

TDP: CVAR-UPM (Spain), Indoors Category

Javier Melero-Deza ¹, Carmen DR.Pita-Romero ¹, Rafael Perez-Segui ¹, Francisco J. Anguita ¹,
Pedro Arias-Perez ¹, Guillermo GP-Lenza ¹, Rodrigo Da-Silva-Gomez ¹, Pascual Campoy ¹ *

ABSTRACT

We present CVAR-UPM's indoor SAR micro-air vehicle for IMAV-2025, built to autonomously execute tunnel ingress, obstacle-field navigation, audio-guided victim localization with med-kit delivery, and landing on a moving, smoke-obscured platform. On Aerostack2's behavior-based stack, a lifecycle manager orchestrates perception, planning, and control. RGB-D is fused into a 2D occupancy grid (virtual laser + Bresenham), enabling online A* planning with time parameterization. State estimation uses an EKF fusing IMU and visual odometry across earth-map-odom-base frames. A nonlinear position MPC outputs thrust and body rates, followed by a calibrated thrust-map to counter battery-voltage effects; low-level actuation runs on Betaflight. Interaction modules include multi-object detection plugins (gates, landing sign) and a two-finger gripper. Acoustic bearing is estimated via rotor-aware spectral suppression, VAD + log-mel classification, and active yaw scanning. The system runs on Jetson Orin NX with RealSense D435i and has been validated in Gazebo Sim and initial real tests

1 INTRODUCTION

Here we can include a summarization of the competition rules; we should enumerate the problems from the competition and then reference them in the rest of the paper.

Unmanned aerial vehicles (UAVs) have matured from niche platforms into versatile tools across civil, industrial, and humanitarian domains. Recent surveys document rapid adoption in precision agriculture and environmental monitoring, power line and bridge inspection, construction progress tracking and digital twins, public safety operations, and time critical logistics by improvements in onboard sensing, autonomy, and fleet coordination. In disaster management specifically, UAVs provide fast, low-risk access to hazardous or obstructed areas and generate georeferenced imagery for situational awareness, mapping, and tasking capabilities that complement and often accelerate conventional ground response [1] [2] [3].

*¹ Computer Vision and Aerial Robotics Group (CVAR), Centre for Automation and Robotics (C.A.R.), Universidad Politécnica de Madrid (UPM-CSIC), 28006 Madrid, Spain. Correspondence e-mail: fj.anguita@upm.es

Mexico offers a compelling and sobering context for evaluating these technologies. The country's high seismicity arises from the subduction of the Cocos and Rivera plates beneath the North American plate (and regional interactions with the Caribbean and Pacific plates), producing frequent moderate-to-large earthquakes along the Mesoamerican Trench and strong site amplification effects in Mexico City's former lakebed sediments. Landmark events include the 19 September 1985 Michoacán earthquake (Mw 8.0–8.1) [4], the 19 September 2017 Puebla–Morelos intraslab earthquake (Mw 7.1) [5], and the 19 September 2022 Michoacán earthquake (Mw 7.6) [6], each causing widespread damage and testing emergency systems. These episodes spurred refinements to seismic zoning and reinforced the need for rapid post-event reconnaissance across dense urban fabric.

In this scenario, indoor's 2025 International Micro Air Vehicles Competition (IMAV25) proposed different UAV challenges for search and rescue in a 10x10(m) room emulating a disaster zone. Challenges seek to push the boundaries of the technology of micro air vehicles in topics like victim search, rapid access, aerial assessment of disaster zones, and the delivery of aid [7]. In order to fulfill these requirements, the proposed challenges for the indoor competition consist of solving five main tasks:

1. Enter the room through a tunnel-like made of gates.
2. Fly around the room without colliding with randomized located poles inside of it.
3. Draw a line on a whiteboard on the wall.
4. Find a location within the room where a cry for help is produced, and once found, deliver on the spot a med-kit.
5. Return to a moving landing platform where some smoke will be emitted on it, thus reducing visibility.

To tackle the challenge, we are using our own developed aerial's robotics modular framework Aerostack2 [8], where each module is designed to perform one task. The solution proposes a mission planner algorithm for activating and deactivating each module when needed. In this paper, we describe our approach and the solution developed aiming to win the competition.

2 HARDWARE

The hardware components integrated into the UAV are summarized in Table 1.



Figure 1: UAV

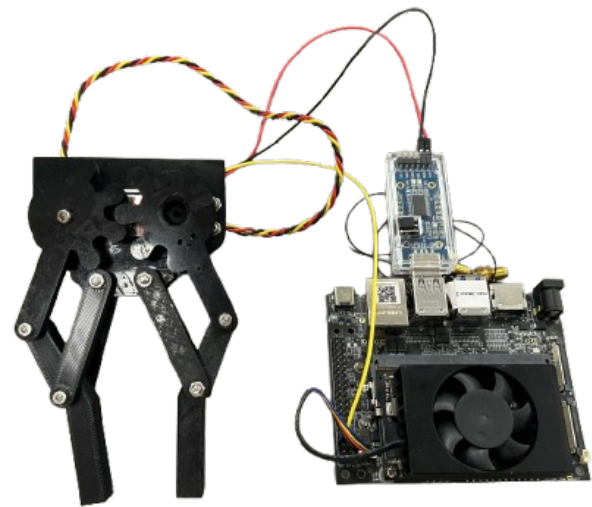


Figure 2: Custom gripper connected to the Onboard Computer.

Structure	Frame	DJI F330 quadrotor
	Propellers	5 inch and 3 blades
	Gripper	Custom two fingers gripper with SG90 servo motor
Electronics	Motors	2207 brushless 2100KV
	ESC	4in1 65A BL32 9 40V
	FCU	Foxeer H4 MPU6000 FC 8S
	Receiver	Ghost 2.4 GHz
Autonomy Module & Sensors	Onboard Computer	Nvidia Jetson Orin NX 16GB
	RGB Camera	Arducam 8MP IMX219 Wide Angle RGB Camera Rolling Shutter
	Depth Camera	Intel Realsense D435i
Power	Battery	Lipo1400mAh 22.2V 150C 6S1P

Table 1: UAV components

A quadcopter with a frame size of 330 mm was selected, as it provides enough space to mount the onboard computer together with the required sensors and actuators.

The UAV has a total mass of 1.5 kg and a maximum available thrust of 44 N, resulting in a thrust-to-weight ratio of 2.99. Power is supplied by a 6S 1400 mAh LiPo battery, which yields an approximate flight time of 5 minutes.

For flight control, the system mounts a Foxeer H4 MPU6000 processor running Betaflight firmware version 4.4.2. High-level autonomy is handled by an NVIDIA Jetson Orin NX, which enables the real-time execution of autonomous navigation algorithms and the management of multiple sensors and actuators.

Regarding perception, the UAV is equipped with a 6-axis IMU that provides measurements of orientation and acceleration. For visual sensing, two cameras are integrated: an RGB rolling-shutter camera for standard image acquisition and an Intel RealSense D435i depth camera, which also includes an onboard IMU.

For aerial manipulation, we use a custom gripper 2 with a total weight of 63.5g. It is built with two fingers actuated by an SG90 DC servo motor. Each finger has two segments that are connected by a revolute joint, which allows them to rotate relative to each other. Each finger's proximal segment is then attached to a common base via a second revolute joint, which enables the entire finger to rotate relative to the base. The total length of each finger is 110cm and have enough range to pick up and place a 20x10x5 cm box from its short side. The base has a square box to insert the servo motor. Additionally, the base is designed with a perpendicular rectangular base to which the gripper can be attached to the drone's frame.

The actuation of it is controlled by a module that remains active throughout the mission. This module has two states:

maintain the gripper open or keep it closed. These states can be modified as needed. The gripper communicates with a node that converts these commands into a pulse-width modulation (PWM) digital signal that is then received by the servo.

3 SOFTWARE

Our solution is single drone oriented. The system deployment is done onboard. Our software solution is based on Aerostack2 framework [8].

3.1 Overview

Aerostack2 is a novel framework for designing Aerial Robotics systems in an easy and versatile way. The main contributions of our work, compared with other solutions are:

- **Platform Independence** Offers platform-agnostic design, facilitating seamless integration across various drones and simulators, enhancing adaptability and interoperability.
- **Behavior-based mission definition and monitoring:** Implements behavior-based logic for mission control, providing a structured approach for task formulation and execution monitoring.
- **Plugin oriented architecture:** Enables dynamic reconfiguration and easy integration of new components, enhancing system adaptability and long-term maintainability.
- **Versatility:** Serves as a versatile framework supporting a modular architecture that allows users to easily customize or replace each module, providing flexibility for different application scenarios.
- **Validation in different showcases:** These contributions are validated in a variety of environments, including controlled laboratory settings and industrial simulations, demonstrating their reliability and suitability for real-world use.

As platform we choose Betaflight. Betaflight is a flight autopilot that supports a wide range of flight controllers from a variety of manufacturers [9]. We use several behaviors. The most basic ones were already implemented within Aerostack2, however, for this case scenario, we have implemented several new behaviors such as *GripperControl* and *DetectGates* behaviors. Our Navigation solution integrates a cloud map to occupancy grid utility, working alongside the *PathPlanning* behavior, which creates a path within the generated grid. for the state estimation we integrate visual-inertial Odometry output into an EKF2 module. We developed a new model based control optimized for the specific hardware we are using. We also integrate into our stack all the drivers needed for the other sensors and actuators such as the gripper, microphone, RGB and depth cameras.

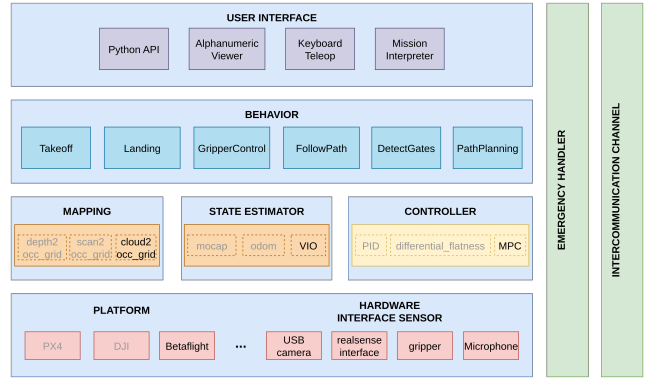


Figure 3: Aerostack2 used modules. This scheme includes the used and IMAV developed modules, along with some unused (grey) modules to illustrate other alternatives that might be used. Starting from user interface and high level components, followed by more specific tasks behaviors, going through the basic robotic functions such as mapping, state estimation, and controller, which are pluginized, finally ending with the platform and hardware interfaces in the lowest levels. We also dispose an emergency channel which can be triggered by any component at any time if a fault is detected.

3.2 Control

The control strategy relies on a model-based approach. This module is organized into two layers control loop.

3.2.1 High-Level control

The high-level controller uses a position-based model predictive control (MPC) scheme to compute rates $\nu(T, w)$ given a reference pose x_{ref} . Consecutively we use a Thrust Map to convert this thrust T into a collective throttle Th .

For this purpose, we work with a non-linear MPC. We define the following control input u representing the rates and state vector x .

$$u = \begin{bmatrix} T \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}, \quad x = \begin{bmatrix} p \\ v \\ \theta \end{bmatrix}, \quad (1)$$

The state is defined by the position $p(x, y, z)$, the orientation represented by the Euler angles $\theta(\phi, \theta, \psi)$ and the linear velocity $v(v_x, v_y, v_z)$. The control input is defined by the thrust T and the angular velocity $w(w_x, w_y, w_z)$.

This variables are related as follows:

$$T = m a_i = m \frac{dv_i}{dt}, \quad \forall i \in \{x, y, z\} \quad (2)$$

$$w_i = \dot{\theta}_i, \quad \forall i \in \{x, y, z\} \quad (3)$$

As we work with a position control loop, the references

provided to the model consist of a set of poses.

$$x_{ref} = \begin{bmatrix} p_{ref} \\ \theta_{ref} \end{bmatrix}, \quad (4)$$

These are defined by the position $p_{ref}(x, y, z)$ expressed in the inertial frame and orientation vector $\theta_{ref}(\phi, \theta, \psi)$, representing roll, pitch, and yaw, respectively in the body frame. Finally, the system dynamics are described by $\dot{x}$:

$$\dot{x} = \begin{bmatrix} \dot{p} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v \\ a \\ \omega \end{bmatrix} \quad (5)$$

Based on the above, the nonlinear optimal control problem (OCP) is formulated with input and state constraints. In this case, the state is limited only by velocity bounds.

$$\begin{aligned} J = \min_U \sum_{k=0}^{N-1} & \left(\|x_k - x_{ref, k}\|_Q^2 + \|u_k - u_{ref, k}\|_R^2 \right) \\ & + \|x_N - x_{ref, N}\|_P^2 \\ \text{s.t. } x_0 &= \hat{x}_0, \\ u_{\min} &\leq u_k \leq u_{\max}, \quad k=0, \dots, N-1, \\ x_{\min}(v_k) &\leq x_k \leq x_{\max}(v_k), \quad k=0, \dots, N, \end{aligned} \quad (6)$$

where $Q \succeq 0$, $R \succeq 0$ are the state and input penalty matrices, while $P \succeq 0$ is the terminal state error penalty. Furthermore $x_{ref, k}$, $u_{ref, k}$ are the target state and target control input respectively at time k .

This MPC's output provides the desired commanded thrust, which must then be converted into a collective throttle command Th . This conversion is performed using a calibrated mapping function called the Thrust Map (TM). The TM is a nonlinear function that accounts for the influence of battery voltage on available motor power.

$$Th = f(T, B) \quad (7)$$

f is a polynomial surface approximated from experimental data consisting of triplets of the form $Ds_i(T_i, B_i, Th_i)$ for $i = 1, \dots, N$.

However, there are discrepancies between the commanded thrust and the actual thrust produced due to calibration inaccuracies and in-flight effects. To compensate for this, we introduce a correction factor computed from experimental measurements consisting of triplets $Df_j(B_j, T_{c_j}, F_{m_j})$, for $j = 1, \dots, N$, where T_{c_j} is the thrust command sent to the TM , and F_{m_j} is the measured z-force obtained, using the vertical acceleration of the drone a_z and mass m .

The correct model is formulated as:

$$Th = \hat{f}(T_\gamma, B) = \hat{f}(\gamma(B) \cdot T_c, B) \quad (8)$$

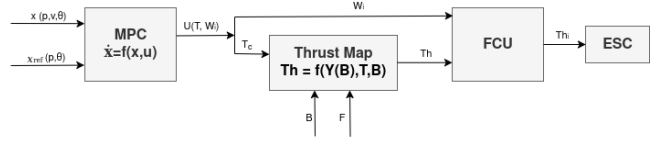


Figure 4: Control Architecture Modules. The high-level controller consists of the position-based model predictive control (MPC) and thrust map (TM) modules. The low-level controller is the flight controller unit (FCU). The Electronic Speed Control (ESC) receives the controller's output.

3.2.2 Low-Level Control

The low-level controller converts the Thrust Map (TM) output into individual motor throttle commands Th_i from the collective throttle and angular velocity rates (Th, ω_i).

Each Th_i is then converted into a servo signal in the range 1000–2000. In our implementation, we use Betaflight [9] as the Flight Controller Unit (FCU) for this task.

The Electronic Speed Control (ESC) communicates this normalized signal to actuate the motors.

The overall control workflow is illustrated in Figure 4

3.3 Navigation

The navigation component enables the aerial robot to move autonomously through the environment using data from the onboard RGB-D sensor from the D435i. Depth information is processed to build an occupancy map of the surroundings, which serves as the basis for path planning. An A* search algorithm is then employed to compute an optimal path toward the designated goal point.

For clarity, this component can be divided into two main submodules: mapping, responsible for constructing and updating the occupancy representation, and path planning, which determines the collision-free trajectory to the target.

3.3.1 Mapping

To simplify the representation of the environment, the mapping submodule employs a 2D occupancy grid, under the assumption that most obstacles in the competition scenario (cylindrical or prismatic structures) extend uniformly across height as show in image 5.

Depth information from the RGB-D camera is converted into a virtual laser scan (extracted from a single row of the depth image) using a standard ROS 2 perception package¹. Sensor rays are then efficiently projected onto grid cells through the Bresenham line algorithm, marking free and occupied spaces according to the measured distances.

¹https://github.com/ros-perception/depthimage_to_laserscan/tree/ros2

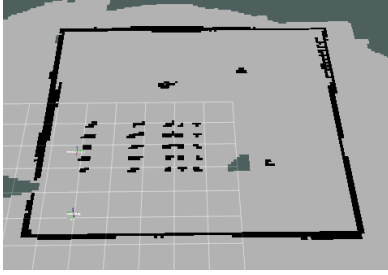


Figure 5: Generated occupancy grid

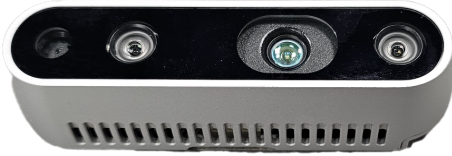


Figure 6: D435i Camera, providing both depth and stereo infrared sensors.

A probabilistic update scheme is applied: each hit increases the occupancy probability of a cell, each miss decreases it, and unknown cells remain unchanged. The probabilities are clipped between fixed bounds (0 to 100) to maintain consistency across updates. The resulting map is maintained in a global reference frame (world frame in the TF tree).

Furthermore, the implementation allows for customization through ROS 2 parameters, including grid resolution, sensor range limits, and update confidence values for hits and misses, enabling adaptation to different mission requirements. To prevent the ground from being falsely detected as an obstacle during takeoff or landing, measurements below a minimum threshold of 1 meter are discarded.

3.3.2 Path Planning

Guided by the chosen map representation, the path planning submodule adopts a graph-based search approach, specifically the A* algorithm, to compute the shortest and cost-optimal paths through the environment. Each graph node $n(i, j)$ corresponds to a cell in the occupancy grid at position (i, j) . A node can have up to eight connections, one for each neighboring cell, and is evaluated using a cost function $f(n)$.

A* operates by combining the actual cost of reaching a node, $c(n)$, with a heuristic estimate of the remaining distance to the goal, $h(n, G)$. Considering a node n and a goal node G , the evaluation function is defined as:

$$f(n) = c(n) + h(n, G) \quad (9)$$

$$c(n) = \alpha + f(n_{par}) \quad (10)$$

$$h(n) = \|n, G\|_2 \quad (11)$$

$$(12)$$

Here, α is a constant factor that accounts for the cost of moving to a neighboring node, and n_{par} is the parent node of n in the search graph.

To improve computational efficiency, the algorithm does not rely on a pre-computed graph of the entire occupancy grid. Instead, the graph is generated dynamically during the search, creating nodes and edges only as required by the algorithm's progression.

The output of this process is a sequence of waypoints that define the planned path. These waypoints are then processed by the motion planning layer, where a cruise speed reference is assigned, and subsequently by the trajectory planning layer, which generates a smooth time-parameterized trajectory. This trajectory reference is finally delivered to the controller to command the drone along the planned path.

3.4 Localization

The component in charge of computing a localization of the drone in the flight arena is called state estimator. This component of the Aerostack2 framework defines a reference frame system following the ROS convention, defining the origin in the world (earth), the static position the drone is at the start (map), the correction of the drift of the estimation (odom), and the local position of the drone computed by the drone sensors (base_link).

The state estimator component in Aerostack2 allows the use of multiple plugins as shown in 7. The user can configure which plugin is used to compute each frame position depending on the application. For this specific scenario, the EKF plugin developed to solve the problem of having multiple data sources that need to be fused into a localization as precise as possible, is the one selected.

This state estimator plugin has two different steps, the predict and the update. The predict step integrates the sensor data, in this scenario from the IMU, at a very high rate to get an estimation of the state of the drone, updating the odom-base_link reference frame. This data can be noisy, so the integration is not as precise as needed, so the covariance of the state rises. Then, the predict step, that works at a lower rate, receives information from other sources, like camera movement estimation or known position object detection. This information is usually more precise, so the plugin updates the map-odom reference to correct the error. The earth-map reference frame is static and defined in a configuration file.

3.5 Detection

The software component to perform the detection has been fully integrated within the Aerostack2 framework as a

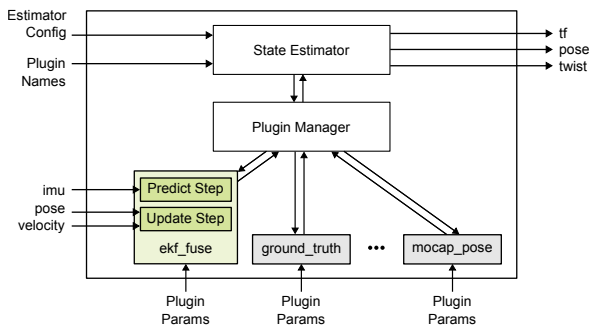


Figure 7: Localization diagram. All plugins defined in the configuration work in parallel to achieve the best estimation. Each plugin has its own configuration, and its own inputs. The estimator manages the information from the plugins and outputs the reference frames, the pose and twist.

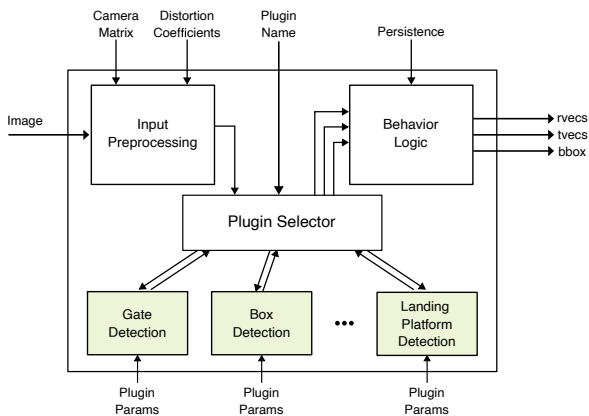


Figure 8: Detect behavior diagram. Camera matrix, distortion coefficients, plugin name and persistence are parameters for the detection process while each plugin may handle its own parameters.

behavior that encapsulates the logic of the detection process. This behavior can be configured to run continuously or to end once an object is detected with a confidence value above a specified threshold. Additionally, camera parameters for image undistortion can also be configured to provide robustness to lens distortion effects.

This behavior allows for the use of detection algorithms for many different objects, which are embedded in plugins that can be dynamically loaded. This approach provides modularity and adaptability, facilitating the integration of different detection strategies as shown in 8 like classic approaches based on image filtering, used to detect known elements such as the landing platform and other deep learning based architectures like YOLO, which have been useful to detect gates in real environments.

Using this approach, the system is able to detect multiple objects during the mission, complying with a wide range of application requirements and computational constraints and

enabling the drone to recognize and position elements in the environment.

3.6 Mission Planning

The mission planner coordinates the execution of the competition tasks by managing the lifecycle of the required modules. For each mission stage, it initializes the relevant modules, monitors their progress against predefined completion conditions, and, once these conditions are satisfied, gracefully terminates them before activating the modules for the subsequent stage. This ensures a structured and reliable progression through the mission.

4 INTENDED SOLUTION

Details about the intended solution for each competition problem.

4.1 Simulation

In order to test our solutions we have created a simulation environment following IMAV's 2025 indoor competition specifications using Gazebo Sim simulator [10]. This simulator enables a high precision physics engine, paired with a decent graphics engine which allows for visual representation of objects. It can simulate the whole array of sensors and actuators that are going to be used for the challenge. It's main advantage is its direct integration with ROS 2 through the ros-gz bridge [11].

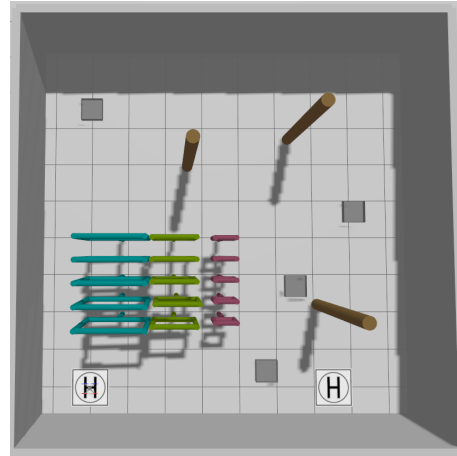


Figure 9: Simulation environment used for testing. Gazebo allows the recreation of all elements and obstacles to fully test our solution to the missions.

4.2 Mission 1: Tunnel crossing

To cross the tunnel and enter the arena, the system first detects the tunnel entrance in the RGB camera, and visually anchors the central point of the entrance in the map. Once the point is localized, the MAV follows a straight line trajectory through the tunnel to cross it. During this trajectory, depth information of the onboard camera is continuously processed

to apply corrections to the trajectory, ensuring the MAV traverses through the tunnel while keeping a safety distance to the walls.

4.3 Mission 3: Medkit deployment

We use a custom gripper to overcome this challenge.

One of the main concerns for this mission was the change in mass during operation. To address this, we added an additional module to the control system to estimate the mass. The module updates the parameters smoothly at a predefined frequency, in order to avoid undesired behaviors that could lead to malfunction or even a crash.

With a single onboard microphone a cry for help can be detected and a coarse bearing estimated by means of active scanning. The first step consists of suppressing rotor noise using signal-aware filtering: motor RPM obtained from ESC telemetry is used to compute the blade-pass frequency $f_{BPF} = \frac{RPM}{60} \times \text{blades}$, followed by adaptive IIR notch filters at multiples $k f_{BPF}$ with $k = 1, 2, \dots$. A voice band-pass (e.g., 300–3500 Hz) and an adaptive noise canceller (LMS/NLMS) that employs an RPM-derived reference $r[n]$ (or a microphone located near a motor) further improve the signal-to-noise ratio; the error signal $e[n]$ represents the “cleaned” stream. A lightweight VAD is then applied to gate frames, after which log-mel spectrogram patches are classified by a compact CNN/CRNN trained for “help/ayuda/scream” versus rotor, wind, and urban noise. Training should involve heavy augmentation (insertion of rotor noise, Doppler shifts, and reverberation). Final decisions are smoothed with hysteresis, maintaining a rolling estimate of frame SNR and the class posterior p_{help} .

To transform detection into guidance without an array, an active yaw scan is performed in order to maximize detected-signal quality. The drone is rotated at a slow, constant rate while frames are binned by heading ψ ; for each bin a score is computed as $S(\psi) = \alpha \text{SNR}(\psi) + (1 - \alpha) \log p_{\text{help}}(\psi)$. The estimated bearing is then $\hat{\psi} = \arg \max_{\psi} S(\psi)$. The platform is oriented toward $\hat{\psi}$, paused to confirm with new VAD/classifier output, and the scan-approach cycle is repeated while maintaining safe standoff to mitigate wind and multipath effects. When available, a supercardioid or shotgun microphone mounted on a micro-gimbal can be servo-scanned instead of rotating the airframe, which sharpens the $S(\psi)$ peak and reduces self-noise. Absolute loudness is treated only as a weak, non-metric cue for distance, since source intensity and propagation vary; repeated bearings over motion are fused to home in on the source, with an exponential moving average on $\hat{\psi}$ used to stabilize control.

4.4 Mission 4: Platform landing

To take on the moving platform landing challenge, a Position Based Visual Servoing control algorithm is implemented. The drone will first approach the landing area, as its approximate location is known. During this movement, a classic vision algorithm will run over the images obtained by the

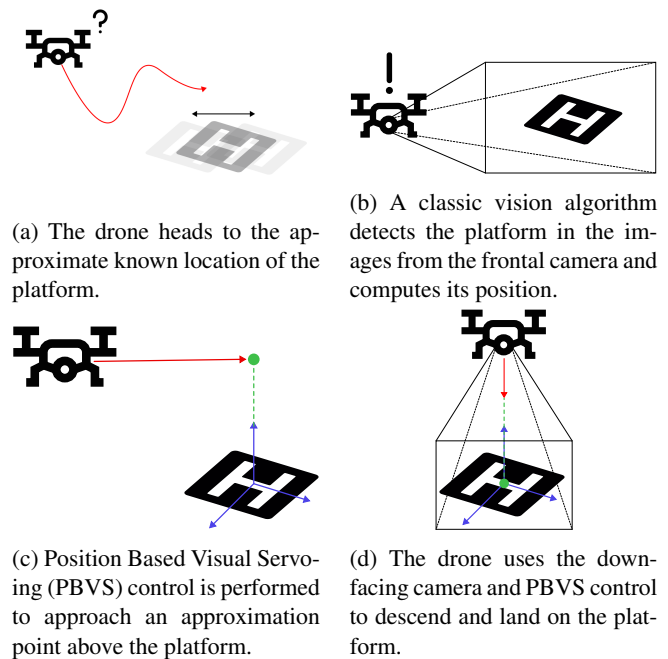


Figure 10: Platform landing behaviour step-by-step description.

frontal D435i camera to detect the platform marked with a typical landing sign. From the detection in the image and the camera calibration data, the exact pose of the landing platform is computed. This pose obtained is used as an input reference for our position MPC controller.

When the drone is close enough to the landing platform, the down-facing RGB camera replaces the frontal one, as the last one will not be able to detect the platform. A second proximity threshold will indicate our landing behaviour that drone is too close to the platform for the landing sign to be detected, so the final part of the landing is carried out with no information of the platform’s pose, but with an estimation of its pose considering the speed at which it was previously moving. The landing behaviour and its different steps are depicted in Figure 10.

REFERENCES

- [1] S. A. H. Mohsan, N. Q. H. Othman, Y. Li, M. H. Alsharif, and M. A. Khan. Unmanned aerial vehicles (UAVs): practical aspects, applications, open challenges, security issues, and future trends. *Intelligent Service Robotics*, 16(1):109–137, 2023.
- [2] Sharifah Mastura Syed Mohd Daud, Mohd Yusmi-aidil Putera Mohd Yusof, Chong Chin Heo, Lay See Khoo, Mansharan Kaur Chainchel Singh, Mohd Shah Mahmood, and Hapizah Nawawi. Applications of drone

- in disaster management: A scoping review. *Science & Justice*, 62(1):30–42, 2022.
- [3] Mingyang Lyu, Yibo Zhao, Chao Huang, and Hailong Huang. Unmanned aerial vehicles for search and rescue: A survey. *Remote Sensing*, 15(13), 2023.
- [4] R. P. Masse and R. E. Needham. NEIC; the National Earthquake Information Center. *Earthquakes & Volcanoes (USGS)*, 21(1):4–45, 1989. USGS Publications Warehouse.
- [5] Mehmet Celebi, Valerie Sahakian, Diego Melgar, and Luis Quintanar. The 19 september 2017 M 7.1 puebla–morelos earthquake: Spectral ratios confirm mexico city zoning. *Bulletin of the Seismological Society of America*, 2018. Index ID: 70236833; Record source: USGS Publications Warehouse; USGS Organization: Earthquake Science Center.
- [6] U.S. Geological Survey. M 7.6 - 35 km ssw of aguillilla, mexico. USGS Earthquake Event Page, September 2022. Event time: 2022-09-19 18:05:08 UTC.
- [7] IMAV 2025. Competitions’ rulebook for imav 2025. IMAV 2025 Official Website, 2025. 16th International Micro Air Vehicle Conference and Competition (IMAV) 2025; Nov 3–7, 2025; San Andrés Cholula, Puebla, Mexico.
- [8] Miguel Fernandez-Cortizas, Martin Molina, Pedro Arias-Perez, Rafael Perez-Segui, David Perez-Saura, and Pascual Campoy. Aerostack2: A software framework for developing multi-robot aerial systems. *arXiv preprint arXiv:2303.18237*, 2023.
- [9] Betaflight Developers. Betaflight: Open-source flight controller firmware. <https://github.com/betaflight/betaflight>, 2025. Accessed: 2025-09-01.
- [10] Open Robotics. Gazebo sim: Open-source robotics simulator. <https://gazebo.org/libs/sim/>, 2025. Accessed: 2025-08-29. ROS 2 integration via ros_gz.
- [11] Gazebo Project. ros_gz: Integration between ros 2 and gazebo. https://index.ros.org/p/ros_gz/, 2025. Accessed: 2025-08-29.