A Plug-and-Play TinyML-based Vision System for Drone Automatic Landing

Luca Santoro, Andrea Albanese, Marco Canova, Matteo Rossa, Daniele Fontanelli and Davide Brunelli
Department of Industrial Engineering, *University of Trento*, Trento, Italy
name.surname@unitn.it

***Abstract*—Automatic landing is a feature that allows aerial robotic platforms to safely and accurately land without human intervention. This paper presents a plug-and-play tiny machine learning vision-based system for automatic landing compatible with the Pixhawk flight controller series. The proposed system is implemented on a low-power microcontroller, specifically OpenMV Cam H7 Plus, demonstrating that a constrained resources board can be used as a companion computer to enable autonomous functions for UAVs. The experiments confirm the proposed system's effectiveness, capable of correctly identifying a landing pad and consequently controlling the UAV to align it over the pad center before landing. The system overhead is only 2% of the UAV's total energy budget, with an accuracy of 93.5% and precision of 94.0%.**

***Index Terms*—Ultra-wideband, device-free localization, sensorless sensing, multipath-assisted localization, passive localization.**

I. Introduction

Autonomous Unmanned Aerial Vehicles (UAV) are gaining popularity across different industries owing to their capacity to execute tasks more efficiently and safely compared to drones piloted by humans. Autonomous UAVs have proven beneficial in a variety of scenarios, for example, in agriculture [1], [2], surveillance [3], environmental inspection [4]–[6] and exploration [7], road safety [8], [9] structural health monitoring [10], target detection and tracking [11] and leader following system [12]. Autonomous drones can assist in monitoring and exploring locations difficult for human access, even in harsh conditions. While many companies rely on UAVs for their operations, there is always the possibility of encountering unforeseen events during their missions, such as battery depletion, adverse weather conditions, and environmental changes that could compromise the safety of the flight.

UAVs can use various tools and enabling technologies to enhance navigation capabilities, including environmental sensors, computer vision, radio frequency sensors, and Artificial Intelligence (AI). AI and its recent boost provided by Deep Neural Networks (DNN) represent key drivers in developing autonomous vehicles. DNNs are particularly adapt at learning patterns from diverse sources, such as physical signals and images. Unfortunately, UAVs are vulnerable to high energy consumption due to their reliance on batteries with limited capacity. As a result, implementing high-accuracy DNNs on a UAV is a not negligible overhead to the autonomy of these drones. Low-power microcontrollers (MCU) could be a viable solution for deploying lightweight DNNs on a UAV; however, their limited computational resources challenge the design of efficient DNNs.

Researchers recently suggested to utilize cloud architectures to process data on remote servers [13], [14]. Although cloud computing is powerful, this approach relies on network connectivity and can introduce communication latency issues that compromise the real-time functionality of the system. Therefore, the most scalable and robust approach is Tiny Machine Learning (tinyML), which involves conducting all computations onboard. This method optimizes and adapts resource needs to match the hardware used. Thus, various studies have utilized deep learning-based systems to facilitate the autonomous navigation of UAVs. For example, [15] used a Recursive-Convolutional Neural Network combined with a feature-matching algorithm to localize landing areas and detect obstructions. The network was trained using a dataset of 250 images for 22000 epochs, achieving an accuracy of 90%. The system was then tested in the field using a Tello Ryze tech drone with a maximum frame-rate of 5 FPS. [16] presents a precise UAV landing system using YOLOv3, which uses a one-stage algorithm based on a darknet53 backbone and re-trains the pre-trained extraction network using 100 images for landing mark detection, resulting in 99% accuracy. The system's performance was evaluated in a simulated environment and on a real drone, exhibiting an error margin of only 0.3 meters for the landing maneuver. However, many current state-of-the-art approaches rely on either cloud architecture or single-board computers (e.g., Raspberry Pi, Google Coral, Nvidia Nano Jetson) to execute image and neural processing tasks. These solutions are inefficient due to their reliance on network connectivity and energy consumption. Therefore, we can increase the system's overall efficiency by using MCU-class devices to bring the processing closer to the sensor.

This paper presents a plug-and-play vision system based on the tinyML approach for an automatic landing application. The system uses an MCU-based camera, OpenMV Cam H7 Plus, which acts as a decision-maker based on the onboard inference result and as a path planner for drone control. In particular, this paper aims at providing the implementation of a tinyML-based plug-and-play automatic landing system by using an OpenMV Cam, extending the official documentation, showing how this constrained microcontroller may be used not only as an optical flow sensor but as an onboard computer, suitable for any drone endowed with the commercial-of-the-shelf Pixhawk flight controller. Additionally, we provide

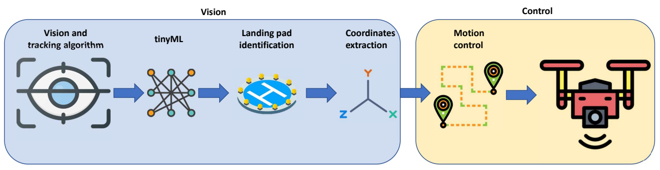


Fig. 1. Flowchart of the two running stages of the algorithm.

a system characterization considering energy consumption, frame rate, and a qualitative assessment of the accuracy of the landing maneuver by using simulations and real tests using a drone class-250 mm.

The rest of the paper is organized as follows. Section II presents and discusses the background and formulates the problem to solve. Section III presents the solution and the system design. Finally, experimental results and the hardware used in this work are presented in Section IV. Section V close this work with final remarks and possible improvements.

II. Background and Problem Formulation

The automatic landing system can be easily integrated with the Pixhawk flight controller[1], a widely used open-source autopilot for drone navigation and control. Our design combines the vision and tracking algorithms with the drone control algorithm and runs them on a single MCU. Although these processes are interconnected, they can be designed as separate stages in a unified pipeline. The flowchart in Figure 1 illustrates the system's pipeline. In the first stage, the vision and tracking algorithm detects and locates the landing pad, which is then used as an input to the drone control stage. The drone control algorithm uses this information to determine the drone's relative position to the landing pad. The two components are linked by the landing pad coordinates, which allows for easy replacement of the vision algorithm without compromising the drone control algorithm correctness.

A. Drone Control Algorithm

The Pixhawk flight controller offers a flight mode referred to as *off-board* mode (for the PX4 flight stack) or *Guided* mode (for Ardupilot flight stack), which enables the UAV control by means of a companion computer installed on-board. Companion computers such as Raspberry Pi 4, Nvidia Jetson Nano, Odroid, and Tegra K1 are commonly used, as they provide enough computational power to carry out real-time image processing tasks and control the UAV. Our approach utilizes a microcontroller with limited resources as a companion computer to acquire the images, execute the neural network and control the UAV. The companion computer communicates with the flight controller through UART, using MAVlink 1 as the messaging protocol. MAVlink 1 is a lightweight communication protocol that enables smooth communication between the on-board components.

The Cartesian coordinates of the camera, which is pointing downwards, is expressed in the UAV reference frame, dubbed

[1]https://pixhawk.org/

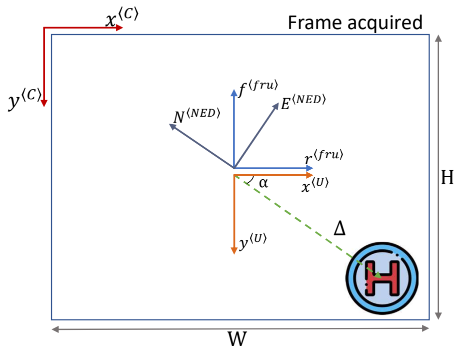


Fig. 2. Sketch of the reference systems used for drone navigation. $(x^{\langle C\rangle}, y^{\langle C\rangle})$ is the image reference system. $(x^{\langle U\rangle}, y^{\langle U\rangle})$ is an auxiliary reference system used for taking account of the drone position from the landing pad. (f, r) are the *Forward* and *Right* axes of the *Forward, Right, Down*, i.e., $\langle fru\rangle$ reference system, in which we compute the velocities to control the drone using velocity key points. Similarly, (*N, E*) are the *North* and *East* axes of the *North, East, Down*, i.e., $\langle NED\rangle$, reference system, in which we control the drone through positional key-points.

$\langle U\rangle$, in position $\mathbf{p}_c = [x_g, y_g]^T$, i.e., the coordinates of the geometric centre of the UAV. Thus, we can regard the centre of the acquired image as the position of the drone. For control purposes, the coordinates $\mathbf{l} = [x_l, y_l]^T$ of the centre of the landing pad are expressed in the camera reference frame, dubbed $\langle C\rangle$, whose origin is located in the top-left corner of the acquired image (see Figure 2 for reference). Therefore, to express $\mathbf{l}$ in $\langle U\rangle$ we need to perform a coordinate transformation:

$$x_l^{\langle U\rangle} = x_l^{\langle C\rangle} - \frac{w}{2}, \qquad y_l^{\langle U\rangle} = y_l^{\langle C\rangle} - \frac{h}{2}, \tag{1}$$

where $[x_l^{\langle U\rangle}, y_l^{\langle U\rangle}]^T$ express the centre of the landing pad in the UAV reference frame, $[x_l^{\langle C\rangle}, y_l^{\langle C\rangle}]^T$ express the centre of the landing pad in the camera reference frame, h and w respectively denote the height and width of the acquired image as reported in Figure 2.

The distance Δ between the UAV and the landing pad, as well as the angle α between the landing pad and the x-axis in $\langle U\rangle$, can be calculated as

$$\Delta = \sqrt{x_l^2 + y_l^2}, \qquad \alpha = \arctan(y_l, x_l). \tag{2}$$

Once the landing pad is detected and the distance between the UAV and landing pad centres is greater than a given threshold $\Delta \geq d_{thr}$, a proportional controller is applied to control the drone velocity, generating a velocity set-point in

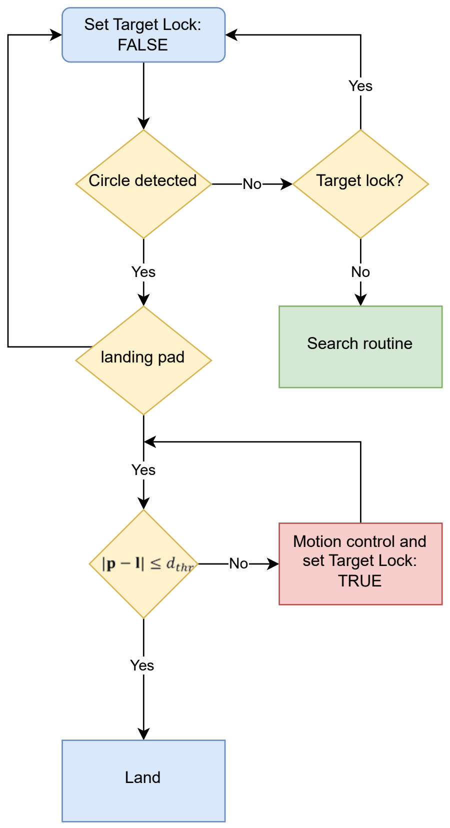


Fig. 3. Block diagram of the landing algorithm.

$\langle fru \rangle$ expressed in $\langle U \rangle$

$$[v_f, v_r, v_d] =$$
$$\begin{cases} [-K_p\Delta\sin(\alpha), K_p\cos(\alpha), 0], & \text{if } f, r \geq 0 \\ [-K_p\sin(\pi-\alpha), -K_p\cos(\pi-\alpha), 0], & \text{if } f \leq 0 \wedge r \geq 0 \\ [-K_p\sin(\pi-\alpha), -K_p\cos(\pi-\alpha), 0], & \text{if } f, r \leq 0 \\ [-K_p\sin(\alpha)|, K_p\cos(\alpha), 0], & \text{if } f \geq 0 \wedge r \leq 0, \\ [0,0,0], & \text{otherwise.} \end{cases} \tag{3}$$

The flowchart depicting the logical structure of the control algorithm is reported in Figure 3: notice the state *target lock*, which indicates if the landing pad has been recently detected and it is set to *True* when a landing pad is detected. On the contrary, if the landing pad is not visible, the algorithm waits for a sampling period before setting the flag to *False*. This strategy adds a dwell time to prevent sudden switches between the landing pad alignment and the mission task.

B. Communication interface

The OpenMV Camera is officially supported by Pixhawk as an optical flow sensor to aid the autopilot for velocity estimation. We develop an ad-hoc custom API set to extend the OpenMV as an on-board microcontroller for autonomous

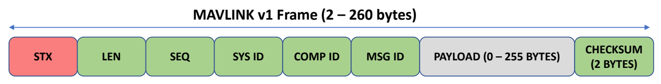


Fig. 4. MAVlink 1 packet format.

flight. The Mavlink 1 message structure[2] is reported in Figure 4, where

- **STX:** Protocol-specific start-of-text (STX) marker used to indicate the beginning of a new packet;
- **LEN:** Indicates the length of the following payload section (fixed for a particular message);
- **SEQ:** Detects packet loss. Components increment value for each message sent;
- **SYS ID:** ID of the system (vehicle) sending the message;
- **COMP ID:** ID of component sending the message. Used to differentiate components in a system (e.g., autopilot and a camera);
- **MSG ID:** ID of message type in payload. Used to decode data back into message object (1 Byte);
- **PAYLOAD:** Message data. Content depends on message type (i.e. Message ID) $(0-255$ Bytes);
- **CHECKSUM:** includes a 2 Byte CRC-16/MCRF4XX to allow detection of message corruption (2 Bytes).

The OpenMV Cam sends the inputs for the UAV through the UART port, which supports asynchronous serial data transfer. Data must be provided at a constant rate of minimum 2 Hz.

C. Problem formulation and solution overview

To perform a landing maneuver, a system that can extract the landing pad's coordinates in the UAV's reference frame from an onboard camera image is required. Moreover, we must ensure that the system's weight and power consumption do not compromise the UAV's flight autonomy. To address this challenge, we started with an optimized and quantized Neural Network (NN) for the OpenMV Cam H7 plus, developing custom APIs based on the same communication protocol used by the Pixhawk. The algorithm controls the drone for alignment with the landing pad and communicates with the Pixhawk flight controller ensuring low-power consumption.

III. TiNY NEURAL NETWORK FOR AUTOMATIC LANDING: THE NEWTON ALGORITHM

The acquired images are preprocessed to identify a secure landing zone within a video frame. Specifically, the safe landing area is characterized by a landing pad displaying an "H" symbol, which can appear against an orange, blue, or white background. To this end, the algorithm must first isolate a Region of Interest (RoI) that most likely contains the target within the frame. Then the CNN uses the RoI to generate a binary classification as output: either *Landing Pad* or *Not Landing Pad*. The training dataset comprises 13576 images for the *Landing Pad* class and 12807 images for the *Not Landing Pad* class. In Table I are reported the training parameters to

[2]https://mavlink.io/en/messages/common.html#mav_commands

TABLE I
NEURAL NETWORK TRAINING PARAMETERS.

Epochs	20
Batch Size	32
Initial learning rate	10^{-3}
Input image	Single channel
	64×64 px
Optimizer	Adam
Loss function	Binary cross-entropy
Source framework	Tensorflow

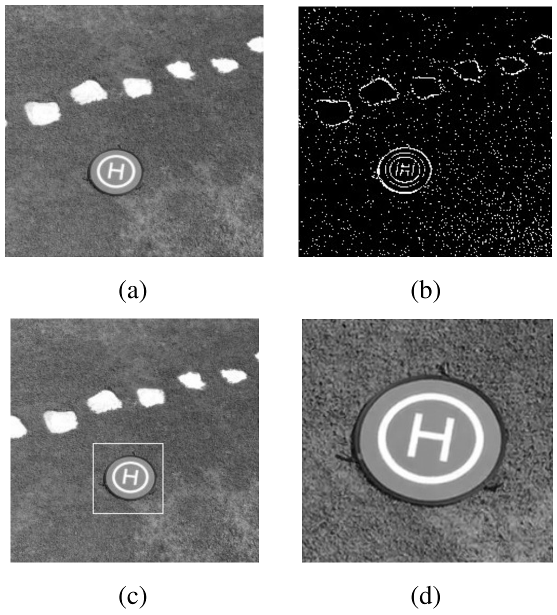


Fig. 5. Sample of vision algorithm workflow. (a) Original frame. (b) Frame after the application of the mean filter. (c) ROI highlighting after the application of the Hough transform. (d) Final crop to be given as input to the NN.

develop the proposed neural network The preprocessing stage is computationally intensive and it is critical: it has to carefully select all the necessary functions and algorithms and has to preserve the system performance.

NEWTON's preprocessing comprises the following stages:

- Acquisition of greyscale frames with QVGA resolution (320×240) (Figure 5-a);
- Backup of the original image for further investigation and debugging;
- Application of the mean operator for contour extraction (Figure 5-b).
- Application of the Hough Transform for identify all the circles in the image;
- Circles filtering by radius. In this case, it is chosen a minimum and maximum radius of 10 and 50 pixels, respectively;
- Storing the RoIs which contain the filtered circles (Figure 5-c);
- Crop of the RoIs collected taking as reference the centre of the circle with a width and height equal to $1.5 \cdot r$, where r is the circle radius (Figure 5-d).

We choose a mean operator instead of other operators, such as median or Canny operators, since it offers the optimal balance between contour highlighting and frame rate. Furthermore, we avoided using the conventional Gaussian blur filter for noise reduction as it considerably decreased the frame rate of the video stream.

The preprocessed Regions of Interest are fed into a tiny-NN for landing pad classification. The NN is based on a modified *LeNet-5* architecture [17] based on [18]. After obtaining the landing pad's center, the second stage of NEWTON employs the procedure outlined in Section II to steer the UAV.

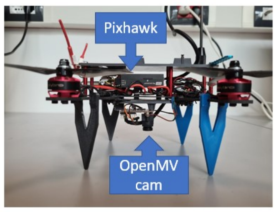
The original model size is around 77.2 MB, more than the available memory on the OpenMV Cam H7 plus. Considering the available limited memory, a NN compression is necessary to meet the hardware constraints. Therefore we use a full-integer quantization [19], a technique used to reduce the memory requirements and increase the execution speed of neural networks by representing the network weights and activations as integers instead of floating-point numbers. However, it can lead to slightly degrading the model's accuracy due to the loss of precision associated with the quantization. Thus, it is important to carefully choose the quantization parameters and optimize the model accordingly to minimize the impact on accuracy. The main drawbacks of this approach are that it only works with *tflite*-models and full integer quantization with *int*-encoding. After applying full-integer quantization to the original model size of 77.2 MB, we obtained a compressed model size of 6 MB, resulting in a compression factor of approximately 13 times. Subsequently, we conducted several tests to evaluate the board's performance, both with only preprocessing and with preprocessing/inference, thus achieving a frame rate of, respectively, 12 frame-per-second (fps) and 6 fps.

IV. RESULTS

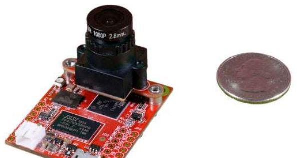
We evaluate the proposed landing system's effectiveness in a simulation environment and a real scenario. Furthermore, we evaluate the performance of the tiny-NN by measuring the accuracy, precision, recall, and f-score. The Pixhawk framework supports testing the algorithm in Software-In-The-Loop (SITL) and Hardware-In-The-Loop (HITL) configurations. Specifically, the SITL simulations allowed us to evaluate the algorithm's ability to identify the landing pad, calculate the landing pad center coordinates, and control the drone while excluding communication issues between the Pixhawk and OpenMV Cam H7 plus. Instead, the HITL simulations assessed the algorithm's performance on the target hardware and evaluated the communication linkage between the flight controller and the companion computer.

A. Hardware

We validate our work through a real-world test using a Holybro QAV250 quad-rotor class 250 mm, equipped with the Pixhawk 4 Mini flight controller and the OpenMV Camera



(a)



(b)

Fig. 6. In (a) the custom QAV250 drone prototype used for the practical testing. OpenMV camera is fixed at the drone center, such that it points straight down, in (b) the OpenMV Cam H7 Plus.

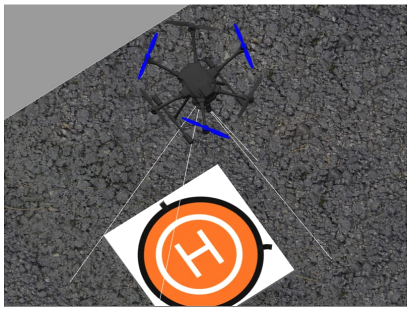


Fig. 7. Typhoon H480 hexacopter in the simulation environment. An instance where the UAV identifies the landing pad and initiates the alignment phase.

H7 Plus[3], reported in Figure 6. The OpenMV Cam H7 Plus features an STM32H743II ARM Cortex M7 processor, and among the communication interfaces it has an SPI bus that can run up to 80 MBs, an I2C bus up to 1 Mb/s and an asynchronous serial bus TX/RX up to 7.5 Mb/s. The OpenMV Cam H7 Plus is equipped with an OMNIVISION OV5645 5-megapixel system-on-chip image sensor capable of taking 2592×1944 images. The OpenMV Cam weighs only 17 g, excluding cables, and 20 g when cables are included. This weight confirms that the hardware has the potential as a suitable choice for medium-sized UAVs and mini-sized UAVs.

B. Software In The Loop Simulation and Hardware In The Loop Simulation

We developed the SITL simulations using Gazebo[4] and, as an example, we used a hexa-copter, specifically the *Typhoon H480*, equipped with a camera gimbal for the images acquisition. In Figure 7, we can see an instance where the UAV identifies the landing pad and initiates the alignment phase. This simulation validates the algorithm used in the automatic landing application, obtaining a mean error of approximately 5 cm. Then, we proceeded with the HITL simulation to test the communication between the OpenMV Cam and the Pixhawk flight controller. This test was conducted to ensure that the generation of the MAVLink messages and their transmission were correctly managed with the UART communication channel. This simulation validated the control algorithm performance and ensured the target hardware proper functioning.

[3]https://openmv.io/products/openmv-cam-h7-plus
[4]https://gazebosim.org/home

TABLE II
PERFORMANCE COMPARISON BETWEEN THE ORIGINAL MODEL [18] AND THE QUANTIZED MODEL.

Model	Accuracy	Precision	Recall	F-score
LeNet-5(original)	99.4	99.9	99.5	99.7
LeNet-5 (quantized)	93.5	94.0	94.0	93.0

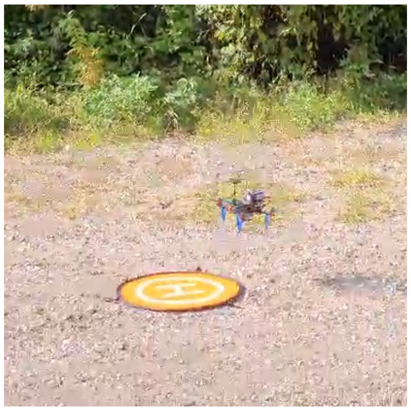


Fig. 8. Preliminary outdoor test results.

C. TinyNN test

To test the effectiveness of the tiny-NN, we used approximately 3000 images of a landing pad that were not included in the training session. In Table II, we report the accuracy, precision, recall, and F-score of the model, as well as a comparison with the original NN-model [18]. As explained in Section III, it is worth noting that the quantization process slightly degrades the model's performance. However, the model's classification accuracy is still satisfactory for identifying landing pads correctly.

D. Preliminary experimental results

At this writing stage, real-world testing has been carried out without the availability of ground truth, thereby requiring manual and qualitative evaluation by tape. Figure 8 shows a frame captured during an outdoor test, which successfully identified the landing pad, alignment phase, and landing manoeuvrer with an accuracy of approximately 30 cm as measured by a centimetre tape. The outdoor tests accurately identified the landing pad, the alignment phase, and the landing maneuver with a precision of approximately 30 cm. In this first version of the proposed architecture, the landing manoeuvrer does not operate in closed-loop once the landing pad is acquired and the UAV is aligned, resulting in the vertical descent being affected by wind flow, particularly the wash-effect generated by the propellers near the ground. Another challenge that impacts the network's performance in real-world experiments is the texture of the ground. Specifically, we have observed that certain ground conditions can influence the preprocessing stage, and consequently, the estimation of the landing pad centre.

TABLE III
ENERGY CONSUMPTION AND PROCESSING-TIME FOR ONE FRAME

	Energy consumption (mJ)	Processing-time (ms)
Pre-process	244.3	156.4
Classification	244.8	156.3

E. Energy consumption

Autonomous navigation systems require a high frame rate to guarantee smooth navigation. For these reasons, we characterize the proposed landing system considering the energy consumption and the frame rate. The energy consumption is measured by monitoring the voltage drop across a shunt resistor. To measure the frame rate, we trigger a GPIO of the OpenMV Cam to track the initiation and completion of the image processing task. Table III shows the energy consumption and processing time for one frame. Processing one frame takes 156.4 ms for the pre-processing task and 156.4 ms for the classification task, resulting in an overall frame rate of 3.2 fps. The total energy consumption is 489.1 mJ. Therefore, a typical UAV class 250 mm and equipped with a battery with a capacity of 1400 mAh, which theoretically provides a flight time of 15 minutes, experiences only a 2% reduction in flight time.

V. CONCLUSION

Automatic landing systems are gaining interest in the industry and research communities due to the expansion of UAVs within a wide range of human activities. Creating temporary landing zones is crucial to integrate this robotic platforms into natural environments and to provide resiliency to unforeseen conditions during the flight, such as under-voltage problems, engine failures, or radio signal loss.

Therefore, this paper presents a plug-and-play vision platform compatible with the Pixhawk flight controller series. The platform uses an MCU-based camera which implements both the safe landing zone detection and the path planning for the landing maneuver. The platform is assessed in simulation using real-world scenarios, validating the effectiveness of the solution. The designed platform takes in due account both the energy consumption and the frame rate. It performs a processing execution of 3.2 fps and has an energy consumption of 489.1 mJ per frame. Considering a drone powered by a battery of 1400 mAh which guarantees a flight time of 15 minutes, our experiments confirm the UAV autonomy-aware feature of the solution affecting the available energy by only 2%.

ACKNOWLEDGMENTS

Funded by the European Union under NextGenerationEU (iNEST project). Moreover the work was supported by the Italian Ministry for Education, University and Research (MIUR) under the program "Dipartimenti di Eccellenza (2023-2027)".

REFERENCES

[1] B. Yang, T. L. Hawthorne, M. Hessing-Lewis, E. J. Duffy, L. Y. Reshitnyk, M. Feinman, and H. Searson, "Developing an introductory uav/drone mapping training program for seagrass monitoring and research," *Drones*, vol. 4, no. 4, 2020. [Online]. Available: https://www.mdpi.com/2504-446X/4/4/70

[2] N. Delavarpour, C. Koparan, J. Nowatzki, S. Bajwa, and X. Sun, "A technical study on uav characteristics for precision agriculture applications and associated practical challenges," *Remote Sensing*, vol. 13, no. 6, 2021. [Online]. Available: https://www.mdpi.com/2072-4292/13/6/1204

[3] R. Akter, V.-S. Doan, G. B. Tunze, J.-M. Lee, and D.-S. Kim, "Rf-based uav surveillance system: A sequential convolution neural networks approach," in *2020 International Conference on Information and Communication Technology Convergence (ICTC)*, 2020, pp. 555–558.

[4] P. Tosato, D. Facinelli, M. Prada, L. Gemma, M. Rossi, and D. Brunelli, "An autonomous swarm of drones for industrial gas sensing applications," in *2019 IEEE 20th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 2019, pp. 1–6.

[5] S. Asadzadeh, W. J. de Oliveira, and C. R. de Souza Filho, "Uav-based remote sensing for the petroleum industry and environmental monitoring: State-of-the-art and perspectives," *Journal of Petroleum Science and Engineering*, vol. 208, p. 109633, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0920410521012675

[6] S. Ghosh, K. Ghosh, S. Karamakar, S. Prasad, N. Debabhuti, P. Sharma, B. Tudu, N. Bhattacharyya, and R. Bandyopadhyay, "Development of an iot based robust architecture for environmental monitoring using uav," in *2019 IEEE 16th India Council International Conference (INDICON)*, 2019, pp. 1–4.

[7] L. Santoro, D. Brunelli, and D. Fontanelli, "On-line Optimal Ranging Sensor Deployment for Robotic Exploration," *IEEE Sensors Journal*, vol. 22, no. 6, March 2022.

[8] F. Outay, H. A. Mengash, and M. Adnan, "Applications of unmanned aerial vehicle (uav) in road safety, traffic and highway infrastructure management: Recent advances and challenges," *Transportation Research Part A: Policy and Practice*, vol. 141, pp. 116–129, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S096585642030728X

[9] R. Rumba and A. Nikitenko, "The wild west of drones: a review on autonomous- uav traffic-management," in *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2020, pp. 1317–1322.

[10] Y. Tian, C. Chen, K. Sagoe-Crentsil, J. Zhang, and W. Duan, "Intelligent robotic systems for structural health monitoring: Applications and future trends," *Automation in Construction*, vol. 139, p. 104273, 2022.

[11] M. Andreetto, M. Pacher, D. Macii, L. Palopoli, and D. Fontanelli, "A Distributed Strategy for Target Tracking and Rendezvous using UAVs relying on Visual Information only," *Electronics*, no. 10, 2018. [Online]. Available: http://www.mdpi.com/2079-9292/7/10/211

[12] L. Santoro, M. Nardello, M. Calliari, W. C. Guarienti, G. Luchi, D. Fontanelli, and D. Brunelli, "Catch-me-if-you-can infrastructure-less uwb-based leader-follower system for compact uavs," in *2022 IEEE International Symposium on Robotic and Sensors Environments (ROSE)*, 2022, pp. 1–7.

[13] M. Salhaoui, A. Guerrero-González, M. Arioua, F. J. Ortiz, A. El Oualkadi, and C. L. Torregrosa, "Smart industrial iot monitoring and control system based on uav and cloud computing applied to a concrete plant," *Sensors*, vol. 19, no. 15, 2019. [Online]. Available: https://www.mdpi.com/1424-8220/19/15/3316

[14] S. Namani and B. Gonen, "Smart agriculture based on iot and cloud computing," in *2020 3rd International Conference on Information and Computer Technologies (ICICT)*, 2020, pp. 553–556.

[15] M.-F. R. Lee, A. Nugroho, T.-T. Le, S. N. Bastida *et al.*, "Landing area recognition using deep learning for unammaned aerial vehicles," in *2020 International Conference on Advanced Robotics and Intelligent Systems (ARIS)*. IEEE, 2020, pp. 1–6.

[16] J. Wang, D. McKiver, S. Pandit, A. F. Abdelzaher, J. Washington, and W. Chen, "Precision uav landing control based on visual detection," in *2020 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*. IEEE, 2020, pp. 205–208.

[17] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[18] A. Albanese, M. Nardello, and D. Brunelli, "Low-power deep learning edge computing platform for resource constrained lightweight compact uavs," *Sustainable Computing: Informatics and Systems*, vol. 34, p. 100725, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2210537922000609

[19] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," *arXiv preprint arXiv:2103.13630*, 2021.