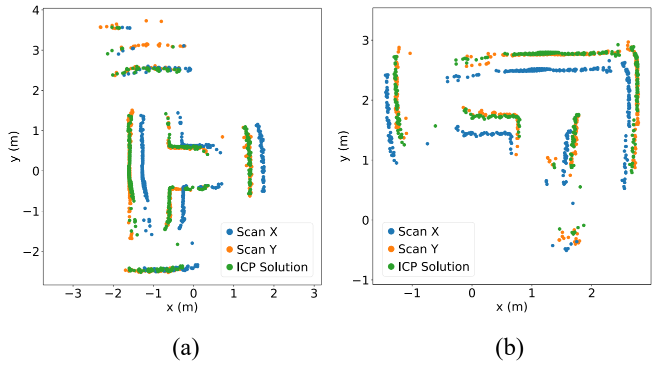


Fig. 6. Translation and rotation error for the onboard computed ICP solutions using empirical ToF scans. (a) $e_t = 0.7$ cm, $e_R = 0.98°$. (b) $e_t = 3.2$ cm, $e_R = 0.56°$.

Specifically, the translation error is calculated as $e_t = \|\Delta \boldsymbol{t}\|$, where $\Delta \boldsymbol{t}$ comprises the translation error on both X and Y axes. The rotation error is retrieved from the rotation matrix as $e_R = \cos^{-1}(\Delta \boldsymbol{R}_{00})$

$$\Delta \boldsymbol{T} = \begin{bmatrix} \Delta \boldsymbol{R} & \Delta \boldsymbol{t} \\ \boldsymbol{0}^\top & 1 \end{bmatrix} = \boldsymbol{T}_{ICP} \boldsymbol{T}_{GT}^{-1}. \quad (6)$$

Fig. 6 presents the in-field results for two scenarios, where the scans are acquired in a corridor intersection [i.e., Fig. 6(a)] and in a corner [i.e., Fig. 6(b)]. In both cases, the 2-D points of each scan are represented in blue and orange, respectively. Green represents the scan created by applying the ICP transformation to the scan X—which in both cases approximately matches scan Y. In the first experiment depicted in Fig. 6(a), the ICP algorithm achieves a highly accurate solution with a translation error of only 0.7 cm and a rotation error of 0.98°. In contrast, the translation error in the second experiment from Fig. 6(b) is slightly higher (i.e., 3.2 cm) while the rotation error is 0.56°. The increased translation error is most likely due to the poorer texture, such as the smaller number of corners in the scan. Furthermore, on the bottom part of Fig. 6(b), one could notice some artifacts. Since the scan generation is based on the drone's state estimate, errors in this estimate can lead to outliers in the scan. Acquiring a scan requires the drone to spin around its z-axis, and the state estimate that mainly relies on the onboard optical flow sensor is not very stable during spinning.

As first mentioned in Section IV, we use a scan size of 480 2-D points. In the following, we justify this choice by analyzing how the rotation and translation error achieved by ICP changes with the scan size. Since the scan is obtained by stacking multiple scan frames, we perform this analysis

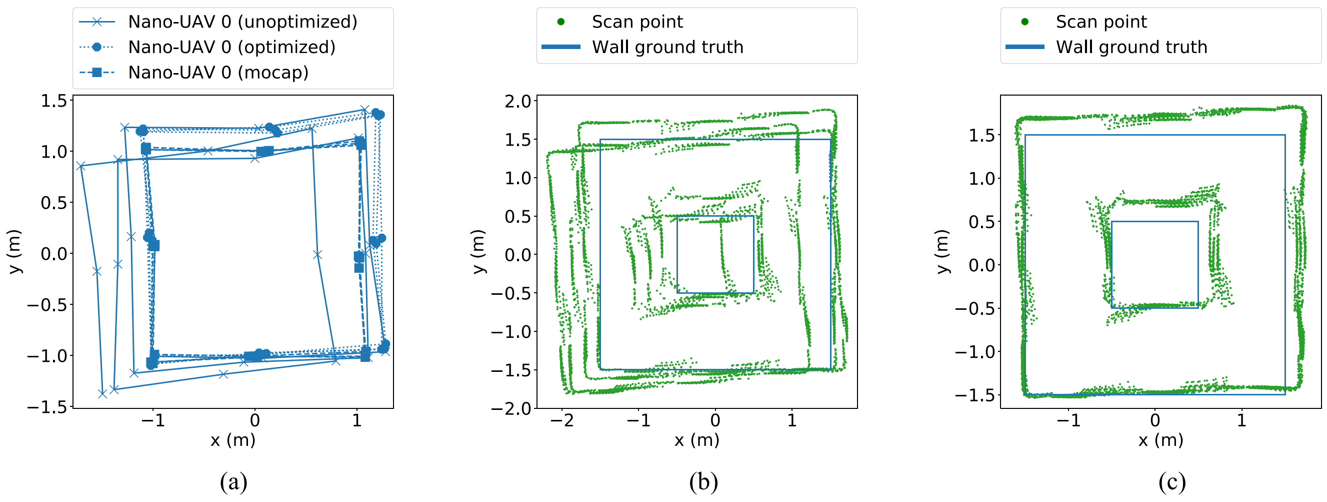


Fig. 7. Mapping a square maze with a single nano-UAV. (a) Trajectories and poses. (b) Map before optimization. (c) Map after optimization.

TABLE II
ACCURACY OF ICP AS A FUNCTION OF THE NUMBER OF SCAN FRAMES

	Frames	3	6	9	12	15
Figure 6a	e_t (cm)	33.8	8.6	1.5	0.6	0.7
	e_R (deg)	2.21	1.34	1.39	1.02	0.98
Figure 6b	e_t (cm)	6.3	4.4	3.6	3.2	3.2
	e_R (deg)	1.44	1.54	1.30	0.57	0.56
	Memory (B)	768	1536	2304	3072	3840

as a function of the number of scan frames. Therefore, we vary the frame count in the range of 3–15 with a step of 3 and evaluate the translation and rotation errors. Furthermore, this evaluation is performed for both experiments depicted in Fig. 6(a) and (b), respectively. The results are shown in Table II and prove that using more than 12 scan frames does not bring any improvement in terms of accuracy for both rotation and translation. Beyond this value, the change in rotation and translation is below 5%. However, we chose the value of 15 scan frames to have more robustness and account for possible corner cases. Table II also provides the maximum size of one individual scan, which scales linearly with the size of the scans. For a scan consisting of 15 scan frames, the scan size is 3840 B as every scan frame contains 32 2-D points (8 B per 2-D point).

B. SLAM Results

We recall that generating accurate maps requires accurate trajectory estimation, as the drone's trajectory represents the foundation for projecting the distance measurements in the world frame, as shown in Section IV-B. Therefore, we proceed by assessing the performance of the SLAM in correcting the trajectory errors and then generate the map and evaluate its accuracy. The experiments in this section only involve one drone agent. To evaluate the performance of the SLAM algorithm in correcting the trajectory, we check how close the estimated poses are to the ground truth poses provided by the mocap. Given a set of estimated poses $\boldsymbol{x}_1 \dots \boldsymbol{x}_n$, the corresponding set of ground truth poses is $\boldsymbol{x}_1^{GT} \dots \boldsymbol{x}_n^{GT}$. We mention that we use a reduced representation of the poses, each consisting only of the x and y coordinates. The pose estimation root-mean-squared-error (RMSE) of the optimized poses with respect to the ground truth is computed as

$$\mathrm{RMSE}_{\text{poses}} = \sqrt{\frac{\sum_{i=1}^{n} \|\boldsymbol{x}_i - \boldsymbol{x}_i^{GT}\|^2}{n}}. \quad (7)$$

To evaluate the pose correction performance, we proceed with an experiment where the drone is autonomously flying in a simple maze of square shape, with another square obstacle in the middle, just as shown in Fig. 7(b) or (c) in blue or in Fig. 5(a). The drone starts flying from the bottom left corner of the square and completes the loop three times. It flies at a constant height of 0.6 m and uses an exploration speed of $v_{\text{exp}} = 0.8$ m/ s, acquiring scans in each waypoint. Whenever it revisits a waypoint, it uses ICP scan matching to create SLAM constraints with respect to the scans acquired in the first trajectory loop. After flying the three loops, the poses are optimized by running SLAM.

Fig. 7(a) shows the three trajectories: 1) before SLAM (i.e., unoptimized); 2) after SLAM (i.e., optimized); and 3) the ground truth. The markers indicate the location of the poses while the lines interpolate between poses. The solid line indicates the unoptimized trajectory, i.e., as determined by the internal state estimator, while the dashed line represents the ground truth measured by the mocap. Furthermore, the dotted line shows the optimized flight path, which is closer to the ground truth. Specifically, the RMSE of the unoptimized and optimized poses is 34.6 cm and 19.8 cm, respectively.

We further evaluate the accuracy of the map, which is generated by putting together all the scans. First, we propose a metric that quantifies the mapping accuracy. We define walls as lines that span between two endpoints. If the point is above the wall, we define the distance of the point to the wall as the shortest distance to the line intersecting the endpoints. Otherwise, we define it as the distance to the closest endpoint. Given a set of walls W and a set of 2-D points in the map $\boldsymbol{p}_1 \dots \boldsymbol{p}_n$, we define the mapping RMSE as in (8). We apply this metric to the map generated with the unoptimized poses and show the results in Fig. 7(b), which leads to a mapping RMSE

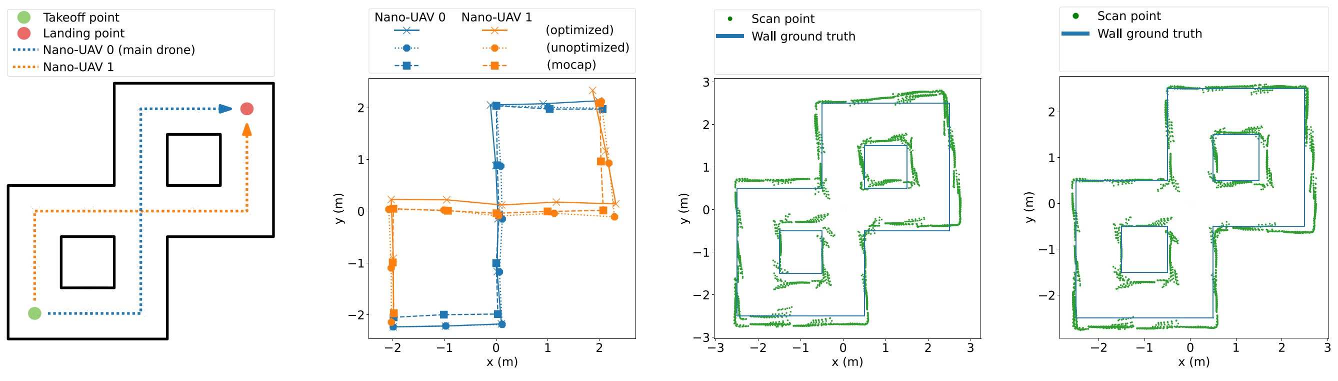


Fig. 8. First Experiment: Mapping experiment with a swarm of two nano-UAVs. (a) Maze layout. (b) Trajectories. (c) Unoptimized map. (d) Optimized map.

of 25.1 cm. Re-projecting the scans using the optimized poses results in a corrected map shown in Fig. 7(c). The mapping RMSE of the corrected map is 16 cm, proving that applying the correction based on ICP and SLAM improves the mapping accuracy by about 35%. Also, this time, we observe some artifacts, particularly at the corners of the walls, which are caused by inaccurate state estimation during the yaw rotation of a scan. Note, however, that the map appears scaled by some constant factor. This is due to the drone's state estimator, which consistently overestimates distances. This problem can be mitigated by performing odometry calibration, but it is out of scope and left for future work

$$\mathrm{RMSE}_{\mathrm{map}} = \sqrt{\frac{\sum_{i=1}^{n}\left(\min_{\forall w\in W}\operatorname{dist}(w,\boldsymbol{p}_i)\right)^2}{n}}. \quad (8)$$

The experiment was run with an exploration velocity $v_{\mathrm{exp}} = 0.8\,\mathrm{m/\,s}$. If we run the same experiment again but with $v_{\mathrm{exp}} = 0.2\,\mathrm{m/\,s}$, we obtain a pose estimation RMSE of 20.4 cm and 14.5 cm for the unoptimised and optimized poses, respectively. Note that even without any SLAM optimization, the pose estimation RMSE is smaller at lower velocity, indicating that the odometry drift is strongly related to the drone's velocity.

C. Mapping Results

So far, in this section, we evaluated the performance of ICP and SLAM, and we proved the benefit of applying SLAM for correcting the trajectory and generating a map with one drone. In the following, we focus on distributed mapping and evaluate the mapping capabilities when a swarm of drones is employed. To prove the generalization capabilities of our system, we performed three experiments, each consisting of mapping a different maze. We recall that the swarm-based experiments imply having one main drone, which collects the poses and scans from the other drones, runs the optimization, and then sends out the corrected poses. We proved in Section VI-B that the exploration velocity impacts the odometry accuracy. However, since a very low velocity of the drones would result in a large mapping time, we set $v_{\mathrm{exp}} = 0.4\,\mathrm{m/\,s}$ for the main drone and a larger velocity $v_{\mathrm{exp}} = 0.8\,\mathrm{m/\,s}$ for the other drones as a tradeoff between mapping accuracy and time.

TABLE III
POSE ESTIMATION AND MAPPING ACCURACY OF THE SECOND AND THIRD EXPERIMENT

Drones	Second experiment			
	Pose estimation RMSE		Mapping RMSE	
	no-SLAM	SLAM	no-SLAM	SLAM
2	45.5 cm	20.6 cm	30.9 cm	13.1 cm
4	28.6 cm	16.5 cm	22.5 cm	14.5 cm
	Third experiment			
2	34.2 cm	20.1 cm	25.3 cm	15.9 cm
4	21.8 cm	15.1 cm	17.9 cm	12.7 cm

1) First Experiment: The scenario of the first experiment is shown in Fig. 5(b) and described in Fig. 8(a). The green and red markings denote the takeoff and landing locations for all drones, which are connected by the expected flight paths of each drone (dashed lines). The crosses mark the expected locations of the poses acquired during autonomous exploration. The unoptimized and optimized trajectories are shown in Fig. 8(b) along with the ground truth poses for both drones—shown in blue and orange. Indeed, the pose estimation RMSE for the main drone is 15.9 cm, and almost 31 % higher for the other drone (i.e., 20.8 cm). The SLAM optimization can improve the situation significantly, reducing the overall pose estimation RMSE for both drones by about 25 %. Inspecting Fig. 8(b) once again, we can see that the optimized poses (orange dotted line) for drone 1 are now close to the ground truth (orange dashed line). Similarly, comparing the generated map from Fig. 8(c) and (d) shows a clear improvement, with the mapping RMSE of the point cloud improving by 19 % from 14.4 cm to 11.6 cm. The blue lines indicate the ground truth of where the walls are located.

2) Second Experiment: In the second experiment [Fig. 5(c)], we demonstrate the system's scalability as a swarm, evaluating not only the trajectory and mapping accuracy but also how the mapping time changes with the number of drones. For this scope, we map the same layout twice using swarms of two and four drones. The layout, as well as the intended flight paths, are visualized in Fig. 9(a) for two drones and in Fig. 9(b) for the scenario with four drones. The main drone is always *drone 0*. Table III lists the accuracy of the pose estimations and mapping with and

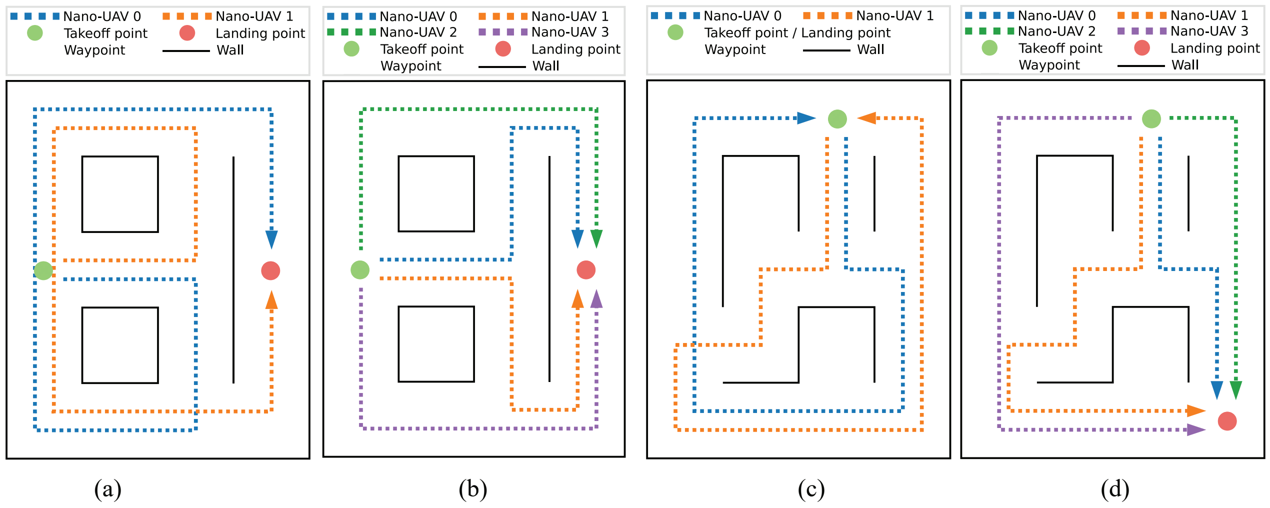


Fig. 9. Maze layouts for the second (a) and (b) and third experiment (c) and (d).

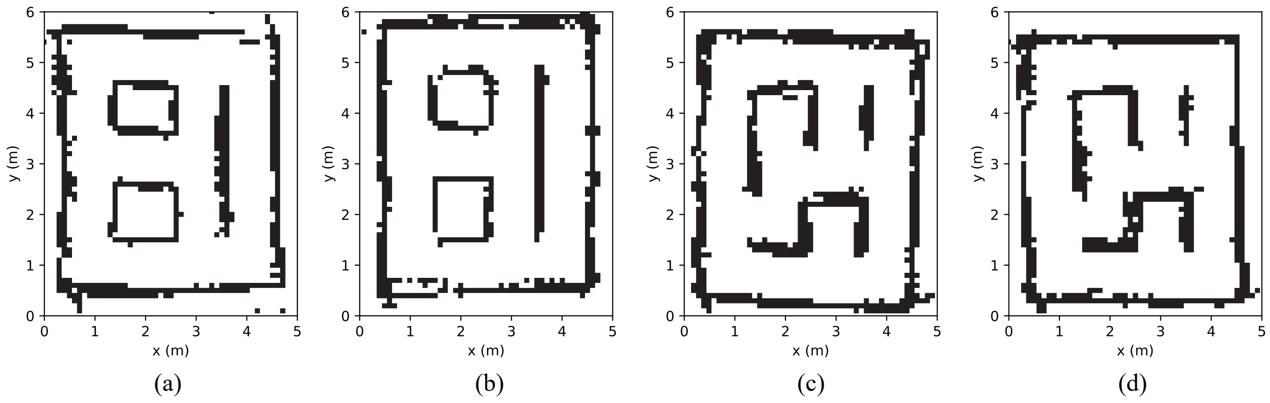


Fig. 10. Binary occupancy grid maps for the second (a) and (b) and third experiment (c) and (d).

without SLAM optimization for each swarm configuration. In general, the accuracy of pose estimations and mapping improves with more drones due to the shorter flight path per drone required to map the same area. On average, the SLAM optimization approximately halves the pose estimation error. The reduction in the mapping RMSE when SLAM is employed is 58% and 36% for the situation with two and four drones, respectively. While the RMSE of the corrected map is about the same for two and four drones, the mapping time is heavily reduced, from 5 min 28 s with two drones to 2 min 38 s with four drones. While keeping the same takeoff and landing location, we can cover the same area in less than half the time with four drones compared to two drones. This result motivates the need for a drone swarm and proves the scalability potential. In the first experiment, we provided the generated map in a dense representation (i.e., Fig. 8). Now we use the approach presented in [46] to convert the dense map to an occupancy grid map with a resolution of 10 cm, which we directly show in Fig. 10(a) and (b). We chose this representation as it is more meaningful for enabling future functionalities, such as path planning.

3) Third Experiment: In this experiment, the objective is the same as in the second one, but the maze layout differs, as shown in Fig. 5(d). The layout and the drones' trajectories are presented in Fig. 9(c) and (d) for two drones and four drones, respectively. For this experiment, the maze layout is slightly more complicated to better approximate the diverse shapes of indoor environments. In this layout, it is not possible to use the same takeoff and landing locations for both swarm configurations. Instead, the drones in the two-drone swarm return to their starting point, while the drones in the four-drone swarm finish their flight path in a different location. Table III presents the pose estimation and mapping RMSE. We observe an improvement in the pose estimation of 41% for the two-drone case and 31% for the four-drone case when using the SLAM optimization, while the mapping RMSE is reduced by 37% and 29%, respectively. Due to the particularly long flight paths of drones 1 and 3 in the four-drone swarm, this configuration only benefits from a 20 % mission time advantage in this layout, as the mission times are 3 min 7 s and 2 min 30 s for the cases with two drones and four drones, respectively. We further remark that the accuracy figures when using SLAM optimization are similar to those observed in the previous experiment, as seen in Table III. As for the previous experiment, we provide the resulting occupancy grid maps, shown in Fig. 10(c) and (d).

D. Performance

As its name suggests, ICP is an iterative algorithm whose execution time scales linearly with the chosen number of

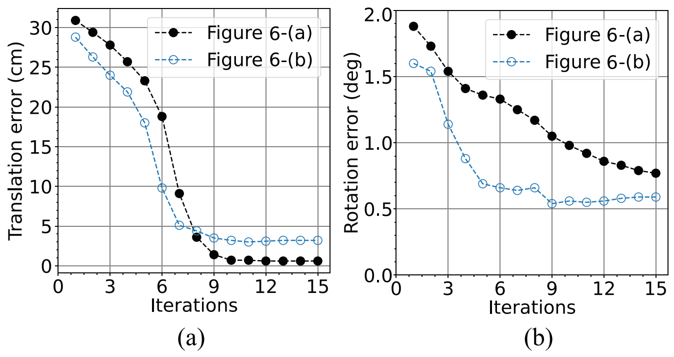


Fig. 11. ICP error as a function of the number of iterations. (a) Translation error. (b) Rotation error.

iterations and quadratically with the number of 2-D points in a scan—due to the correspondence calculation as explained in Section IV. Since we already motivated our choice for the scan size, in the following, we investigate how the accuracy of ICP varies with the number of iterations. Fig. 11 provides the rotation and translation errors for the experiments depicted in Fig. 6(a) and (b). More in detail, Fig. 11(a) shows how the translation error evolves with the number of iterations. We notice that the translation error tends to converge after about seven iterations and nine iterations for the experiments in Fig. 6(a) and (b), respectively. Beyond these values, the two curves tend to flatten. Furthermore, Fig. 11(b) shows the rotation error. In this case, we observe a slower convergence trend, but for both experiments, the rotation error decreases below 1° for more than nine iterations. Since we consider it acceptable to have rotation errors below 1°, we fix the number of iterations to 10, which is the value used in all our experiments. While it is true that a dynamic number of iterations is possible, we prefer to use a fixed value to ensure deterministic execution time.

Next, we evaluate the computation time required to run ICP and SLAM. Since the execution speed is critical for real-time systems, we analyze how different settings for these algorithms impact the execution time. For the ICP algorithm, the most critical parameter is the maximum number of 2-D points in a scan, which yields to a quadratic increase in the execution time. Fig. 12(a) shows how the execution time varies with the size of the scan, using a step of 32—the number of 2-D points in a scan frame. As expected, the empirical results also show a purely quadratic dependency. This is expected because the most dominant computation in ICP is finding the correspondences—for each point in the first scan, find the closest point in the second scan—which is implemented as a double for loop. Matching the empirical measurements presented by Fig. 12(a) with the quadratic function $0.002 \cdot x^2 + 0.0135 \cdot x - 3.38$ results in a matching over 99%. The chosen scan size of 480 points is just below 0.5 s and meets the real-time requirements of the embedded platform. Since every ICP iteration implies the same computations, obtaining the execution time of one iteration can be performed by dividing the times reported in Fig. 12(a) by 10. Moreover, running ICP requires having two scans in the memory, and thus, a scan size of 480 results in a memory consumption of 7680 B, which is

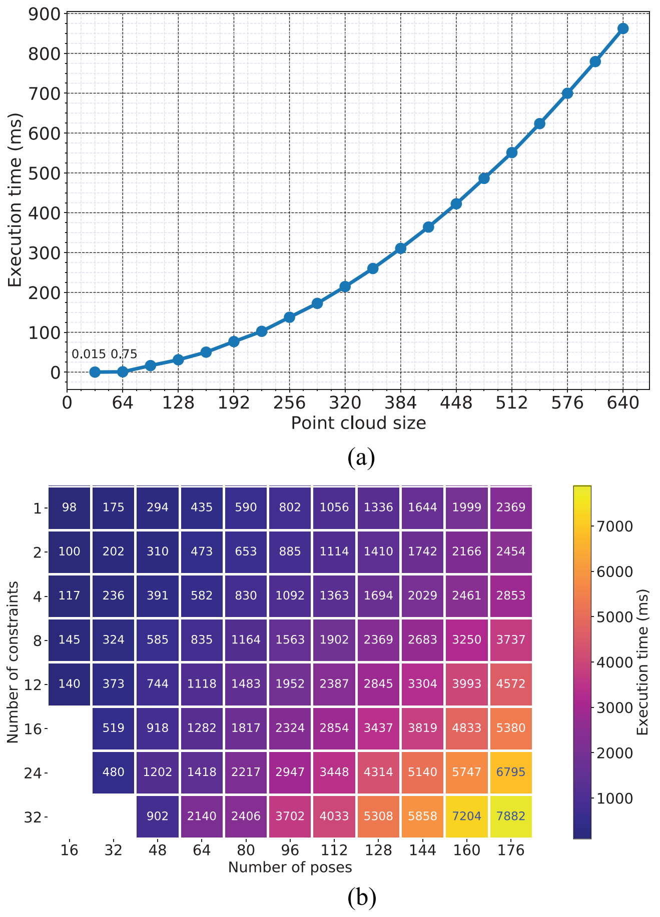


Fig. 12. Execution times of SLAM and ICP for different configurations. (a) ICP. (b) SLAM.

still much smaller than the total amount of 50 kB available random access memory (RAM).

Next, we provide an evaluation of the execution time of SLAM as a function of both the number of poses and constraints in Fig. 12(b). We vary the number of constraints in the range 2^0–2^5, which is representative of the number of loop closures typically required in a real-world scenario. The number of poses is swept in the range of 16–176, with a step of 32. 176 is the largest number of poses that can accommodate 32 constraints without overflowing the available RAM of 50 kB. The minimum execution time is 98 ms for 16 poses with one constraint, and the maximum is 7882 ms for a combination of 176 poses and 32 constraints. While quite large, even the maximum execution time of almost 8 s is still feasible for a real-world mission. The only drawback is that the drones have to hover and wait while SLAM is running, which would increase the mission time by 8 s every time SLAM is executed during the mission.

E. Discussion

We present the limitations of our system as well as possible improvements. The main goal of our work was to demonstrate that the novel multizone ToF sensor can enable new applications for nano-drones, which can generate accurate maps fully onboard due to our optimized implementations of ICP and

SLAM. A notable limitation is the maximum mappable area that our system supports. As mentioned earlier in this work, this is mainly limited by the maximum amount of poses that the SLAM algorithm can optimize at a time, as well as the interpose distance. With the available onboard RAM of 50 kB, our system can optimize about 180 poses, which for a 1 m interpose distance corresponds to an area of about 180 m^2. However, the 1 m distance for adding a new pose was selected because our testing area is small. Since the range of the ToF multizone sensor is at least 2 m with less than 5% invalid pixels [17], the interpose distance could also be increased to 2 m which would result in a maximum mappable area four times as large, so about 720 m^2. A more effective way to cope with larger pose graphs is using more onboard RAM memory to optimize large graphs fast enough to meet the real-time requirements. Emerging low-power embedded MCUs, such as multicore RISC-V SoCs, are the perfect candidates for this task as they have a power consumption within hundreds of mW, and their parallel capabilities could heavily reduce the execution time of ICP and SLAM, such as in [47]. In any case, so far, ICP and SLAM have been executed on workstation-level with general-purpose CPUs, such as the Intel i7-7700HQ, featuring a thermal design power (TDP) of 45 W [30], [31], or onboard with a 7.5 W computer on module. In Table I, Huang et al. [15] and Zhou et al. [19] reported a mapping accuracy aligned with this work, but with a power consumption 31× higher, 7.5 W versus 240 mW and a flying UAV 52× heavier. Compared with [30] and [31], our results achieve a 188× lower power consumption. Regarding the onboard execution time, our optimized ICP is aligned with the SoA of 26-83 ms [30], [31] with a point cloud size up to 192, see Fig. 12, while still respecting the real-time performance requirements for nano-UAV application with an increased number of scans.

Another limitation of our system is the resolution of the ToF sensor. While more capable than the other available solutions of its weight and size class, the multizone ToF sensor still has a poorer range and resolution than the larger and more power-hungry LiDARs. As shown in Fig. 3(a), each ToF zone represents a triangle in 2-D (or a cone in 3-D), whose base increases with the distance magnitude and results in higher uncertainty for the obstacle location. Therefore, flying the drone closer (i.e., within 2 m) to the walls or objects is ideal for accuracy-critical processes. Due to the larger resolution of LiDARs, the zones are narrower, which translates into a higher accuracy that remains in bounds even at large distances. Moreover, since LiDARs have a higher measurement range than the 4 m of the multizone ToF sensor, they require drones to fly less when mapping larger rooms.

In our work, we perform 2-D pose correction and mapping. However, since the downward-facing distance sensor directly measures the drone's height, our system is also relevant for 3-D mapping applications with uniform ground. Within this scenario, the drone could change its height during the mission and build an augmented pose structure based on the height provided by the downward-pointing distance sensor and the x and y coordinates provided by SLAM. In this way, projecting the whole 8×8 distance matrices from the four ToF sensors in the world frame would result in a 3-D point cloud. This mechanism would require minimal modifications to our system as the scan-matching algorithm would still operate in 2-D. However, for operating in environments with uneven grounds, 3-D scan matching is necessary to derive 3-D relative constraints between poses. While the structure of the mapping pipeline would be very similar, more capable MCUs are necessary to enable the heavier real-time computation introduced by an additional degree of freedom.

VII. Conclusion

This article presented a complete system to enable mapping on a swarm of IoRT nano-UAVs based on a novel ISCC system and four low-power multizone ToF sensors. The main contributions include a novel lightweight sensor module that provides a 360° depth map with an accuracy comparable with SoA LiDARs. Moreover, ICP and SLAM are optimized to run on resource-constrained MCUs with a total available memory of only 50 kB, supporting and combining sparse measurements from up to 20 swarm agents. Using a different wireless physical link, e.g., the UWB protocol commonly used for indoor localization, the swarm can potentially be extended to hundreds of nano-UAVs [48]. We also demonstrated, with three field experiments, that a collaborative group of robots can accurately map an environment and decrease the mission latency linearly w.r.t. the number of employed swarm agents. The swarm redundancy, and its improved robustness, are assured by a system-level strategy specifically designed for this work. Indeed, all the collected poses are broadcast to the whole swarm; thus, the *main drone* can be selected at will or easily replaced by another nano-UAV without losing any information—the swarm data is distributed across all its agents. Despite its extremely lightweight setup (38.4 g), the system proposed in this article features a mapping accuracy comparable with the SoA contributions designed for MAVs and standard-size UAVs. We showed how our swarm-based solution reduces the pose estimation and mapping errors by about half while flying with a speed of 0.8 m/ s. The system proposed in this article paves the way for more autonomous micro-robots characterized by ultraconstrained hardware, *de-facto* enabling a system technology not present in the nano-UAV field so far. Indeed, with a complete environmental map, advanced navigation solutions can be inferred to enhance flight autonomy with advanced path planning and mission strategies.

Supplementary Material

Demo video at: https://youtu.be/c9hajp_43aw. Code available at: https://github.com/ETH-PBL/Nano_Swarm_Mapping.

References

[1] M. Zhang and X. Li, "Drone-enabled Internet-of-Things relay for environmental monitoring in remote areas without public networks," *IEEE Internet Things J.*, vol. 7, no. 8, pp. 7648–7662, Aug. 2020.

[2] C. Wang, J. Wang, J. Wang, and X. Zhang, "Deep-reinforcement-learning-based autonomous UAV navigation with sparse rewards," *IEEE Internet Things J.*, vol. 7, no. 7, pp. 6180–6190, Jul. 2020.

[3] H. Zhou, Z. Hu, S. Liu, and S. Khan, "Efficient 2D graph SLAM for sparse sensing," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, 2022, pp. 6404–6411.

[4] S. Khan, S. Ullah, H. U. Khan, and I. U. Rehman, "Digital twins-based Internet of Robotic Things for remote health monitoring of COVID-19 patients," *IEEE Internet Things J.*, vol. 10, no. 18, pp. 16087–16098, Sep. 2023.

[5] D. Liu, W. Bao, X. Zhu, B. Fei, T. Men, and Z. Xiao, "Cooperative path optimization for multiple UAVs surveillance in uncertain environment," *IEEE Internet Things J.*, vol. 9, no. 13, pp. 10676–10692, Jul. 2021.

[6] B. P. Duisterhof, S. Li, J. Burgués, V. J. Reddi, and G. C. de Croon, "Sniffy bug: A fully autonomous swarm of gas-seeking nano quadcopters in cluttered environments," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, 2021, pp. 9099–9106.

[7] B. P. Duisterhof et al., "Tiny robot learning (TinyRL) for source seeking on a nano quadcopter," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2021, pp. 7242–7248. [Online]. Available: https://doi.org/10.1109/ICRA48506.2021.9561590

[8] T. Polonelli, M. Magno, V. Niculescu, L. Benini, and D. Boyle, "An open platform for efficient drone-to-sensor wireless ranging and data harvesting," *Sustain. Comput. Inf. Syst.*, vol. 35, Sep. 2022, Art. no. 100734.

[9] M. W. Nawaz, O. Popoola, M. A. Imran, and Q. H. Abbasi, "K-DUMBs IoRT: Knowledge driven unified model block sharing in the Internet of Robotic Things," in *Proc. IEEE 97th Veh. Technol. Conf. (VTC-Spring)*, 2023, pp. 1–6.

[10] A. Loquercio, E. Kaufmann, R. Ranftl, M. Müller, V. Koltun, and D. Scaramuzza, "Learning high-speed flight in the wild," *Sci. Robot.*, vol. 6, no. 59, pp. 1–23, 2021.

[11] N. Dilshad, A. Ullah, J. Kim, and J. Seo, "LocateUAV: Unmanned aerial vehicle location estimation via contextual analysis in an IoT environment," *IEEE Internet Things J.*, vol. 10, no. 5, pp. 4021–4033, Mar. 2023.

[12] F. Bai, T. Vidal-Calleja, and G. Grisetti, "Sparse pose graph optimization in cycle space," *IEEE Trans. Robot.*, vol. 37, no. 5, pp. 1381–1400, Oct. 2021.

[13] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. Montiel, and J. D. Tardós, "ORB-SLAM3: An accurate open-source library for visual, visual–inertial, and multimap SLAM," *IEEE Trans. Robot.*, vol. 37, no. 6, pp. 1874–1890, Dec. 2021.

[14] Y. Chang et al., "LAMP 2.0: A robust multi-robot SLAM system for operation in challenging large-scale underground environments," *IEEE Robot. Autom. Lett.*, vol. 7, no. 4, pp. 9175–9182, Oct. 2022.

[15] P. Huang, L. Zeng, X. Chen, K. Luo, Z. Zhou, and S. Yu, "Edge robotics: Edge-computing-accelerated multirobot simultaneous localization and mapping," *IEEE Internet Things J.*, vol. 9, no. 15, pp. 14087–14102, Aug. 2022.

[16] 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.

[17] V. Niculescu, H. Müller, I. Ostovar, T. Polonelli, M. Magno, and L. Benini, "Towards a multi-pixel time-of-flight indoor navigation system for Nano-drone applications," in *Proc. IEEE Int. Instrum. Meas. Technol. Conf. (I2MTC)*, 2022, pp. 1–6.

[18] S. Karam, F. Nex, B. T. Chidura, and N. Kerle, "Microdrone-based indoor mapping with graph SLAM," *Drones*, vol. 6, no. 11, p. 352, 2022.

[19] B. Zhou, H. Xu, and S. Shen, "Racer: Rapid collaborative exploration with a decentralized multi-UAV system," *IEEE Trans. Robot.*, vol. 39, no. 3, pp. 1816–1835, Jun. 2023.

[20] E. Jeong, S. Kang, D. Lee, and P. Kim, "Parsing indoor manhattan scenes using four-point LiDAR on a micro UAV," in *Proc. 22nd Int. Conf. Control, Autom. Syst. (ICCAS)*, 2022, pp. 708–713.

[21] G. A. Kumar, A. K. Patil, R. Patil, S. S. Park, and Y. H. Chai, "A LiDAR and IMU integrated indoor navigation system for UAVs and its application in real-time pipeline classification," *Sensors*, vol. 17, no. 6, p. 1268, 2017. [Online]. Available: https://www.mdpi.com/1424-8220/17/6/1268

[22] F. Gao, W. Wu, W. Gao, and S. Shen, "Flying on point clouds: Online trajectory generation and autonomous navigation for quadrotors in cluttered environments," *J. Field Robot.*, vol. 36, no. 4, pp. 710–733, 2019. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21842

[23] Z. Fang et al., "Robust autonomous flight in constrained and visually degraded shipboard environments," *J. Field Robot.*, vol. 34, no. 1, pp. 25–52, 2017. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21670

[24] F. Causa et al., "UAV-based LiDAR mapping with Galileo-GPS PPP processing and cooperative navigation," in *Proc. Int. Conf. Unmanned Aircr. Syst. (ICUAS)*, 2022, pp. 938–947.

[25] H. Shen, Q. Zong, B. Tian, X. Zhang, and H. Lu, "PGO-LIOM: Tightly-coupled LiDAR-inertial odometry and mapping via parallel and gradient-free optimization," *IEEE Trans. Ind. Electron.*, vol. 70, no. 11, pp. 11453–11463, Nov. 2023.

[26] H. Müller, V. Niculescu, T. Polonelli, M. Magno, and L. Benini, "Robust and efficient depth-based obstacle avoidance for autonomous miniaturized UAVs," *IEEE Trans. Robot.*, vol. 39, no. 6, pp. 4935–4951, Dec. 2023.

[27] S. Karam et al., "Micro and macro Quadcopter drones for indoor mapping to support disaster management," *ISPRS Ann. Photogrammetry, Remote Sens. Spatial Inf. Sci.*, vol. 1, pp. 203–210, May 2022.

[28] V. Niculescu, T. Polonelli, M. Magno, and L. Benini, "NanoSLAM: Enabling fully onboard SLAM for tiny robots," *IEEE Internet Things J.*, early access, Dec. 5, 2023, doi: 10.1109/JIOT.2023.3339254.

[29] Y. Pan, P. Xiao, Y. He, Z. Shao, and Z. Li, "MULLS: Versatile LiDAR SLAM via multi-metric linear least square," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2021, pp. 11633–11640.

[30] P. Dellenbach, J.-E. Deschaud, B. Jacquet, and F. Goulette, "CT-ICP: Real-time elastic LiDAR odometry with loop closure," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, 2022, pp. 5580–5586.

[31] I. Vizzo, T. Guadagnino, B. Mersch, L. Wiesmann, J. Behley, and C. Stachniss, "KISS-ICP: In defense of point-to-point ICP—Simple, accurate, and robust registration if done the right way," *IEEE Robot. Autom. Lett.*, vol. 8, no. 2, pp. 1029–1036, Feb. 2023.

[32] H. Matsuki, R. Scona, J. Czarnowski, and A. J. Davison, "Codemapping: Real-time dense mapping for sparse slam using compact scene representations," *IEEE Robot. Autom. Lett.*, vol. 6, no. 4, pp. 7105–7112, Oct. 2021.

[33] G. Sonugür, "A review of quadrotor UAV: Control and SLAM methodologies ranging from conventional to innovative approaches," *Robot. Auton. Syst.*, vol. 161, Mar. 2023, Art. no. 104342.

[34] K. Beevers and W. Huang, "SLAM with sparse sensing," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2006, pp. 2285–2290.

[35] A. Doucet, N. de Freitas, K. Murphy, and S. Russell, "Rao-Blackwellised particle filtering for dynamic Bayesian networks," in *Proc. 16th Conf. Uncertainty Artif. Intell.*, 2000, pp. 176–183.

[36] D. Wilbers, C. Merfels, and C. Stachniss, "A comparison of particle filter and graph-based optimization for localization with landmarks in automated vehicles," in *Proc. 3rd IEEE Int. Conf. Robot. Comput. (IRC)*, 2019, pp. 220–225.

[37] K. Chengbo, Y. Huibin, Y. Juan, and L. Wenchao, "Research on wheeled mobile robot positioning based on EKF multi-sensor data fusion," in *Proc. 3rd Int. Conf. Electron. Inf. Technol. Comput. Eng. (EITCE)*, 2019, pp. 457–460.

[38] Z. Zhu, Y. Kaizu, K. Furuhashi, and K. Imou, "Visual-inertial RGB-D SLAM with encoders for a differential wheeled robot," *IEEE Sensors J.*, vol. 22, no. 6, pp. 5360–5371, Mar. 2022.

[39] T. Du, S. Shi, Y. Zeng, J. Yang, and L. Guo, "An integrated INS/LiDAR odometry/polarized camera pose estimation via factor graph optimization for sparse environment," *IEEE Trans. Instrum. Meas.*, vol. 71, pp. 1–11, 2022.

[40] K. R. Beevers and W. H. Huang, "An embedded implementation of SLAM with sparse sensing," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2008, pp. 1–6.

[41] M. Alpen, C. Willrodt, K. Frick, and J. Horn, "On-board SLAM for indoor UAV using a laser range finder," in *Proc. Unmanned Syst. Technol. XII*, 2010, Art. no. 769213. [Online]. Available: https://doi.org/10.1117/12.849984

[42] M. Alpen, K. Frick, and J. Horn, "An autonomous indoor UAV with a real-time on-board orthogonal SLAM," *IFAC Proc. Vol.*, vol. 46, no. 10, pp. 268–273, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474667015349430

[43] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based SLAM," *IEEE Intell. Transp. Syst. Mag.*, vol. 2, no. 4, pp. 31–43, 2010. [Online]. Available: https://ieeexplore.ieee.org/document/5681215

[44] H. Jinqiang, W. Husheng, Z. Renjun, M. Rafik, and Z. Xuanwu, "Self-organized search-attack mission planning for UAV swarm based on wolf pack hunting behavior," *J. Syst. Eng. Electron.*, vol. 32, no. 6, pp. 1463–1476, 2021.

[45] F. Pomerleau, F. Colas, R. Siegwart, and S. Magnenat, "Comparing ICP variants on real-world data sets open-source library and experimental protocol," *Auton. Robots*, vol. 34, no. 3, pp. 133–148, 2013. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01143458

[46] S. Thrun, “Probabilistic robotics,” *Commun. ACM*, vol. 45, no. 3, pp. 52–57, 2002.

[47] D. Brunelli, T. Polonelli, and L. Benini, “Ultra-low energy pest detection for smart agriculture,” in *Proc. IEEE SENSORS*, 2020, pp. 1–4.

[48] N. Koul, N. Kumar, A. Sayeed, C. Verma, and M. S. Raboca, “Data exchange techniques for Internet of Robotic Things: Recent developments,” *IEEE Access*, vol. 10, pp. 102087–102106, 2022.

Carl Friess received the master’s degree in computer science from ETH Zürich, Zürich, Switzerland, in 2023.

He served as the Avionics Team Lead and later as a President of the Swissloop Student Team, contributing to the team’s successes in the SpaceX Hyperloop Pod Competition, Hawthorne, CA, USA. Later, during his time as an Embedded Software Engineer, The Boring Company, Los Angeles, CA, USA, he had the privilege of serving as a Judge in the 2020 Edition of the Competition. He is currently focusing on control and cloud-based orchestration of mobile energy storage systems, seeking to contribute to distributed renewable energy solutions.

Vlad Niculescu received the master’s degree in robotics, systems, and control from ETH Zürich, Zürich, Switzerland, in 2019, where he is currently pursuing the Ph.D. degree in electrical engineering with the Integrated Systems Laboratory.

His research is currently focused on developing localization, mapping, and autonomous navigation algorithms that target ultra-low-power platforms which can operate onboard nano-drones. During the bachelor and master period, he competed in more than ten international student competitions, and he was the Electrical Lead of the Student Project Swissloop, which won Second Place and the Innovation Award in the SpaceX Hyperloop Pod Competition 2019.

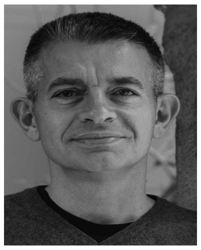
Tommaso Polonelli (Member, IEEE) received the M.Sc. and Ph.D. degrees in electronic engineering from the University of Bologna, Bologna, Italy, in 2017 and 2020, respectively.

He is currently a Postdoctoral Researcher with ETH Zürich, Zürich, Switzerland. His research work focuses on wireless sensor networks, Internet of Things, autonomous unmanned vehicles, power management techniques, structural health monitoring, and the design of ultra-low-power battery-supplied devices with onboard intelligence. He has collaborated with several universities and research centers, such as the University College Cork, Cork, Ireland, and the Imperial College London, London, U.K. He has authored over 40 papers in international journals and conferences.

Michele Magno (Senior Member, IEEE) received the master’s and Ph.D. degrees in electronic engineering from the University of Bologna, Bologna, Italy, in 2004 and 2010, respectively.

He is currently a Senior Scientist with the Department of Information Technology and Electrical Engineering (D-ITET), ETH Zürich, Zürich, Switzerland, where he is leading the D-ITET Center for Project-Based Learning since 2020. He is with ETH since 2013 and has become a Visiting Lecturer or a Professor with several universities, namely, the University of Nice Sophia, Nice, France; Enssat, Lannion, France; Univerisity of Bologna, Bologna, Italy; and Mid University Sweden, Östersund, Sweden, where currently is a Full Visiting Professor with the Electrical Engineering Department. He has authored more than 220 papers in international journals and conferences. His current research interests include smart sensing, low-power machine learning, wireless sensor networks, wearable devices, energy harvesting, low-power management techniques, and extension of the lifetime of batteries-operating devices.

Dr. Magno were awarded Best Papers Awards at IEEE Conferences for some of his publications. He also received awards for industrial projects or patents. He is an ACM Member.



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

He holds the Chair of Digital Circuits and Systems, 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, smart sensing micro-systems, and machine learning hardware.

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