

Simplifying Autonomous Aerial Operations: LUCAS, a Lightweight framework for UAV Control And Supervision

J.I. Murillo ¹, M.A. Montes ¹, R. Zahinos ¹, M.A. Trujillo ¹, A. Viguria ¹, G. Heredia ²

Abstract—This work introduces LUCAS³, an open-source framework designed to control and monitor highly autonomous UAV systems. This framework is composed of two modules, the Control Manager Finite-State-Machine (FSM), an efficient and easily extensible state machine that controls the behavior of the robot during the mission, and the Cascade Controller, which handles the commands to the low-level autopilot. The methodology followed to develop the framework has been the use of C++ and a modularized implementation, to separate communications from core functionality and allow the use of diverse middlewares, in this case, ROS and ROS2. The system can be integrated with other software nodes to form a complete autonomous setup, which has been successfully tested in a simulated environment and in a real scenario, where a quad-rotor has to perform an indoor inspection in a building under construction.

I. INTRODUCTION

The autonomy of UAVs is a great advantage when looking for applications in different fields. They are suitable for inspection tasks involving industrial infrastructure, such as power lines [1] or pipelines in oil refineries [2]. The ability of these systems to reach remote spaces and perform these tasks autonomously gives them a great advantage in efficiency and safety over manual methods traditionally used in the industry. This autonomy can also provide value in activities such as image acquisition to create 3D models [3], construction monitoring [4], or to develop multi-UAV systems with advanced algorithms that allow surveillance of a large area [5].

In order to achieve the necessary autonomy in the robotic aerial system, it is necessary to develop some critical software components [6]: perception, localization and mapping, mission/task planning, motion planning and obstacle avoidance, states estimation or control.

Several previous works focus on building a complete system with all the modules necessary to achieve autonomous tasks, which is especially useful when trying to develop operations in complex environments. In [7], a work is presented in which the purpose is to map a complete forest environment with the help of a UAV. To this end, they use a stack for autonomous flight that contains the mapping, estimation, planning, control, and autopilot interface modules [8]. Since this stack solves the main

problems for autonomous flight with aerial robots, they can focus on integrating a new module, such as the semantic slam for this specific case. Selecting a middleware to connect the software modules is essential to guarantee an easy integration. In this case, ROS is the one used.

A different approach can be found in Aerostack2 [9], where a system is designed for ROS2-based multi-robot aerial systems. Here, modules such as sensor-actuator interface, basic robotic functions, behaviors, plan execution control, and mission control are implemented. The modular architecture enables users to integrate custom modules in the stack.

Another highlighted framework is the MRS UAV system [10], which also implements estimation and control modules to build a complete autonomous system based on ROS Noetic and using a PX4 [11] flight controller. In our work, we have been inspired by components belonging to this stack, such as the UAV Manager and the Control Manager.

The integration of these stacks with the Flight Control Unit (FCU) is one of the problems that this type of development has to solve. In UAL (UAV Abstraction Layer) [12], the work is focused on facilitating integration with open source firmware such as PX4, ArduPilot [13] or proprietary firmware such as DJI A3 [14] or N3 [15]. In addition to simplifying interfaces, UAL provides basic commands such as take-off, landing, velocity control, position control, or the management of safety pilot authority. It also uses ROS as middleware, allowing the standardized use of services or topics to perform these actions.

The tools and frameworks presented here show that it is important to manage the different states of the system, whether it is to ensure that the behavior during the mission is as expected, to increase the security of the operation, or to let the user be aware of exactly what is happening, especially in complex scenarios. One of the most widely implemented tools for the management of such systems is the FSM. It represents the conditions in which the system can exist (states) and how the system moves from one state to another based on inputs or events (transitions).

State machines are used in many UAV applications to control the behavior of the system: a mission in which an aircraft has to track a dynamic object autonomously for surveillance [16], where it is necessary to switch between hovering, searching and tracking tasks to ensure the success of the operation; individual and general state in cooperative multi-UAV mission [17]; the inspection of a pipe in a petrochemical refinery, differentiating between states such as search, landing, or take-off [2].

¹CATEC (Advanced Center for Aerospace Technologies), Calle Wilbur y Orville Wright, 19, La Rinconada, 41309 Sevilla, Spain; [jimurillo, mmontes, rzahinos, matrujillo, avigurial]@catec.aero

²Robotics, Vision, and Control Group School of Engineering, University of Seville, Seville, Spain.

³<https://github.com/catec/lucas>

Almost all tools, frameworks, and works presented here use ROS as middleware, which has popularized the use of SMACH [18] for the software implementation of these finite-state machines. The SMACH library allows developers to model robotic behavior as a collection of modular states and transitions. As it interacts directly using ROS communications, it is easy to prototype an FSM with this tool. However, since it is designed for high-level tasks, it is not intended for applications with demanding requirements. Since it only supports Python, its usability is limited in C++-based projects that are more appropriate for high-efficiency needs. There are some examples of implementations using SMACH to manage the autonomous behavior of robots. In [19], the navigation and mapping of an environment with a terrestrial robot is achieved by controlling the states and transitions with this library. An FSM that implements the search and tracking task using this tool is described in [20].

This work describes LUCAS, a framework for improving the autonomous capabilities and safety of aerial robotic systems, specially designed for research purposes. With only two modules, we cover the main problems when working with this kind of system:

- Control Manager FSM: its main function is to handle the state of the aircraft and to create or adapt the necessary control references according to the flight status. In addition, it manages the authority of the UAV, only allowing us to send autonomous commands when the safety pilot enables us to do so. This module is the bridge between the low-level actions of the aerial robot and the other modules such as planning, exploration, or tasks given directly by the user.
- Cascade Controller: based on a cascade architecture. Position or velocities references can be used as inputs, as well as a trajectory defined by position, velocity, and acceleration.

The key advancements that distinguish this work from other existing systems are:

- The FSM is implemented using the TinyFSM [21] header-only library, designed for optimal performance and low memory footprint, which makes the framework lightweight and resource efficient.
- This system is designed with a modular architecture that allows it to interact with other software nodes that complete all functions of the system, such as route planners or odometry algorithms.
- Customization is a key feature of the development, allowing users to create new states and transitions guided by a base state.
- Although the FSM has been implemented and validated using ROS and ROS2, it remains ROS-agnostic, with the communication layer clearly separated from the functionality, ensuring flexibility for integration with other middlewares.
- Provides a robust controller that has been successfully validated in multiple scenarios, allowing integration

with MAVROS-compatible autopilot firmware such as PX4 or ArduPilot.

- The framework is robust in terms of safety, as it allows the FCU to focus only on attitude control, without the need to send positioning data in GNSS-denied environments. This enables a safer switching to manual mode if there is a problem, which is especially crucial for research projects.

The manuscript is structured as follows: in Section II, the framework is placed within all the modules needed to build an autonomous aerial robotic system. Then, the design and implementation of two main modules are covered in Section III (Control Manager FSM) and Section IV (Cascade Controller). Experimental validation is addressed in Section V, where we will show how the framework behaves in different systems, with various aircraft, software modules, and use cases. Finally, the conclusions are presented in Section VI.

II. GENERAL FRAMEWORK SCHEME

As discussed in Section I, putting an UAV in flight and carrying out an autonomous mission require several critical modules to be implemented depending on the complexity of the mission and its purpose. In general terms, the necessary modules that we have identified can be seen interconnected in Fig. 1.

Sensors are required to provide information about the environment to the guidance, navigation, and control modules, which may be in the form of, for example, LiDARS, visual cameras, GNSS receivers, inertial sensors, altimeters, etc.

Navigation modules, using information from sensors, are capable of computing their location in the environment, either relative (how far UAV has traveled) or absolute (where UAV is with respect to a map), using algorithms such as Visual Inertial Odometry, LiDAR Odometry, SLAM, and using estimators such as LKF, EKF or UKF, being able to merge data from various sources.

Guidance modules use the information from the sensors to calculate the route to the destination in a safe and obstacle-free way, offline or real time, creating a dynamically safe and feasible trajectory, by using Global-Local planners and Obstacle Avoidance algorithms that can be based, for example, on point cloud or image processing.

Control Manager module has several information inputs from navigation and guidance modules, and in conjunction with the commands it receives from the user, establishes the states of the aircraft and sends the appropriate control actions for that state to the controller. The controller simply applies the programmed control laws to adjust the multi-rotor to the references.

Finally, the FCU driver communicates with the autopilot firmware, receiving sensor, battery or status data, low-level status, and sending control references and mode-change commands.

Now it is clear what role our system plays in a complete autonomous UAV system.

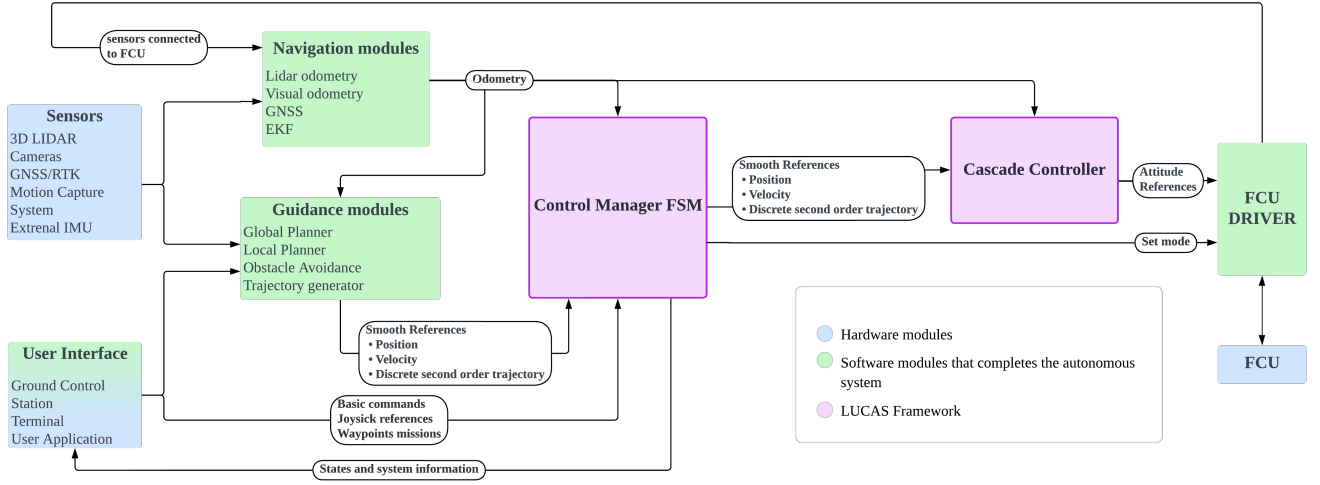


Fig. 1: LUCAS framework into the scheme of modules needed to develop an autonomous aerial robotic system.

III. CONTROL MANAGER FSM

In this section, we present the complete development of the FSM Control Manager FSM. We describe each of the functionalities of the states and the transitions between them, the significant modules that are part of the architecture, and details of the implementation.

A. States and transitions

A complete diagram of all states and the allowed transitions between them can be seen in Fig. 2. In the section for each state, we provide a brief explanation of what happens, what is implemented, and a table that represents all possible transitions.

1) *UNINITIALIZED*: The system is not initialized, there is no communication with the FCU. Nothing happens.

State to Transit	Condition
LANDED DISARMED	Communication with FCU starts

2) *LANDED DISARMED*: The system is initialized, there is communication with the FCU, and the aircraft is on the ground and disarmed.

State to Transit	Condition
LANDED ARMED	Safety pilot arms UAV with RC

3) *LANDED ARMED*: The aircraft is armed on the ground and has no authority to send commands to the FCU from the companion computer.

State to Transit	Condition
FLYING MANUAL	Safety pilot takes off the UAV
LANDED AUTO	Authority is requested by the user

4) *FLYING MANUAL*: The safety pilot controls the aircraft.

State to Transit	Condition
LANDED DISARMED	Aircraft is disarmed
LANDED ARMED	Land detection and not disarmed
HOVER	Hover is requested

5) *LANDED AUTO*: The aircraft is armed on the ground and has the authority to send commands to the FCU.

State to Transit	Condition
TAKING OFF	Takeoff is requested

6) *TAKING OFF*: The control reference to execute the takeoff is computed and sent to the controller. Waits until the height error converges.

State to Transit	Condition
HOVER	UAV reach the designed takeoff height
FLYING MANUAL	Safety pilot revokes authority
EMERGENCY	Critical messages are not being received

7) *HOVER*: The control reference is set to the current position of the UAV. If it has a velocity module above a threshold, it decelerates before setting the new reference.

State to Transit	Condition
GOING TO WP	To go to a defined way-point is requested
ASSISTED	Set assisted mode is requested
OFFBOARD	Set off-board mode is requested
PRELANDING	Perform a landing is requested
FLYING MANUAL	Safety pilot revokes authority
EMERGENCY	Critical messages are not being received

8) *GOING TO WP*: A time-parametrized trajectory is computed to reach the desired way-point considering the maximum acceleration and velocity parameters, and is sent to the controller.

State to Transit	Condition
HOVER	Hover is requested
FLYING MANUAL	Safety pilot revokes authority
EMERGENCY	Critical messages are not being received

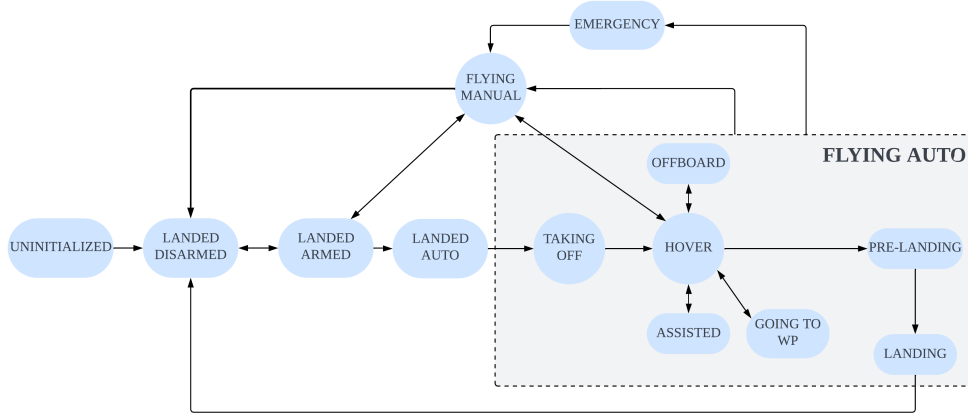


Fig. 2: Control Manager FSM states and allowed transitions.

9) *ASSISTED*: A velocity reference is received (for example, from a joystick). Before sending the trajectory to the controller, it is saturated to the defined maximum values.

State to Transit	Condition
HOVER	Hover is requested
FLYING MANUAL	Safety pilot revokes authority
EMERGENCY	Critical messages are not being received

10) *OFFBOARD*: In this mode, it is possible to send commands of any type to the controller, such as control in position, velocity, or a trajectory that defines position, velocity, and acceleration at the same time. This is the most unrestricted mode for controlling the UAV.

State to Transit	Condition
HOVER	Hover is requested
FLYING MANUAL	Safety pilot revokes authority
EMERGENCY	Critical messages are not being received

11) *PRELANDING*: A time-parametrized trajectory to reach a way-point with predefined pre-landing height and the current horizontal position is sent to the controller.

State to Transit	Condition
HOVER	Hover is requested
FLYING MANUAL	Safety pilot revokes authority
EMERGENCY	Critical messages are not being received
LANDING	Pre-landing height is reached

12) *LANDING*: A predefined negative vertical velocity is sent to the controller until land detection is triggered based on vertical velocity, accelerometer, and PWM information.

State to Transit	Condition
FLYING MANUAL	Safety pilot revokes authority
EMERGENCY	Critical messages are not being received
LANDED DISARMED	Aircraft is disarmed
LANDED ARMED	Land detection and not disarmed

13) *EMERGENCY*: Some critical message is not being received, so it stops acting on the controller and enters this state to warn the user and the safety pilot.

State to Transit	Condition
FLYING MANUAL	Safety pilot revokes authority

B. Implementation

The state machine is implemented in C++ using TinyFSM [21]. The states are classes derived from a base FSM state, providing *react()* functions for every event, as well as *entry()* and *exit()* functions. The *entry()* and *exit()* methods define what will take place every time the system reaches the state and when it leaves it, while the *react()* method handles the response in that state to an event of some kind occurring (e.g., service call, emergency...).

To this base class, we add the *executeThreadJob()* method, which will define the behavior of the UAV while it is in that state. In Fig. 3, this scheme can be seen, and an example of the implementation for the GOING TO WP state. In that case, several attributes are created to develop functionalities, such as the planner or timer. In addition, the *react()* methods are defined for the case when an emergency is detected or when hover mode is requested. With this base, other states could be developed in an easy and customizable way.

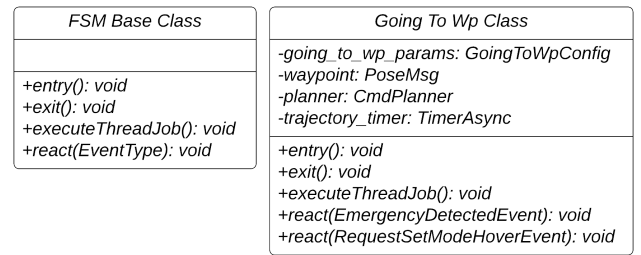


Fig. 3: Base State and Going To Wp Class diagrams.

Transition functions are implemented in TinyFSM. By including these in *react()* methods or in the main thread when a condition is met is how we implement transitions between states.

The class is designed to be communications-agnostic, allowing for easy integration with various middleware systems. It relies on C++ *std::functions* to send commands,

and it uses setters and getters for state and data management. Although the current implementation utilizes ROS or ROS2 nodes to instantiate the state machine, the design is not restricted to ROS. This flexibility ensures that the FSM Control Manager can fit into a wide range of communication ecosystems without modification to its core functionality.

As shown, this state machine can be connected to a multitude of different modules, including various types of autopilot firmware. Our communication scheme consists of using MAVROS, which transforms our ROS communications into MAVLINK [22] messages, which is the protocol that ArduPilot and PX4 softwares can understand. This also allows us to use their Software-In-The-Loop (SITL) implementations to be able to deploy the complete system in a realistic simulation, which helps us to test the more complex algorithms before implementing them in a real UAV, improving security in the research environment. Through MAVROS, it is possible to send commands to change the autopilot mode, which is necessary when the user requests the authority. If you want the autopilot to listen to commands through the companion computer, the mode needed for PX4 is OFFBOARD mode, while for ArduPilot, it is GUIDED mode.

Finally, to implement many of the behaviors that occur in the states, we have developed some important modules inside the Control Manager FSM code architecture.

- **Planner** module is a motion planning algorithm designed to generate smooth trajectories. It calculates a time-parameterized trajectory while enforcing kinematic constraints, including maximum velocity, acceleration, and deceleration. The module follows a three-phase motion profile consisting of acceleration, constant velocity, and deceleration, ensuring a dynamically feasible trajectory. This module is used by states that need to implement precise movement, like *TAKING OFF*, *GOING TO WP* and *PRELANDING*.
- **Landing detector** module, which is based on [23], detects the moment at which the drone has touched down not based on altitude, but using measurements of acceleration, vertical speed, and PWM value.
- A fast and lightweight **logger** based on spdlog [24].
- **Parameter handler** that allows the user to modify state-related aspects, such as the speed and acceleration of the UAV during motion states, or the time that must elapse without receiving a critical message to enter in the EMERGENCY state.

IV. CASCADE CONTROLLER

This module is the controller in the framework proposed in this work, fully interfacing with the state machine to perform the necessary behaviors. It can be used in three different modes:

- **Position Controller.** The reference is only a position and orientation. Used in mode *HOVER*.
- **Velocity Controller.** The reference is a linear velocity. Used in mode *ASSISTED* or *LANDING*.

- **Trajectory Controller.** The reference can be a trajectory that defines the position, velocity, and acceleration at the same time. Used in mode *GOING TO WP*, *TAKING OFF*, *PRELANDING* or *OFFBOARD*.

The controller output in all cases is the multi-rotor angle set-point, which goes directly to the attitude controller in the autopilot.

A. Controller Scheme

The horizontal controller is based on a cascade PID architecture. It consists of an outer loop that regulates the position of the UAV by generating a velocity set-point for the inner loop that regulates the velocity. Finally, the output of the inner loop is an acceleration set-point that is transformed to roll and pitch angles and sent to the attitude controller. This explanation is illustrated in Fig. 4.

The altitude controller and the yaw controller are simpler, as the inner loop will be executed directly on the autopilot. Fig. 5 shows this scheme.

It is possible to include velocity and acceleration references in the feed-forward terms to improve the response of the controller, leaving the feedback control to take care of the disturbances only.

B. Implementation

We continue to develop the code in C++, using the control toolbox package, belonging to the ros-controls framework [25], which contains some widely used control theory implementations, such as PIDs, smoothers, or noise generators. The use of these tools in the controller is really useful, especially because it facilitates logging and tuning task, we are able to change parameters in real time and see the effects in the performance of the controller.

In addition, we implement two versions of the ROS and ROS2 module. The use of this middleware not only facilitates the communication between the different modules of the framework and allows the complete integration with the Control Manager FSM, but also allows us to interact with MAVROS, which will be the node in charge of transmitting our commands to the autopilot software. This ROS package acts as a bridge between the MAVLink protocol and the Robot Operating System (ROS). MAVROS enables high-level control by publishing attitude setpoints to the `/mavros/setpoint_raw/attitude` topic using the `mavros_msgs/AttitudeTarget` message type. This message contains fields such as orientation (quaternion representation of roll, pitch, and yaw), thrust (normalized from 0 to 1), and body rate (angular velocity). The equivalent of this message in MAVLink is `SET_ATTITUDE_TARGET`. This architecture allows for precise drone maneuvering using software control on the companion computer, regardless of whether PX4 or ArduPilot is used.

V. EXPERIMENTAL EVALUATION

A. Simulation tests

Preliminary simulation tests can save a lot of time and money and facilitate the research and development

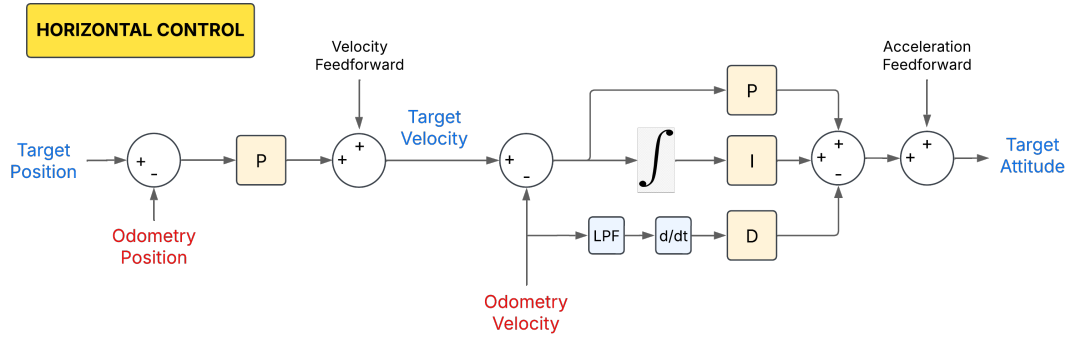


Fig. 4: Horizontal controller diagram.

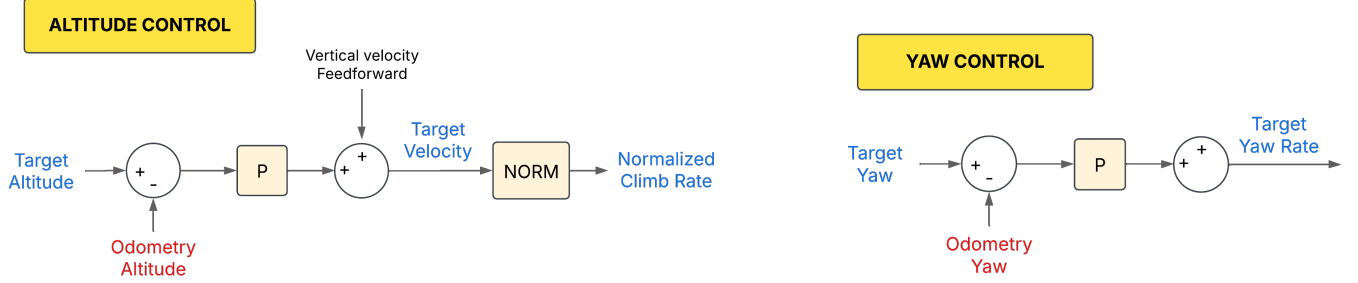


Fig. 5: Altitude and yaw controller schemes.

of applications on real UAVs. For this reason, we have integrated our framework with the SITL tool provided by ArduPilot. Not only can we simulate the autopilot software, but we can also use Gazebo [26] to simulate the platform dynamics, sensors, and also the mission environment. The simulated platform designed for this purpose is shown in Fig. 6a.

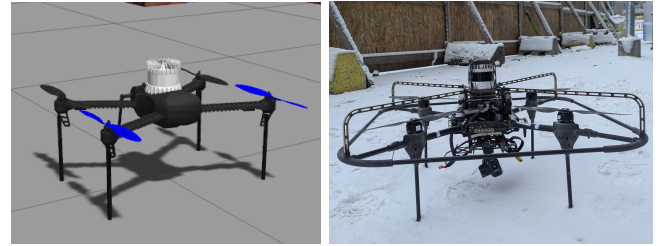
As our goal is to check the correct functioning of most of the modules that complete the complete system before a real mission, we use almost the same software packages as in Section V-B, as well as a simulated drone model with a LiDAR sensor that allows us to approximate the inputs that the algorithms will have.

The navigation module in the simulated environment is provided by the SITL. Since we need a real-time obstacle avoidance planner using point cloud, we choose a custom version of Fast-Planner [27] for this development, which is a motion planning framework designed for real-time autonomous navigation that generates obstacle-free paths. The obtained paths are refined using B-splines to produce dynamically feasible and efficient trajectories with position, velocity, and acceleration references, which are compatible with our Cascade Controller. Ouster-gazebo-simulation [28] is the plugin that emulates the LiDAR point cloud.

The mission consists of taking off, following a route avoiding walls, and landing. In Fig. 7, it can be checked how this is done successfully.

B. Real platform tests

The use case in which the system has been tested consists of the acquisition of different types of data from a building under construction (see Fig. 9), using a quad-rotor for



(a) Iris platform with a LiDAR. (b) CADRIN platform.

Fig. 6: Simulated and real platform.

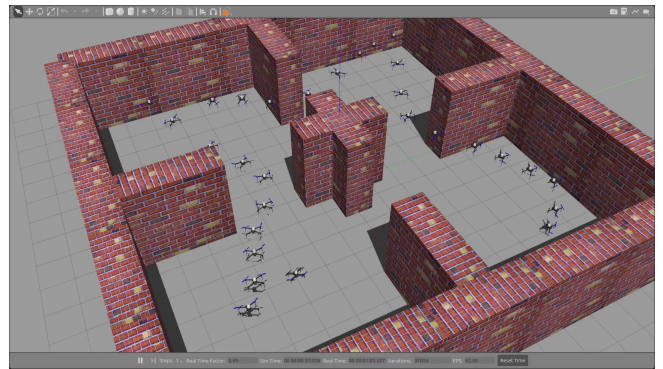


Fig. 7: Simulated mission footage from Gazebo.

indoor flight with on-board sensors: Ouster OS0 LiDAR, which is used to obtain point clouds of the rooms, and a Teledyne FLIR Hadron640 camera, which can gather visual and thermal data to monitor the progress of the work, as well as to ensure that the necessary safety measures are fulfilled. The CADRIN platform, developed for this use case, is shown

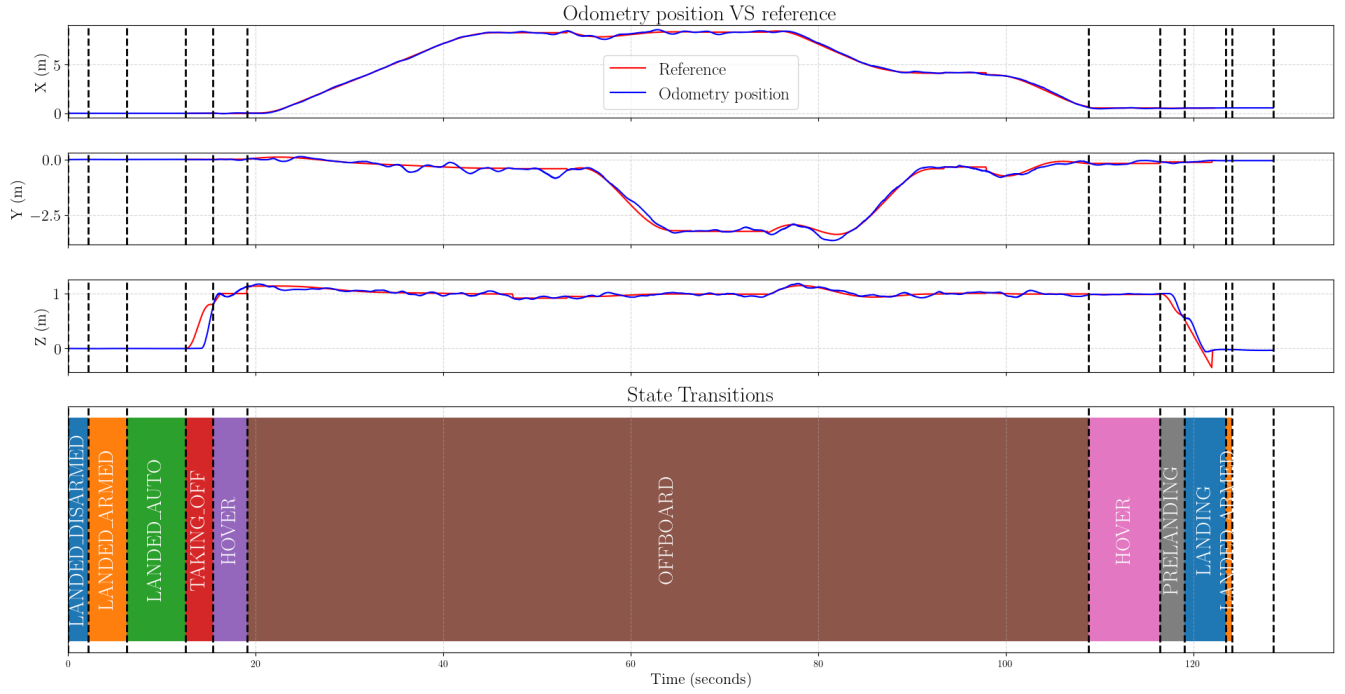


Fig. 8: Odometry vs reference and state transitions during the mission.

in Fig. 6b.

Indoor flight requires a navigation module that can estimate a robust position in a GNSS-denied environment using the on-board sensors. In this case, we opt for navigation software that fuses LiDAR information with the inertial sensor readings from the autopilot. This module is a custom version of LIO-SAM [29].

The planning module is the same as that in the simulation test, Fast-Planner. The Control Manager FSM is in charge of managing the mission states, and the Cascade Controller sent the commands to the autopilot, which on this platform consists of a CubePilot Cube Orange hardware module with ArduPilot software installed.

Several experiments have been conducted to test how the framework works in this use case. A representative one is presented in Fig. 8, where the trajectory proposed by the guidance module is properly tracked, along with the state changes that allow the development of the entire use case. The objective of the mission is to go through the way-points determined by the user in order to obtain the necessary data from the building under construction. First, the system is in *LANDED DISARMED* mode, where the algorithms are initialized and the drone is unarmed on the ground. The security pilot arm the drone, triggering the transition to *LANDED ARMED*. The user takes authority (*LANDED AUTO*) and initiates the take-off (*TAKING OFF*). After reaching the desired altitude, the UAV enters into the *HOVER* state, ready for new commands. While the guidance module sends it the necessary references to reach the way-points while avoiding obstacles, the system is in *OFFBOARD* mode. At the end of the mission, the aircraft goes through *HOVER* mode again, and the user starts the landing sequence, going

through *PRELANDING*, *LANDING* and finally *LANDED DISARMED* mode after completing the descent successfully.

We compared the maximum and average CPU usage of the Control Manager FSM a similar node that uses SMACH implemented in Python in Table I, with 100% being the total usage of one of the processor cores.

TABLE I: Average and maximum CPU usage for each node

	Avg CPU (%)	Max CPU (%)
Control Manager FSM	25.04	55.66
SMACH-based FSM	55.69	75.1



Fig. 9: Mission footage in construction scenario.

VI. CONCLUSION

In this work, we have introduced a framework designed to control and monitor an unmanned aerial vehicle (UAV) during fully autonomous missions. The framework ensures effective command execution throughout each phase of the mission, with the Control Manager FSM overseeing

the monitoring and management of trajectory set-points, while the Cascade Controller is responsible for generating commands for the low-level controller from autopilot.

Some key features of the system that have been highlighted in this article are: the separation of communication from implementation, facilitating integration through ROS and ROS2 middleware; the use of the TinyFSM library, which has allowed us to have a lower CPU usage compared to other software designed for the same purpose; and the possibility of creating new states using the base class as a reference.

To complement its functionalities, the system is integrated with open-source autopilot software such as PX4 and ArduPilot, in conjunction with state-of-the-art navigation and guidance modules.

Finally, we have validated the framework in both simulated environments using SITL and real-world scenarios, demonstrating its effectiveness in enabling a quad-rotor to navigate autonomously inside a building under construction.

ACKNOWLEDGMENTS

The experiments described in this paper would not have been possible without our safety pilots, Manuel Barbero and Francisco García. This work has been performed within the context of BEEYONDERS project, funded by the Horizon Europe research and innovation programme under grant agreement No 101058548, the SIMAR project, funded by the European Union's Horizon Europe research and innovation programme under grant agreement No 101070604, and the AEROSTREAM project, funded by Horizon Europe research and innovation programme under grant agreement No 101071270.

REFERENCES

- [1] J. Cacace, S. M. Orozco-Soto, A. Suarez, A. Caballero, M. Orsag, S. Bogdan, G. Vasiljevic, E. Ebeid, J. A. A. Rodriguez, and A. Ollero, "Safe local aerial manipulation for the installation of devices on power lines: Aerial-core first year results and designs," *Applied Sciences*, vol. 11, no. 13, p. 6220, 2021.
- [2] M. A. Montes-Grova, J. Ortuño-Conde, D. Tejero-Ruiz, J. Olmedo-Pradas, F. J. Pérez-Gran, M. A. Trujillo-Soto, and A. Viguria, "Petrochemical industry aerial robotic inspection: A novel concept for landing and deploying robots on pipes," in *2024 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2024, pp. 1377–1384.
- [3] F. Nex and F. Remondino, "Uav for 3d mapping applications: a review," *Applied geomatics*, vol. 6, pp. 1–15, 2014.
- [4] N. Anwar, M. A. Izhar, and F. A. Najam, "Construction monitoring and reporting using drones and unmanned aerial vehicles (uavs)," in *The Tenth International Conference on Construction in the 21st Century (CITC-10)*, vol. 8, no. 3, 2018, pp. 2–4.
- [5] R. Zahinos, H. Abaunza, J. Murillo, M. Trujillo, and A. Viguria, "Cooperative multi-uav system for surveillance and search&rescue operations over a mobile 5g node," in *2022 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2022, pp. 1016–1024.
- [6] T. Elmokadem and A. V. Savkin, "Towards fully autonomous uavs: A survey," *Sensors*, vol. 21, no. 18, p. 6223, 2021.
- [7] X. Liu, G. V. Nardari, F. Cladera, Y. Tao, A. Zhou, T. Donnelly, C. Qu, S. W. Chen, R. A. Romero, C. J. Taylor *et al.*, "Large-scale autonomous flight with real-time semantic slam under dense forest canopy," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 5512–5519, 2022.
- [8] KumarRobotics, "Kr autonomous flight," 2025, accessed: 2025-02-07. [Online]. Available: https://github.com/KumarRobotics/kr_autonomous_flight
- [9] M. Fernandez-Cortizas, M. Molina, P. Arias-Perez, R. Perez-Segui, D. Perez-Saura, and P. Campoy, "Aerostack2: A software framework for developing multi-robot aerial systems," *arXiv preprint arXiv:2303.18237*, 2023.
- [10] T. Baca, M. Petrlik, M. Vrba, V. Spurny, R. Penicka, D. Hert, and M. Saska, "The mrs uav system: Pushing the frontiers of reproducible research, real-world deployment, and education with autonomous unmanned aerial vehicles," *Journal of Intelligent & Robotic Systems*, vol. 102, no. 1, p. 26, 2021.
- [11] L. Meier, D. Honegger, and M. Pollefeys, "Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 6235–6240.
- [12] F. Real, A. Torres-González, P. Ramon-Soria, J. Capitán, and A. Ollero, "Unmanned aerial vehicle abstraction layer: An abstraction layer to operate unmanned aerial vehicles," *International Journal of Advanced Robotic Systems*, vol. 17, no. 4, p. 1729881420925011, 2020.
- [13] A. D. Team, "Ardupilot: The most advanced, full-featured, and reliable open-source autopilot software," <https://ardupilot.org>, accessed: 2025-01-17.
- [14] DJI, "Dji a3 flight controller," 2016, accessed: 2025-02-07. [Online]. Available: <https://www.dji.com/a3>
- [15] —, "Dji n3 flight controller," 2017, accessed: 2025-02-07. [Online]. Available: <https://www.dji.com/n3>
- [16] L.-Y. Lo, C. H. Yiu, Y. Tang, A.-S. Yang, B. Li, and C.-Y. Wen, "Dynamic object tracking on autonomous uav system for surveillance applications," *Sensors*, vol. 21, no. 23, p. 7888, 2021.
- [17] J. A. Millan-Romera, H. Perez-Leon, A. Castillejo-Calle, I. Maza, and A. Ollero, "Ros-magna, a ros-based framework for the definition and management of multi-uas cooperative missions," in *2019 international conference on unmanned aircraft systems (icuas)*. IEEE, 2019, pp. 1477–1486.
- [18] W. Garage and R. Community, "Smach: Task-level state machine library for robot operating system," <http://wiki.ros.org/smach>, 2010, accessed: YYYY-MM-DD.
- [19] M. G. Ocando, N. Certad, S. Alvarado, and Á. Terrones, "Autonomous 2d slam and 3d mapping of an environment using a single 2d lidar and ros," in *2017 Latin American robotics symposium (LARS) and 2017 Brazilian symposium on robotics (SBR)*. IEEE, 2017, pp. 1–6.
- [20] A. Chakrabarty, R. A. Morris, X. Bouyssounouse, and R. Hunt, "An integrated system for autonomous search and track with a small unmanned aerial vehicle," in *AIAA Information Systems-AIAA Infotech@ Aerospace*, 2017, p. 0671.
- [21] Digint, "tinyfsm," <https://github.com/digint/tinyfsm>, n.d., accessed: 2025-01-20.
- [22] M. D. Team, "Mavlink micro air vehicle communication protocol," n.d. [Online]. Available: <https://mavlink.io>
- [23] PX4 Autopilot User Guide, *Land Detector Configuration*, 2025, accessed: 2025-01-29. [Online]. Available: https://docs.px4.io/main/en/advanced_config/land_detector.html
- [24] A. Wojciech, "spdlog: A fast c++ logging library," 2017, accessed: 2025-02-05. [Online]. Available: <https://github.com/gabime/spdlog>
- [25] S. Chitta, E. Marder-Eppstein, W. Meeussen, V. Pradeep, A. Rodríguez Tsouroukdissian, J. Bohren, D. Coleman, B. Magyar, G. Raiola, M. Lüdtkke, and E. Fernández Perdomo, "ros_control: A generic and simple control framework for ros," *The Journal of Open Source Software*, 2017. [Online]. Available: <http://www.theoj.org/joss-papers/joss.00456/10.21105.joss.00456.pdf>
- [26] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2004, pp. 2149–2154.
- [27] B. Zhou, F. Gao, J. Pan, and S. Shen, "Robust real-time uav replanning using guided gradient-based optimization and topological paths," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 1208–1214.
- [28] Gepetto, "ouster-gazebo-simulation," <https://github.com/Gepetto/ouster-gazebo-simulation>, 2022, version 2.1.2.
- [29] T. Shan, B. Englot, D. Meyers, W. Wang, C. Ratti, and R. Daniela, "Lio-sam: Tightly-coupled lidar inertial odometry via smoothing and mapping," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 5135–5142.