Unmanned Systems, Vol. 10, No. 4 (2022) 355–367

DOI: 10.1142/S2301385022410035

UNMANNED SYSTEMS
https://www.worldscientific.com/worldscinet/us

Framework and Evaluation Methodology for Autonomous Drone Racing

Miguel Fernández-Cortizas*,‡, David Pérez-Saura*, Pablo Santamaría*,†, Javier Rodríguez-Vázquez*,†, Martín Molina*,†, Pascual Campoy*

*Computer Vision and Aerial Robotics Group (CVAR), Centre for Automation and Robotics (CAR), Universidad Politécnica de Madrid (UPM), Madrid, Spain

†Department of Artificial Intelligence, Universidad Politécnica de Madrid, Madrid, Spain

In recent years, autonomous drone races have become increasingly popular in the aerial robotics research community due to challenges in perception, localization, navigation, and control at high speeds, pushing forward the state-of-the-art every year. However, autonomous racing drones are still far from reaching human pilot performance, and a lot of research has to be done to accomplish that. In this work, a complete architecture system and an evaluation method for autonomous drone racing research are proposed, based on the Aerostack 5.0 open-source framework. To evaluate the performance of the whole system and of each algorithm used separately, this framework is validated not only with simulated flights, but also through real flights in an indoor drone race circuit using different configurations.

Keywords: Autonomous drone racing; UAV; framework.

US

1. Introduction

1.1. *Motivation*

Autonomous drones have been increasing their application in different tasks in recent years. The short flight time of a quadcopter limits its use in missions such as search and rescue. To take advantage of this, it is interesting to develop agile drones that can explore or traverse an area in a short period of time. Autonomous drone racing is a great environment to push agile drones to the limit. The high speeds and agile maneuvers required for this purpose increase the difficulty in locating, controlling, and generating trajectories. To test different techniques to get the best results, it is useful to work in a modulated environment that allows you to develop and validate different algorithms independently.

Received 24 January 2022; Accepted 18 April 2022; Published 15 June 2022. This paper was recommended for publication in its revised form by editorial board member, Jose Martinez-Carranza.
Email Address: ‡*miguel.fernandez.cortizas@upm.es*

1.2. *Related work*

Over the years, many different techniques have been developed to get the best results in Autonomous Drone Racing. At the beginning, the speeds achieved in competition were considerably low. In ADR 2017, a combination of a monocular SLAM localization algorithm and a PID control won the competition with a mean speed of 0.7 m/s [1]. For IROS 2018 ADR, the winners used machine learning techniques for gate detection with an MPC controller. They also improved the trajectories by adding two points to the path, one before and one after each gate, achieving flight speeds near 2 m/s [2].

In 2019, there was a great improvement in the speed with which autonomous drones could fly. For the first AlphaPilot competition, a novel architecture was developed [3]. In this architecture, machine learning techniques are combined with a nonlinear filter for sensor detection and time-optimal trajectory planning. Using a PD controller, they reached a maximum flight speed of 8 m/s. In the same year, the winners of the first simulated drone race, NeurIPS 2019

Game of Drones, developed a controller using reinforcement learning techniques [4], achieving a maximum speed of 16 m/s in simulation.

On the other hand, there are many simulation environments to test the different approaches. Moreover, many competitions have been held on these simulators. Flightgoggles [5] is a photorealistic simulator used for AlphaPilot 2019. Some of them have some APIs for a specific development. Flightmare [6] has an API for reinforcement learning. AirSim Drone Racing Lab [7] is a Microsoft framework with some APIs that allows you to focus your test on each of the different research directions in autonomous drone racing. However, there are not many frameworks that combine state-of-the-art algorithms with simulators to work as baseline repositories for autonomous drone racing research.

1.3. *Contribution*

In this paper, we present a modular framework for developing and validating new algorithms to improve agile drone flying using autonomous drone races as a perfect test environment. This framework provides a modular system architecture with some state-of-the-art algorithms that allow researchers to concentrate efforts on improving one of the fields related to drone behavior, without needing to build a whole system by themselves. Furthermore, a simulation environment based on gazebo is provided to test the algorithms before jumping into real experiments. For real experiments, we decided to use Pixhawk as the autopilot to ease the use of this framework through the research community. Finally, we propose a set of metrics to measure the performance of some modules separately and the performance of the whole system to be able to compare different algorithms easily.

2. System Architecture

The system architecture presented in this work has been developed using the Aerostack framework [8], an open source multipurpose software framework for the development of autonomous multirobot unmanned aerial systems created by the Computer Vision and Aerial Robotics (CVAR) group. The Aerostack modules are mainly implemented in C++ and Python languages and are based on Robot Operating System (ROS) [9] for intercommunication between the different components; we refer the reader to the extensive documentation and publications available on its website.[a]

[a] www.aerostack.org.

Using the Aerostack software framework, a new design of system architecture was developed for the tasks presented in this work. It has to be noted that, although the Aerostack framework is able to provide predefined components and intercommunication methods to provide autonomy to UAVs, the components described in this work were completely designed and developed for the objectives described in this paper. Figure 1 shows the functionalities that have been implemented in this work. In the figure, colored rectangular boxes represent data processing units (or processes in short) that are implemented as ROS nodes. They are organized in the following main components:

- *Sensor-actuator interfaces:* to receive data from sensors on the aerial platform and send commands to robot actuators.
- *Communication channel:* Based on the Aerostack framework, our architecture uses a common communication channel that contains shared dynamic information between processes. This channel facilitates process interoperability and helps to reuse components across different types of aerial platforms. The channel is implemented with a set of ROS topics and ROS messages.
- *Robot behaviors:* The system architecture partially follows the behavior-based paradigm in robotics. According to this paradigm, the global control is divided into a set of behavior controllers, and each one is in charge of a specific control aspect separately from the other behavior controllers. For example, robot behaviors implement functional capabilities such as motion control, feature extraction, state estimation, and navigation.
- *Mission Control:* Mission control executes a mission plan specified in a formal description. In this implementation, the mission plan is specified using the Python language using a set of prefixed functions to start and stop robot tasks.
- *Behavior Coordinator:* A behavior coordinator is used to translate planned tasks into consistent activations of robot behaviors. This component has been implemented following a constraint-based configuration approach using the software tool called Behavior Coordinator CBC [10].

The following sections, describe in more detail the components related to robot behaviors and mission control.

3. Robot Behaviors

A behavior defines a basic functionality of a system, such as moving to a point, moving an actuator, activating a sensor, and includes the three following principles:

- *Common data channel:* Each behavior controller should be able to execute separately assuming that the required input data is available in the common data channel.

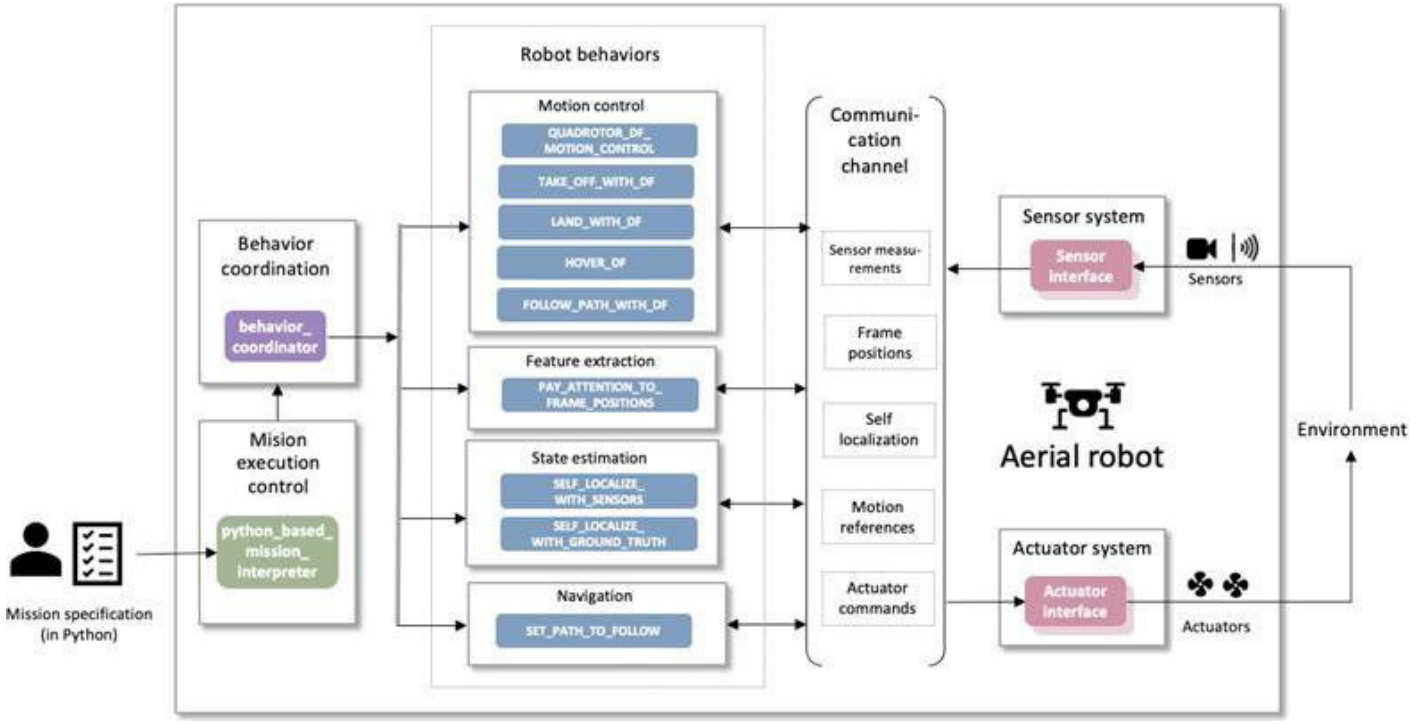


Fig. 1. System architecture.

- *Activation management:* Each behavior is programmed with an activation management mechanism which handles how to start and stop the execution of the behavior controller.
- *Execution monitoring:* Execution monitoring is a kind of self-awareness computing process by which the robot observes and judges its own behavior. This includes possible behavior termination states such as GOAL_ACHIEVED, WRONG_PROGRESS or PROCESS_FAILURE.

The system architecture includes behaviors that can be divided in the categories explained in the following sections. Robot behaviors have been programmed with the help of the behaviorlib [11] library that includes execution management functions. This library is open-source and provides tools for building, executing, and monitoring behaviors.

3.1. *Motion control*

All the set of behaviors that are responsible for controlling the movement of the drone. This category includes simple behaviors such as take-off, hovering, or landing, as well as more complex behaviors such as generating a trajectory and making the drone follow it.

In order to perform aggressive maneuvers on a quadrotor, it is necessary to employ a non-linearized controller. We have implemented a quadrotor control algorithm based on differential flatness and the corresponding behaviors inspired by the work of Mellinger and Kumar [12] with several modifications, so the output of the controller corresponds to angular velocity references and the desired collective thrust of all motors. The input of this controller consists in the position, speed, and acceleration references provided by a trajectory generator.

Due to the need to generate trajectories for the controller, a polynomial trajectory generator [12–14] has been used. The trajectories are generated on a set of waypoints and are optimal in acceleration, which guarantees smoothness in the actuator commands. Moreover, the generated trajectories are limited by the maximum speed $v_{\max}$ and maximum acceleration $a_{\max}$ parameters.

3.2. *State estimation*

In order to achieve the level of control required for this project, it is necessary to provide the best state estimation possible to all components involved. Aerostack provides multiple components for this task, obtaining data from multiple sensors such as cameras (for relative positioning to a certain object), IMU, laser sensors, depth cameras, or lidar.

Some of these sensors or estimators should not always be trusted, as cumulative errors can and will happen, so it is necessary to combine the feedback of several of them using

sensor fusion approaches, such as the multisensor fusion algorithm developed by Lynen *et al.* [15] to achieve the most accurate and reliable state estimation possible in every situation.

However, for real flights, we decided to use the state estimation provided by an Intel Realsense T265 Tracking Module, due to the ease of use and the reduction of the computational load of the on-board computer.

3.3. *Perception*

Perception is a key problem in the proposed scenario, since UAVs need to detect and locate each of the gates accurately to be able to complete the entire track.

Since the detection and pose estimation of the gates is a difficult problem itself, to evaluate the rest of the proposed framework components without the influence of possible errors in perception, we placed ArUco fiducial markers [16] in each gate to estimate the relative position to the UAV's main camera. Each marker also encodes a unique gate ID, allowing us to design arbitrarily complex tracks. For the detection of such markers, we use OpenCV [17] implementation of the method proposed in [18].

3.4. *Navigation*

When the gates are located in the real world, it is necessary to plan the path that the aircraft must follow to pass through them and avoid other obstacles. For this approach, we considered two scenarios:

- **Lack of knowledge of gate positions.** The aircraft does not have any information about where the gates are located, so it must begin looking for them in order to generate the waypoints to pass through the gates. As long as the aircraft passes through the circuit, new gates will be in sight, so the quadrotor can add these gates to its route. In this scenario, the aircraft is always considered to see at least the following gate.
- **Gate position awareness.** The aircraft knows an approximate position of each gate of the circuit, so it can generate complete trajectories through the whole circuit that must be corrected with as long as the gates are in sight, so it has to update the initial gate positions to pass through the circuit. This is how the majority of the autonomous drone racing competitions work.

In both approaches, each gate center is treated as a waypoint in a path, and this waypoint position is updated as the quadrotor flies through the circuit. Whenever the quadrotor passes through one gate, this gate center is removed from the path. This path is sent to the trajectory generator, which takes care of commanding the quadrotor to pass through the gates.

4. Experiments

To validate the proposed system architecture and the suitability of the different algorithms selected, several experiments have been carried out, not only in simulation, but also in real life. For analyzing this performance, several metrics have been used:

- **State Estimation error:** For measuring the accuracy of the state estimation algorithm, we compute the Root Mean Squared Error (RMSE) between the estimated state and the ground truth.
- **Trajectory following error:** To evaluate the performance of the proposed controller, we decided to measure the trajectory following error, calculated with the RMSE between the trajectory sent by the trajectory generator and the real trajectory followed by the aircraft.
- **Speeds:** In autonomous racing, other metrics such as the maximum speed reached, the medium speed, or the elapsed time to go through the whole circuit must be taken into consideration.

The metrics obtained in the different experiments were automatically obtained using an evaluation script on the raw data recorded during the flights.

4.1. *Aerial vehicle platform*

The aerial platform used for the real experiments was a custom quadrotor based on the DJI F330 frame, shown in Fig. 2. This platform was equipped with a Pixhawk 4 mini as the aircraft autopilot, an Intel Realsense T265 Tracking Module used for state estimation, and a USB fish-eye camera for gate detection.

Additionally, the aerial platform was equipped with a Single Board Computer (SBC) NVIDIA Jetson Xavier NX with

Fig. 2. Quadrotor used for real flight experiments.

a 6-core ARM v8.2, 64-bit CPU running Ubuntu Linux 18.04 Bionic Beaver for on-board computing. All computations required for the real experiments occurred in this SBC.

In order to obtain the ground truth position of the aircraft during some experiments, a Motion Capture System (*mocap*) was used. For localizing the aircraft inside the *mocap* area, several IR markers have been attached to the platform.

4.2. *Control experiments*

The first experiment was intended to compare the behavior of a drone flying in a simulation environment with a real experiment. In this experiment, the aircraft had to pass through some points, which are placed forming a lemniscate trajectory, with three different values for $v_{\max}$ parameter: 3, 5 and 7 m/s.

4.2.1. *Simulated flight*

All the simulation experiments where performed using the Gazebo simulator [19] and the *iris* drone as a simulated quadrotor. We decided to use the gazebo simulator because of its ease of use and large community.

We flew three laps through the circuit, increasing the speed configuration during each lap. The trajectory was generated by the trajectory generator and the state of the aircraft was provided by the simulator. Comparisons between both are shown in Figs. 3(a) and 3(b), and trajectory following error obtained in the different laps in Table 1.

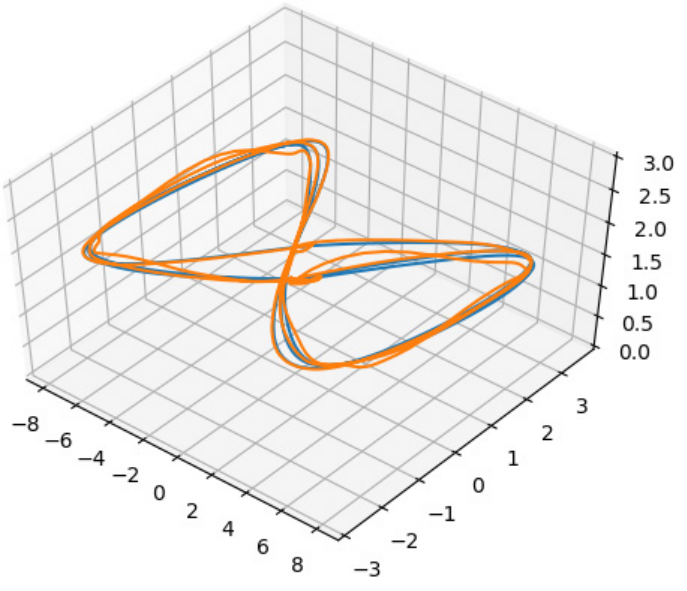
To obtain a deeper analysis of the controller, a comparison was made between the speed indicated by the trajectory generator and the speed realized by the controller. This comparison can be seen in Fig. 3(c) and the RMSE calculated for each lap is shown in Table 2.

Finally, Fig. 3(d) shows the speed performed by the drone during the flight, while Table 3 shows the speed metrics and the time elapsed for each lap.

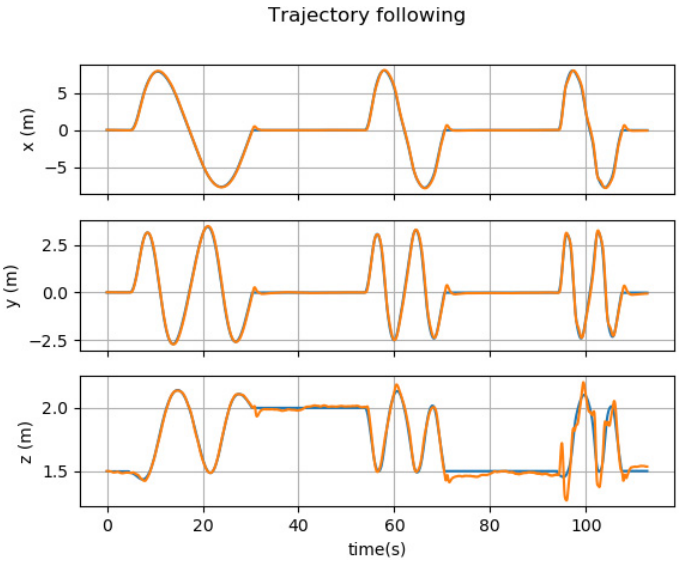
4.2.2. *Real flight*

Having demonstrated that the controller works in simulation, the next step was to test how it works in a real aircraft. The experiment consisted of trying to repeat the same trajectories generated in the previous experiment, in a real environment. For these experiments, the aircraft state estimation was provided by an Intel Realsense T265 tracking camera.

Equivalent figures and tables are shown as for the previous experiment. See Figs. 4(a), 4(b) and Table 4 for trajectory following, Fig. 4(c) and Table 5 for speed analysis, and Fig. 4(d) and Table 6 for speed performance.

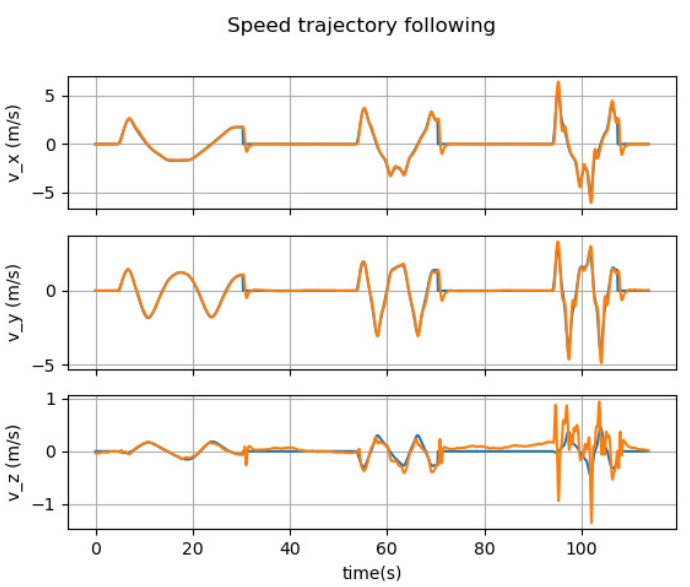


(a) Path followed (orange) by the simulated aircraft compared to the trajectory generated (blue) for the motion control behavior, with different maximum speeds (3, 5 and 7 m/s).

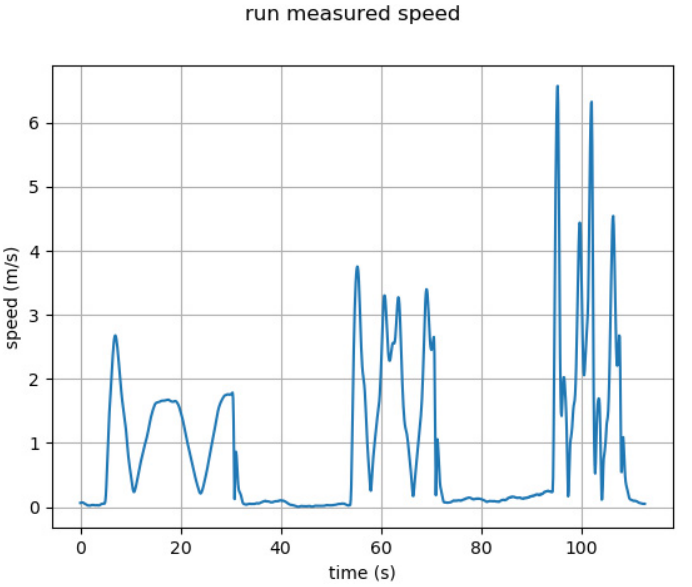


(b) Path followed (orange) by the simulated aircraft compared to the trajectory generated (blue) for the motion control behavior.

Fig. 3. Results for control experiment in a simulated environment.



(c) Speed performed (orange) by the simulated aircraft compared to the speed of trajectory generated (blue) for the motion control behavior.



(d) Speed performed by the simulated drone during the experiment.

Fig. 3. (*Continued*)

Table 1. RMSE between trajectory reference and estimator measurements, expressed in meters, when the simulated quadrotor pass through the whole circuit with trajectories generated with different values of v_{max} parameter.

	x-axis	*y*-axis	*z*-axis	Total
$v_{max} = 3.0$	0.1260	0.0893	0.0125	0.1381
$v_{max} = 5.0$	0.2120	0.1372	0.0322	0.2249
$v_{max} = 7.0$	0.2781	0.1585	0.0971	0.3067

Table 2. RMSE between speed reference and estimator measurements, expressed in meters, when the simulated quadrotor pass through the whole circuit with trajectories generated with different values of v_{max} parameter.

	x-axis	*y*-axis	*z*-axis	Total
$v_{max} = 3.0$	0.2033	0.1287	0.0325	0.1077
$v_{max} = 5.0$	0.3958	0.2444	0.0860	0.2584
$v_{max} = 7.0$	0.6257	0.4431	0.3403	0.6931

Table 3. Performance metrics of the simulated drone with different values of v_{max} parameter.

	V_{max} (m/s)	V_{avg} (m/s)	Elapsed time (s)
$v_{max} = 3.0$	2.6797	1.1767	27.7
$v_{max} = 5.0$	3.7540	1.7092	19.7
$v_{max} = 7.0$	6.5750	2.3254	14.7

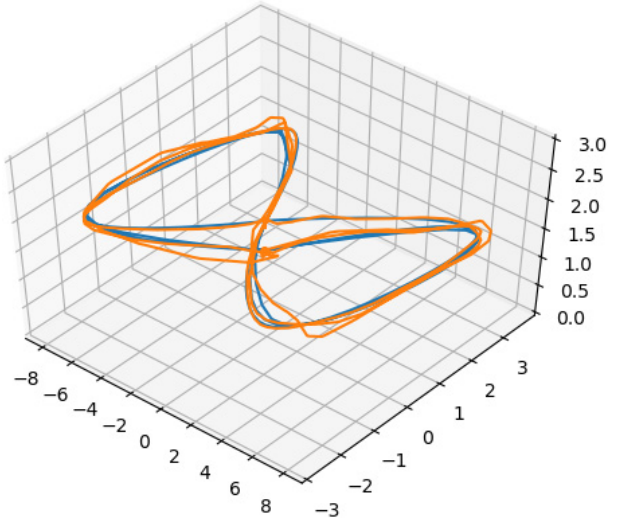
4.3. *Drone racing experiments*

In these experiments, the capabilities of the complete system were tested to run different drone racing circuits autonomously, first in a simulated environment and finally in a real one.

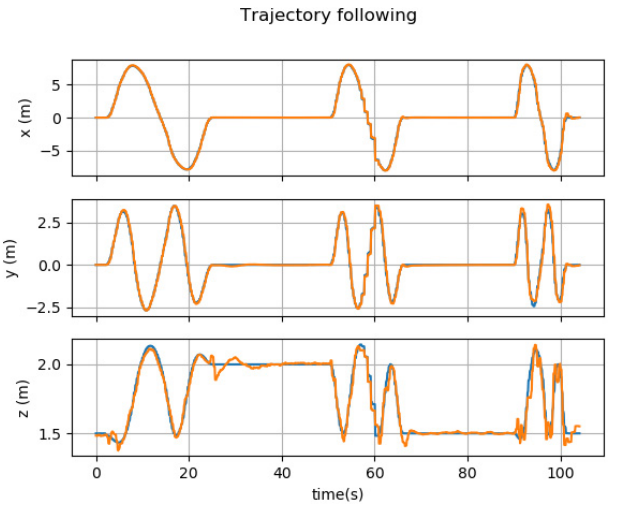
4.3.1. *Simulated flights*

In the simulation environment, we generate a five gates circuit distributed along a $25 \times 20 \times 3.5$ m area; see Fig. 5. The position of each gate was known with an uncertainty of 3 m, forcing the system to recalculate the positions of the gates to avoid crashing into them.

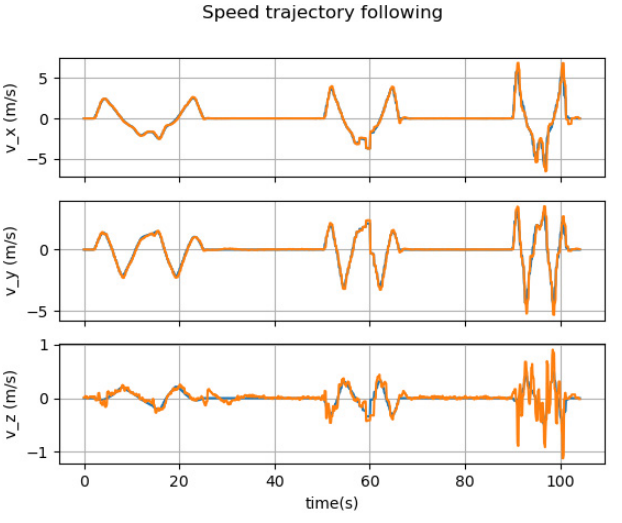
We flew through the circuit multiple times with four different speed configurations during each flight, as we can



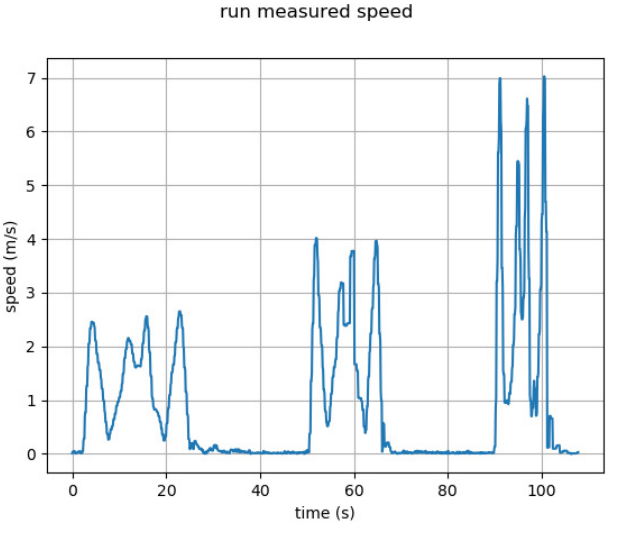
(a) Path followed (orange) by the real aircraft compared to the trajectory generated (blue) for the motion control behavior, each lap with different maximum speeds (3, 5 and 7 m/s).



(b) Path followed (orange) by the real aircraft compared to the trajectory generated (blue) for the motion control behavior.



(c) Speed performed (orange) by the real aircraft compared to the speed of trajectory generated (blue) for the motion control behavior.



(d) Speed performed by the real drone during the experiment.

Fig. 4. Results for control experiment in a real environment.

Table 4. RMSE between trajectory reference and estimator measurements, expressed in meters, when the real quadrotor pass through the whole circuit with trajectories generated with different values of $v_{\max}$ parameter.

	x-axis	*y*-axis	*z*-axis	Total
$v_{\max} = 3.0$	0.0657	0.0867	0.0258	0.0998
$v_{\max} = 5.0$	0.1112	0.1217	0.0430	0.1509
$v_{\max} = 7.0$	0.2857	0.2263	0.0771	0.3352

Table 5. RMSE between speed reference and estimator measurements, expressed in meters, when the real quadrotor pass through the whole circuit with trajectories generated with different values of $v_{\max}$ parameter.

	x-axis	*y*-axis	*z*-axis	Total
$v_{\max} = 3.0$	0.0701	0.0912	0.0477	0.1098
$v_{\max} = 5.0$	0.1618	0.1462	0.1070	0.2075
$v_{\max} = 7.0$	0.6929	0.4625	0.2943	0.7263

Table 6. Performance metrics of the real drone with different values of v_{max} parameter.

	V_{max} (m/s)	V_{avg} (m/s)	Elapsed time (s)
$v_{max} = 3.0$	2.6541	1.2921	24.6
$v_{max} = 5.0$	4.0185	1.6612	19.7
$v_{max} = 7.0$	7.0243	2.4446	14.1

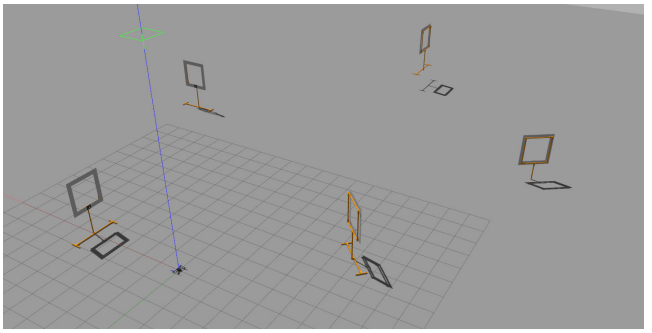


Fig. 5. Gazebo environment used for simulated race track experiments.

see in Fig. 6. In these experiments, the state of the aircraft was provided by the simulator, so the state estimation error was not calculated. All other metrics were obtained at different speeds; see Tables 7 and 8.

4.3.2. *Real flights*

Due to space limitations in the *mocap* area we decided to do two different experiments: in the first one we made the drone pass through one gate with different speeds to evaluate the state estimation error and the trajectory following error of the controller, in the second one the aircraft had to pass through a small circuit with 4 gates to test the performance of the whole system in a complex task.

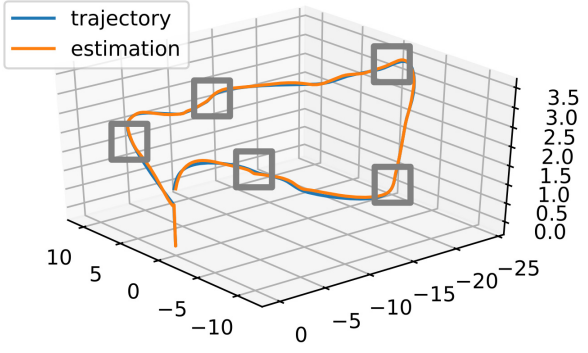


Fig. 6. Path followed by the aircraft when passing through the simulated circuit with $v_{max} = 4.0$ (m/s) compared to the trajectory generated for the motion control behavior.

Table 7. RMSE between trajectory reference and estimator measurements, expressed in meters, when the simulated quadrotor pass through the whole circuit with trajectories generated with different values of v_{max} parameter.

	x-axis	*y*-axis	*z*-axis	Total
$v_{max} = 1.0$	0.0552	0.0572	0.0602	0.0807
$v_{max} = 2.0$	0.0647	0.0831	0.0734	0.1168
$v_{max} = 3.0$	0.0959	0.0963	0.0924	0.1468
$v_{max} = 4.0$	0.1001	0.1103	0.0952	0.1583

Table 8. Speeds (m/s) achieve during flights and elapsed time to complete the whole simulated circuit employing different values of v_{max} parameter in the trajectory generation.

	V_{max}	V_{avg}	Elapsed time (s)
$v_{max} = 1.0$	0.8192	0.3848	127.6
$v_{max} = 2.0$	1.5414	0.6896	61.1
$v_{max} = 3.0$	2.7687	1.1828	38.8
$v_{max} = 4.0$	3.2657	1.2243	34.5

One gate crossing

For these experiments, the aircraft had to locate a gate located in $\text{gate}_{\text{position}} = [2.0, 0.0, 1.5]$ (m) and pass through it with 4 different max speed $v_{max} = \{1.0, 2.0, 3.0, 4.0\}$ (m/s) values for the trajectory generation, see Fig. 7.

To measure the estimator error provided by the Realsense T265 Tracking Module when the quadrotor flies at different speeds we used *mocap* system for making a comparison between the ground truth and the estimated pose of the quadrotor; see Table 9.

Before flying through a more complex circuit, it was convenient to measure the trajectory following error of the

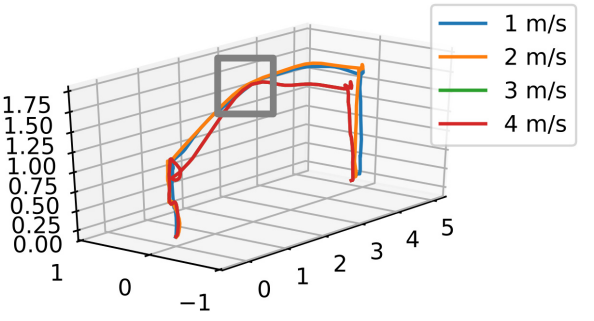


Fig. 7. Path followed by the aircraft when passing through the gate at different speeds. Position measurements had been obtained from the *mocap* system.

Table 9. RMSE between Realsense estimation and ground truth measurements, expressed in meters, when the quadrotor flies through trajectories generated with different values of v_{max} parameter.

	x-axis	*y*-axis	*z*-axis	Total
$v_{max} = 1.0$	0.1140	0.0248	0.0990	0.1236
$v_{max} = 2.0$	0.1178	0.0173	0.1475	0.1561
$v_{max} = 3.0$	0.1079	0.0283	0.2313	0.2468
$v_{max} = 4.0$	0.1019	0.0545	0.3359	0.3474

Table 10. RMSE between trajectory reference and estimator measurements, expressed in meters, when the quadrotor flies through trajectories generated with different values of v_{max} parameter.

	x-axis	*y*-axis	*z*-axis	Total
$v_{max} = 1.0$	0.0360	0.0262	0.0494	0.0558
$v_{max} = 2.0$	0.0683	0.0294	0.0633	0.0757
$v_{max} = 3.0$	0.1229	0.0385	0.0691	0.0948
$v_{max} = 4.0$	0.1699	0.0393	0.0820	0.0980

controller used when the aircraft flies at different speeds; see Table 10.

To evaluate the performance of the system passing through a drone racing circuit, other measures such as the time elapsed to complete the circuit, the average flying speed, and the peak speed are needed; see Table 11.

Four gates circuit crossing

After validating the proper work of the entire system in the previous experiments, the last experiments consisted of flying through a drone racing circuit with four gates arranged in the middle of the sides of a square of dimension 5×4 meters, with different heights for each. We flew through the circuit with two speed configurations: $v_{max} = 0.5$ and $v_{max} = 1.0$, as we can see in Figs. 8 and 9, respectively.

Table 11. Speeds (m/s) achieve during flights and elapsed time to complete the trajectory employing different values of v_{max} parameter in the trajectory generation.

	V_{max}	V_{avg}	Elapsed time (s)
$v_{max} = 1.0$	0.9670	0.5625	9.4
$v_{max} = 2.0$	2.2556	0.9770	5.5
$v_{max} = 3.0$	3.1249	1.2059	4.7
$v_{max} = 4.0$	4.1673	1.5334	3.9

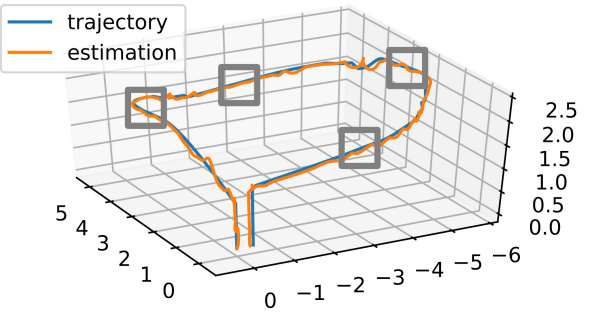


Fig. 8. Path followed by the aircraft when passing through the 4 gates circuit with $v_{max} = 0.5$ (m/s) compared to the trajectory generated for the motion control behavior.

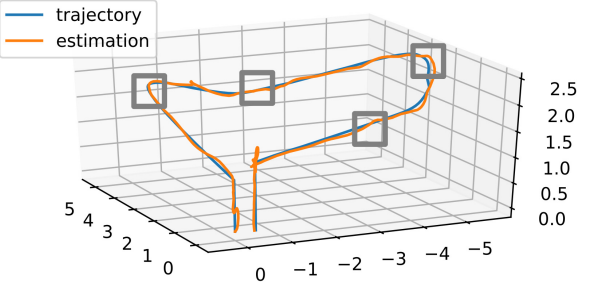


Fig. 9. Path followed by the aircraft when passing through the 4 gates circuit with $v_{max} = 1.0$ (m/s) compared to the trajectory generated for the motion control behavior.

Due to the space limitation of the arena, the ground truth poses were not acquired. However, the trajectory following errors of the controller and the time and speed metrics have been taken to compare both flights; see Tables 12 and 13.

4.4. *Profiling*

Since the resources of the on-board computer of a drone are limited, it is essential to optimize the computation time of the system. Therefore, it is very useful to have tools to measure the execution times of different modules and to be able to compare the execution performance between simulation and real experiments. In our framework, we use the

Table 12. RMSE between trajectory reference and estimator measurements, expressed in meters, when the quadrotor pass through the whole circuit with trajectories generated with different values of v_{max} parameter.

	x-axis	*y*-axis	*z*-axis	Total
$v_{max} = 0.5$	0.0492	0.0701	0.0576	0.0855
$v_{max} = 1.0$	0.0922	0.1491	0.1258	0.1414

Table 13. Speeds (m/s) achieve during flights and elapsed time to complete the whole circuit employing different values of v_{max} parameter in the trajectory generation.

	V_{max}	V_{avg}	Elapsed time (s)
$v_{max} = 0.5$	0.5316	0.2060	79.1
$v_{max} = 1.0$	0.9231	0.4125	32.9

Table 14. Time metrics of the different modules of the framework executed 10 times on both a high-performance computer and a on-board computer.

	Computer	Jetson Xavier NX
Controller	62.30 ± 0.93 ns	219.00 ± 3.85 ns
TrajGen (6 points)	13.71 ± 0.14 ms	51.47 ± 1.57 ms
TrajGen (14 points)	88.93 ± 3.11 ms	321.07 ± 22.11 ms
TrajGen (26 points)	309.08 ± 10.20 ms	1027.76 ± 19.36 ms
Aruco estimation	15.24 ± 0.12 ms	94.16 ± 2.03 ms

Google benchmark[b] profiling tools to standardize the time measurement process between different packages.

We decided to evaluate the execution time of three of the main components of the system:

- **Control loop:** It is a vital module for the flight of the drone, converting the received references into drone movements. To keep the drone in the air, it has to be continuously working, hence the time should be as short as possible.
- **Trajectory generator:** This module has to generate different trajectories when receiving new waypoints. The execution time of this module affects the reaction time of the aircraft, allowing it to change trajectory and avoid obstacles. Since the execution time of the trajectory generator depends on the number of waypoints, we have calculated the times for three different sets of points.
- **Aruco detection and pose estimation:** This module works for the drone racing application to detect the location of the gates. As in the previous module, the run time is critical to have enough reaction during the flight to pass through them and not collide.

We did not analyze other modules such as the state estimator because we use an external module, so it is completely-dependent on the frequency of the tracking camera or the navigation module because its execution time depends mainly on the circumstances of the environment.

Execution time also depends on the capabilities of the computer where they are executed. Therefore, each experiment was run both on a high-performance computer Intel®Core™ i7-10870H CPU @ 2.20 GHz running Ubuntu Linux 20.04 Focal Fossa and on a SBC NVIDIA Jetson Xavier NX with a 6-core ARM v8.2, 64-bit CPU running Ubuntu Linux 18.04 Bionic Beaver. The execution time of each module analyzed is shown in Table 14.

[b] https://github.com/google/benchmark.

5. Results Discussion

5.1. *Control experiments*

When comparing the performance in simulation and in real flight, the behavior is quite similar in both. The controller manages to track the references, and the trajectory following error and velocity error increase as the maximum speed increases. At maximum speed, the RMSE reaches values about 30 cm in trajectory tracking and about 70 cm/s in velocity tracking, which means that the system has been pushed to the limit, making this speed too high to test the whole system. We can appreciate that the real drone is a bit more powerful than the simulated aircraft, so the real drone reached higher speeds. In the simulation, the drone did not reach the maximum speed established for the trajectory, achieving a maximum speed of 6.57 m/s, while real drone reached a speed of 7.02 m/s. Therefore, the simulation environment available provides us with information close to the real results, becoming a very useful tool to test the experiments before moving on to real flights.

5.2. *Drone racing experiments*

On the simulated flights, the system was able to complete the whole circuit at different speeds, reaching speeds up to 3.2 m/s, with the trajectory following average errors of around 15 centimeters, which validates the operation of the entire system. Real experiments show that the proposed framework allows a real quadrotor to fly through drone racing circuit gates at speeds up to 4 m/s with a small increase in the trajectory following error compared with the simulated runs. The state estimator sensor can reach average estimation errors higher than 35 cm, adding this error to the trajectory following error could make the quadrotor collide with the gates if their positioning were not updated with respect to the drone position as long as the quadrotor navigates through the circuit. However, the limited computing capabilities of the SBC make trajectory generation spend a lot of time, which worsens the performance of the system when flying at high speeds through multiple gates. That made it not possible to complete such a small circuit at speeds higher than 1 m/s. To increase the speed in this experiment, it is necessary to find a larger track to have enough reaction time or to improve the processing speed.

5.3. *Profiling experiments*

Comparing the execution time on a high-performance computer and on an SBC, we can see that the execution time is between 3 and 4 times higher in this last one. This difference produces a different behavior of the system in real-time due to the reaction time of the aircraft. For example, execution times of some tenths of a second to generate a trajectory during a high-speed flight may cause the drone to collide before the new trajectory is established. As mentioned above, this is the reason for the limitation of the maximum speed in the previous experiments.

6. Conclusions and Future Work

In this work, a modular framework for autonomous drone racing has been proposed. This modularity makes it possible to isolate and interchange each module, allowing them to be modified and evaluated separately. This framework has a simulation environment to test the system behavior before moving to real experiments and also uses profiling tools to measure processing times, allowing us to adapt the modules to the requirements of different platforms and to compare their performance.

This framework has been validated through several experiments, not only in simulation but also in real environments, being able to cross gates up to 4.16 m/s and to go through a small circuit successfully at lower speeds. The state-of-the-art algorithms proposed for each module, combined with the evaluation metrics proposed, constitute a baseline for research on improving autonomous agile drone flying.

To improve this framework, a wider range of possibilities to choose for each module must be given, such as adding Model Predictive Controllers (MPCs), learning-based gate estimation methods, or a Visual Inertial Odometry estimator fused with other sensors to improve the state estimation. Moreover, the domain gap between simulation and real life is substantial when non-photorealistic simulators are used. Using a simulator like Flightmare [6] would help to develop and test algorithms based on images taken during flight.

Currently, ROS is the basis for intermodule communication. Preparing for the future, due to the growing interest of the community with more distributed systems, adapting the framework to ROS2 may be an option to consider.

Acknowledgments

This work has been supported by the project COMCISE RTI2018-100847-B-C21, funded by the Spanish Ministry of Science, Innovation and Universities (MCIU/AEI/FEDER, UE). The work of the first author is supported by the Research Assistant and Laboratory Technicians Program of the Science, University and Innovation Department of Madrid Government (partially funded by FEDER/EU). The work of the fourth author is supported Pre- and Post-Doctoral Program of the Science, University and Innovation Department of Madrid Government (partially funded by FEDER/EU).

References

[1] H. Moon, J. Martinez-Carranza, T. Cieslewski, M. Faessler, D. Falanga, A. Simovic, D. Scaramuzza, S. Li, M. Ozo, C. De Wagter, G. Croon, S. Hwang, S. Jung, H. Shim, H. Kim, M. Park, T.-C. Au and S.-J. Kim, Challenges and implemented technologies used in autonomous drone racing, *Intell. Serv. Robot.* **12** (2019) 137–148.

[2] E. Kaufmann, M. Gehrig, P. Foehn, R. Ranftl, A. Dosovitskiy, V. Koltun and D. Scaramuzza, Beauty and the beast: Optimal methods meet learning for drone racing, in *2019 Int. Conf. Robotics and Automation (ICRA)* (IEEE, 2019), pp. 690–696.

[3] P. Foehn, D. Brescianini, E. Kaufmann, T. Cieslewski, M. Gehrig, M. Muglikar and D. Scaramuzza, Alphapilot: Autonomous drone racing, *Autonomous Robots* **46** (2022) 307–320.

[4] S. Y. Shin Y.-W. Kang and Y. Kim, Report for game of drones: A neurips 2019 competition (2019), https://microsoft.github.io/AirSim-NeurIPS2019-Drone-Racing/_files/Sangyun.pdf.

[5] W. Guerra, E. Tal, V. Murali, G. Ryou and S. Karaman, Flightgoggles: Photorealistic sensor simulation for perception-driven robotics using photogrammetry and virtual reality, *IEEE Int. Conf. Intelligent Robots and Systems* (IEEE, 2019).

[6] Y. Song, S. Naji, E. Kaufmann, A. Loquercio and D. Scaramuzza, Flightmare: A flexible quadrotor simulator (2020), arXiv:2009.00563.

[7] R. Madaan, N. Gyde, S. Vemprala, M. Brown, K. Nagami, T. Taubner, E. Cristofalo, D. Scaramuzza, M. Schwager and A. Kapoor, arXiv:2003.05654.

[8] J. L. Sanchez-Lopez, M. Molina, H. Bavle, C. Sampedro, R. A. S. Fernandez and P. Campoy, A multi-layered component-based approach for the development of aerial robotic systems: The aerostack framework, *J. Intell. Robot. Syst.* **88**(2) (2017) 683–709.

[9] Stanford Artificial Intelligence Laboratory *et al.*, Robotic operating system, http://robotics.stanford.edu/∼ang/papers/icraoss09-ROS.pdf.

[10] M. Molina and P. Santamaria, arXiv:2103.13128.

[11] M. Molina, P. Santamaria and A. Carrera, arXiv:2103.06545.

[12] D. Mellinger and V. Kumar, Minimum snap trajectory generation and control for quadrotors, in *2011 IEEE Int. Conf. Robotics and Automation* (IEEE, 2011), pp. 2520–2525.

[13] C. Richter, A. Bry and N. Roy, Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments, in *Robotics Research* (Springer, 2016), pp. 649–666.

[14] M. Burri, H. Oleynikova, M. W. Achtelik and R. Siegwart, Real-time visual-inertial mapping, re-localization and planning onboard mavs in unknown environments, in *2015 IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS 2015)* (2015), pp. 1872–1878, doi: 10.1109/IROS.2015.7353622.

[15] S. Lynen, M. Achtelik, S. Weiss, M. Chli and R. Siegwart, A robust and modular multi-sensor fusion approach applied to mav navigation, in *Proc. IEEE/RSJ Conf. Intelligent Robots and Systems (IROS)* (2013), pp. 3923–3929, doi: 10.1109/IROS.2013.6696917.

[16] S. Garrido-Jurado, R. Munoz-Salinas, F. J. Madrid-Cuevas and R. Medina-Carnicer, Generation of fiducial marker dictionaries using mixed integer linear programming, *Pattern Recognit.* **51** (2016) 481–491.
[17] G. Bradski, The OpenCV Library, *Dr. Dobb's J. Software Tools* **120** (2000); 122–125.
[18] F. J. Romero-Ramirez, R. Muñoz-Salinas and R. Medina-Carnicer, Speeded up detection of squared fiducial markers, *Image Vis. Comput.* **76** (2018) 38–47.
[19] N. Koenig and A. Howard, Design and use paradigms for gazebo, an open-source multi-robot simulator, in *2004 IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, IEEE Cat. No. 04CH37566, Vol. 3 (IEEE, 2004), pp. 2149–2154.

Miguel Fernández-Cortizas received the B.Sc. degree in Industrial Engineering (with specialization in Automation and Electronics) and the M.Sc. degree in Automation and Robotics from the Universidad Politécnica de Madrid (UPM), Madrid, Spain, in October 2019 and November 2020, respectively.

He is currently pursuing the Ph.D. degree in Automation and Robotics at Universidad Politécnica de Madrid (UPM). His research interest includes agile drone flying control and trajectory generation with a special interest in deep learning methods for solving control tasks for UAVs. He has received several prizes in international competitions like MBZIRC 2020, OpenCV Competition, or RAMI IROS 2021.

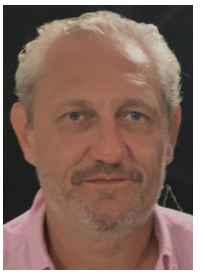
David Pérez-Saura is a Ph.D. student at the Universidad Politécnica de Madrid. He is currently working as a researcher in the Computer Vision and Aerial Robotics Group (CVAR) belonging to the Center for Automation and Robotics of the UPM-CSIC.

He graduated in Industrial Engineering from the Universidad Politécnica de Cartagena (UPCT), Murcia, in 2017, being awarded for his final degree project by Colegio Oficial de Ingenieros Industriales de la Region de Murcia (COIIRM). He obtained a Master's degree in Automation and Robotics from the Universidad Politécnica de Madrid (UPM) in 2021. His research interests are mainly focused on aerial robotics, working on state estimation and environment recognition, but also Deep Learning and Reinforcement Learning applied to aerial robotics. He has been part of the team awarded in OpenCV AI Competition 2021 and RAMI IROS 2021.

Pablo Santamaría received his M.S. degree in Computer Science Engineering from the Universidad Politécnica de Madrid, Spain, in 2021. For the year 2019–2021 he was Reseacher in the CVAR-UPM for the Aerostack project. Pablo is the author of over five technical publications. His research interests include AI, robotics and their applications.

Javier Rodríguez-Vázquez received the B.Sc. degree in computer engineering (double major in hardware engineering and computer science) and the M.Sc. degree in systems engineering and computing research from the Universidad de Cádiz (UCA), Cádiz, Spain, in July 2015 and March 2017, respectively.

He is currently pursuing the Ph.D. degree in artificial intelligence at Universidad Politécnica de Madrid (UPM). His research interest includes deep learning methods for solving computer vision tasks, with a special interest in unsupervised and weakly supervised methods for domain adaptation. He has received an international prize in the MBZIRC 2020 Competition.



Martín Molina is a professor in the Department of Artificial Intelligence, Universidad Politécnica de Madrid since 1994 (full professor since 2012). He received his Ph.D. in computer science and artificial intelligence and his licentiate degree in information technology from the Universidad Politécnica de Madrid (UPM). He was a visiting researcher for approximately three years in several research centers in the USA (AT&T Labs–Research, Stanford University and University of California–Irvine). From 1999 to 2016, Martin Molina led a research group about intelligent systems at his university.

Currently, Martin Molina is a member of the research group Computer Vision and Aerial Robotics at UPM, where he leads research lines related to artificial intelligence methods applied to aerial robotics (e.g. agent-based architectures, machine learning and human-machine interaction).

He has been the principal investigator of 29 research projects and he has authored more than 100 publications related to artificial intelligence. He has more than 20 years of experience working with intelligent systems and their practical applications with an important relation with private companies related to engineering problems (robotics, transport, hydrology, aeronautics, etc.).

Pascual Campoy is Full Professor on Automatics at the Universidad Politécnica Madrid UPM (Spain) where he lectures on Control, Machine Learning and Computer Vision. He has been visiting professor at DCSC Department in TUDelft (The Netherlands) from 2014 to 2019, and previously visiting professor at Tong Ji University (Shanghai–China) in 2013 and in Q.U.T. (Australia) 2011. He received his PhD in Automatics & Robotics at Universidad Politecnica Madrid in 1988, where he previously received his Master tittle in Automatics Engineering in 1983.

He is leading the Research Group on "Computer Vision and Aerial Robotics" at U.P.M. within the Centre of Automatics and Robotics (C.A.R.), whose activities are aimed at increasing the autonomy of the Unmanned Aerial Vehicles (UAV) by exploiting the powerful sensor of Vision, using cutting-edge technologies in Image Processing, Control and Machine Learning.

He has been heading director of over 40 R&D projects, including R&D European projects, national R&D projects and over 25 technological transfer projects directly contracted with the industry. He is author of around 200 international scientific publications and nine patents, three of them registered internationally. He is awarded in the top international UAV competitions: IMAV12, IMAV13, IARC14, IMAV16 and IMAV17, General Chair for IMAV 2019 and he coordinated the international team that was awarded in the third place in the Grand Challenge MBZIRC20.