Special Issue: The Mohamed Bin Zayed International Robotics Challenge (MBZIRC) 2020

Systems Article

Design and development of drones to autonomously interact with objects in unstructured outdoor scenarios

Sirena Cascarano[1], Mario Milazzo[2], Andrea Vannini[1], Andrea Spezzaneve[1] and Stefano Roccella[1]

[1]The BioRobotics Institute, Scuola Superiore Sant'Anna, Livorno, Italy
[2]The BioRobotics Institute, Scuola Superiore Sant'Anna, Pontedera, Italy

Abstract: Aerial drones are systems that have been largely employed in a number of civil applications, from surveillance and environmental monitoring to day-to-day service. In view of this, the 2020 Mohamed Bin Zayed International Robotics Challenge (MBZIRC) held in Abu Dhabi (UAE) was a competition that aimed to push forward the technological limits of the actual aerial devices. We present the design and development of drone prototypes to meet one of the goals of Challenge 1, namely autonomously identifying and attacking randomly-positioned colored targets. This paper reports the main hardware/software design choices to accomplish the highest flexibility and usability of our aerial platforms, namely the strategies to control the overall dynamics and vision for the eventual interaction with targets. We achieved the fourth place among 22 international teams, since we successfully dispatched all the targets, which were dispersed throughout an area of 6000 m^2, in 365 s. Although focused on specific goals, the methodologies developed in view of the competition are suitable for further improvements towards other application fields.

Keywords: aerial robotics planning, obstacle avoidance, computer vision, robotic interaction, MBZIRC

1. Introduction

Nowadays, multirotors aerial vehicles increasingly play a significant role in many applications, such as mapping, surveillance, meteorology, exploration, or search-and-rescue missions because of their versatility and hovering capability (Bachmann et al., 2009; Glock & Meyer, 2020; Leuenberger et al., 2020).

Sirena Cascarano and Mario Milazzo have contributed equally to this work.

Received: 30 September 2020; revised: 16 February 2021; accepted: 05 April 2021; published: 19 October 2021.

Correspondence: Mario Milazzo, The BioRobotics Institute, Scuola Superiore Sant'Anna, Viale Rinaldo Piaggio 34, 56025 Pontedera (PI), Italy, Email: mario.milazzo@santannapisa.it; Stefano Roccella, The BioRobotics Institute, Scuola Superiore Sant'Anna, Via del Cedro 38, 57122 Livorno, Italy, Email: stefano.roccella@santannapisa.it

This is an open-access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Copyright © 2021 Cascarano, Milazzo, Vannini, Spezzaneve and Roccella

Within the last decade, due to this growing interest and wide applicability of such devices, a large number of studies have been published about drones and, in particular, about multiple-rotor Unmanned Aerial Vehicles (MultiCopter UAVs), driven both by the investigation of solutions to overcome their technological limits, such as dynamic stability and energy self-sufficiency, and by scientific interests in different sectors (Baloch & Gzara, 2020; Zhu et al., 2020). In particular, from the engineering standpoint, different fields are involved: from the Aerospace/mechanical areas to Computer Science and Automation (Gupta et al., 2013; Hassanalian et al., 2017), aiming at developing autonomously flying systems able to interact with the surrounding environment and to perform always more complicated tasks.

Pursuing this path, the Khalifa University (Abu Dhabi, UAE) organized in Abu Dhabi in 2017 and, more recently, in 2020, the Mohamed Bin Zayed International Robotics Challenge (MBZIRC), an international competition involving the best teams in the robotic field from worldwide Universities and Research Centers. The organizers planned a number of high-risk, high-reward challenges for autonomous robots (terrestrial and aerial), aiming at pushing the capability of such devices and the competences of the selected teams (MBZIRC, 2020). The idea of such events is to promote the development of innovative strategies and enabling technologies through a competition in which technical excellence is the showcase for cutting-edge research activities. The outcomes of the challenges are not expected to be focused on, or limited to, the competition objectives only, but they are a sponsored tool to positively impact robotics in order to create new resources and approaches for improving the capabilities of such systems in a large number of applications.

This paper reports the strategic design choices and developments of hardware/software solutions for UAVs to address one challenge of the competition. Furthermore, our work aims at providing a toolbox of resources for researchers who intend to work on autonomous drones also in other fields not directly related to the competition.

1.1. Challenge 1 at 2020 MBZIRC: Rules and motivation

The organizers of the Competition designed four different challenges in which terrestrial and aerial robots, singularly or cooperating, aim to achieve specific objectives. The subject of this paper is related to a specific aspect of Challenge 1, in which a swarm of drones (maximum dimensions 1.2 m × 1.2 m × 0.5 m each) was expected to autonomously detect 5 colored targets (viz., equal-size air balloons fixed to the ground with poles at 2.5 m) randomly positioned in an outdoor arena (100 m × 60 m wide and 20 m high). Size, color and total number of these targets were unveiled by the organizers only during the rehearsal days just before the Competition. After the identification, an interaction procedure was requested in order to burst each target (Figure 1). The second part of the Challenge 1, not detailed in the present paper, concerned the interaction with a flying target carried by an externally-controlled autonomous drone. The time limit to complete the Challenge 1 was 15 minutes. Both the tasks were expected to be accomplished autonomously, but a pilot was allowed to take over the control in case of need (e.g., for safety reasons) with a contextual application of a penalty.

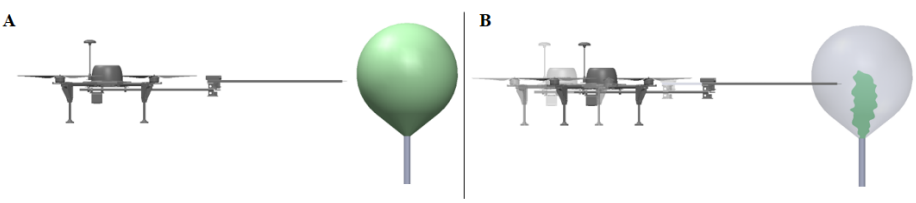


Figure 1. Goal of the challenge: approach (Panel A) and interact with (Panel B) five randomly positioned targets.

The motivation behind Challenge 1 is the achievement of a mature drone technology to detect and eliminate potential intruders in a restricted area, enhancing an active and autonomous security procedure (MBZIRC, 2020).

It is worth stressing that, in order to create a realistic non-structured environment in this scenario, the task statement was progressively unveiled by the organization, pushing the teams to create high-flexible devices able to demonstrate a high reliability in a large array of boundary conditions.

Based on these working conditions, we designed and developed two drone prototypes by assembling commercial and dedicated components (Section 2) and implementing software strategies for vision and navigation (Section 3) to accomplish the objective of the task. We used simulators to predict the behavior of the drones with different boundary conditions and conducted experimental field tests to optimize hardware and control strategies (Section 5) in view of the actual challenge during the competition in Abu Dhabi (Section 6).

1.2. Related works

Based on the design constraints described above, it was immediately clear that systems developed with standard approaches could not completely ensure the requested high flexibility. From the hardware point of view, we investigated both custom and traditional solutions aiming of meeting both the demanding size and weight requirements as well as self-sufficiency and modularity (Elliott, 2016; Floreano et al., 2010; Nonami et al., 2010; Zufferey, 2008). Similarly, a traditional use of an openCV for finding colored targets (Abughalieh et al., 2019; Coskun & Ünal, 2016) needs to be combined with customized adds-on for the specific application.

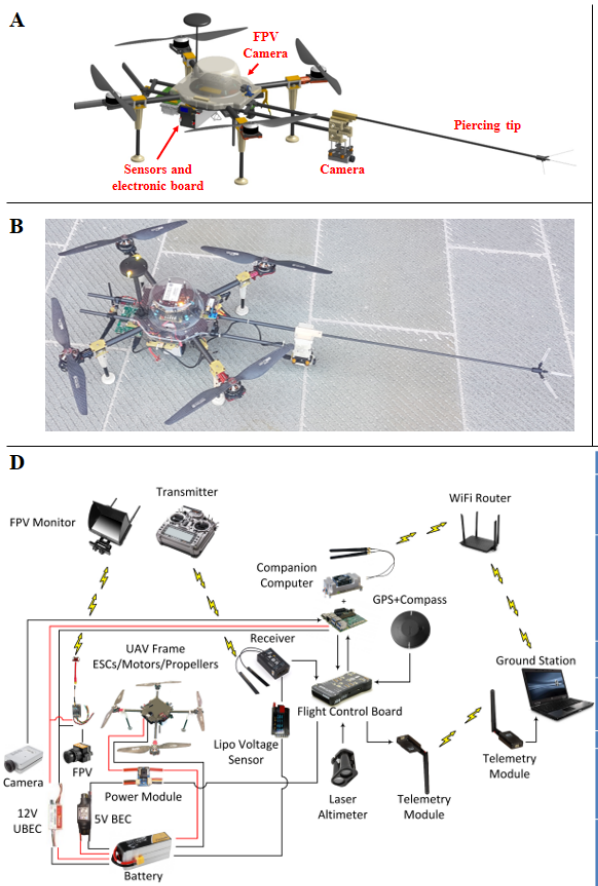
As for this, we analyzed the articles published by the participants of the 2017 MBZIRC: interesting elements were presented in Beul et al. (2019) and Spurný et al. (2019) concerning hardware settings, navigation strategies based on state machines (Bähnemann et al., 2019; Tzoumanikas et al., 2019), as well as some potential issues typical of vision algorithms (Chan et al., 2019).

Furthermore, in order to develop a system able to adapt to different scenarios and tasks, even beyond the Competition, it was essential to extensively know the dynamic behavior of MultiCopters and their performance upon variations of weight or dimensions, and/or flight parameters. From this perspective, we focused on previous studies that investigated the development of mathematical models describing the copter dynamics (Dang-Khanh Le, 2015; Fum, 2015; Vanin, 2013) as a basis for a model developed in Matlab/Simulink softwares (Pantalone, 2015) by interfacing with V-rep softwares *via* Lua scripts (Khalilov & Kutay, 2015). The presented approaches were ultimately with other traditional approaches including tracking strategies for UAVs (Koubaa, 2017; Persico, 2015).

2. Hardware

We designed our drone (Figure 2a,b) to comply with 2020 MBZIRC Competition requirements, in particular by taking into account the features related to the dimensions and the endurance throughout the duration of the challenge. We carried out a preliminary analysis with xcopterCALC, an open source software tool for designing multi-copters (Müller, 2020). Referring to Figure 2c, this tool calculates the range (in km) and time of flight that a specific drone is able to fulfill based on an array of input data. These included drone weight (with all payloads), frame geometry and dimensions, payloads consumption, motors, propellers, Electronic Speed Controllers (ESCs) and, finally, battery in terms of number of cells and capacity (viz., mAh). Along with the time of flight and range, other output data (here not shown) were given, e.g., motors' consumption and throttle level at hovering, motors' parameters at maximum throttle, thrust to weight ratio, specific thrust (total thrust/power consumption) and maximum drone speed. A sensitivity study on the input data led to the drone features that ensured a time of flight above 15 min, the limit required by the Challenge rules (MBZIRC, 2020).

System architecture is composed of commercially available off-the-shelf components, coupled with customized 3D-printed parts (Figure 2d) and wood/fiber glass interfaces. The modular configuration,



#	PART	NAME	COMPONENTS
1	Drone	Tarot 650 Sport quad-copter (modified)	• n.4 T-Motor MN3510-25 KV360 • n.4 CF propellers 15"x5" • n.4 HobbyWing ESCs 40A
2	Flight Control Board (FCB)	Pixhawk 1	• 9 DoFs Inertial Measurement Units • "Here 2" GPS sensor • Compass • 100 mW Unmanned Telemetry Kit V2 (433 MHz) • LightWare SF20/C LIDAR sensor
3	Autonomous flight	Companion Computer	• NVIDIA Jetson TX2 • Auvidea J130-NV Motherboard
4	Manual Flight	Pilot	• FrSKy Taranis X9D plus Transmitter • FrSky X8R Receiver • FrSky Lipo Voltage Sensor
5	Vision System	Camera	• RUNCAM HD camera
6	First Person View System	FPV	• EACHINE 1000TVL CCD camera • SMA Pigtail-Chaos FPV VTX Video Transmitter • BOSCAM Galaxy D2 FPV monitor
7	Power System	-	• GENSACE-Tattu Lipo 6S 7000 mAh 22.2V 25C battery • MAUCH HS-050-LV 50A Current Sensor • MAUCH 2-6S BEC 5.35V/3A • YONGLE UBEC 6A-5V/12V regulator

Figure 2. Hardware architecture of the drone. Panels A, B. 3D CAD model of the drone and actual realization of the platform. Panel C. Drone performance estimated by xcopterCALC Software (Müller, 2020). Panel D. Deployment of the hardware components and their functional connections.

based on open hardware and software (see also Section 3), allows quickly replacements in case of failures, also during the Challenge, and enables an easy reproducibility in view of future applications or for setting-up a fleet of drones.

The fundamental module is a modified Tarot 650 Sport quad-copter (motor inter-axis diameter equal to 590 mm) with a carbon/fiber frame and 4 motors controlled by a Flight Control Board (FCB) Pixhawk 1. The FCB is placed on the top plate of the drone and it is covered by a transparent canopy to protect the sensors from the aerodynamic turbulence. In order to estimate the drone position and attitude, the FCB firmware combines and integrates the data from the on-board sensors by means of an EKF (Enhanced Kalman Filter). However, concerning the altitude, instead of using the on-board pressure sensor, a LightWare SF20/C LIDAR sensor, directly connected to FCB, serves as an altimeter: it provides more accurate and stable measurements than the barometer, independently of light conditions and ground surfaces. As for the 2D positioning, we employ a Here 2 GNSS GPS sensor. This is the latest version of Pixhawk 1 compatible modules with enhanced positioning accuracy and faster response, including also the compass. In order to avoid an increase hardware and software complexity and also a score penalty, we decided not to integrate any RTK module for precision positioning even if MBZIRC regulations allowed it.

The FCB is also interfaced with a companion computer NVIDIA Jetson TX2 carried by the Auvidea J130-NV motherboard and messages are shared by means of a serial connection that uses a

MAVLink protocol. We choose the TX2 for the optimal relationship between its computing capacity, its dimensions, weight and low consumptions. This on-board computer is optimized for computer vision tasks, but it has enough computational power to manage, in parallel, drone's navigation and coordination algorithms (see Section 3). The 5-Ghz Wi-Fi connection with the local network, provided by 2020 MBZIRC organization, is performed by the TX2 embedded dual-band module that, additionally, allows the communication among multiple drones. A direct 2.4-GHz link between the drone and the pilot of the team is implemented through a FrSKy Taranis X9D plus transmitter and FrSky X8R receiver connected directly to the FCB. For safety reasons, the pilot oversees the drone flight and, only in case of failures or explicit consensus, is able to take over the control despite the running state of the ongoing algorithms. Drone's status during flight is monitored by means of a Ground Station implemented on an external laptop. Data are transferred through the Wi-Fi network with the User Datagram Protocol (UDP) connection. However, as a backup, we endowed the platform with a telemetry module.

A high-resolution digital camera RUNCAM HD is connected to the companion computer (jetson TX2) via a USB cable to detect and localize targets. This camera was chosen, compared to similar products, both for its technical specifications (i.e., resolution, wide field of view with low distortion, automatic exposure control and internal algorithms for the reduction of the Jello effect) and for its low weight. The camera is constrained to the drone frame by means of an anti-vibration platform with rubber dampers in order to limit the additional weight. Distances are calculated by vision algorithms (see Section 3.1) based on camera images and are corrected by a dedicated software that is able to take into account drone pitch angle, avoiding the employment of a gimbal, saving additional weight. Compared to other higher-weight solutions (i.e., a stereo-camera), the saved weight increases the autonomy of the drone and, at the same time, the recognition capabilities are not affected. Since size and color of the balloons were specified *a priori* by the competition rules, a single colour camera was considered suitable to estimate the distance and the position of the target. The drone is also provided of a CCD camera to transmit analogic FPV images to the pilot through a monitor with SMA Pigtail-Chaos FPV VTX Video Transmitter. Although we did not use it during the Challenge, we decided to place it in case of software issues to allow the pilot to take the control over and to prevent damages.

The drone is also endowed with a two carbon tubes rail structure constrained under the main plate with anti-vibrating rubber rings: it allows to carry payloads and batteries on sliding supports in order to find the correct weight balance. The companion computer is placed on the backside of the drone and balanced the front part, where a 3D-printed component held the camera and the piercing tip to interact with targets. The length of the piercing (viz., 60 mm) link is designed to avoid potential collisions with the propellers and, at the same time, to be outside the FoV of the camera (Figure 2a,b). The main structure of the drone was slightly modified with respect to the commercial version (Figure 3a,b). The joints for the arms were strengthened because the original polymeric connections, designed to allow the drone folding for a better handleability, were found as a significant source of vibrations. In order to reduce this phenomenon, we designed new clamps made of aluminium that prevented oscillations and backlash during flight maneuvers. A second aspect concerned the original landing gear that was replaced by four legs made of 3D printed plastic material and carbon fiber tubes, specifically designed to prevent any issue during take-off and landing procedures (e.g., drone flipping).

The drone is powered by a GENSACE-Tattu Lipo 6S 7000 mA h 22.2 V 25C battery connected to the integrated power distribution board with a MAUCH HS-050-LV 50 A Current Sensor. This is a Power Module of the FCB, able to precisely measure voltage and current in real time in order to detect battery consumption and execute landing procedures. Moreover, it was also used by the Ground Station to remotely monitor battery status.

The platform, including all the accessories and tools, reached the maximum take-off weight of 3.5 kg with dimensions equal to 1580 mm × 590 mm × 370 mm that perfectly fit the main size requirements (box 1200 mm × 1200 mm × 500 mm, diag 1697 mm).

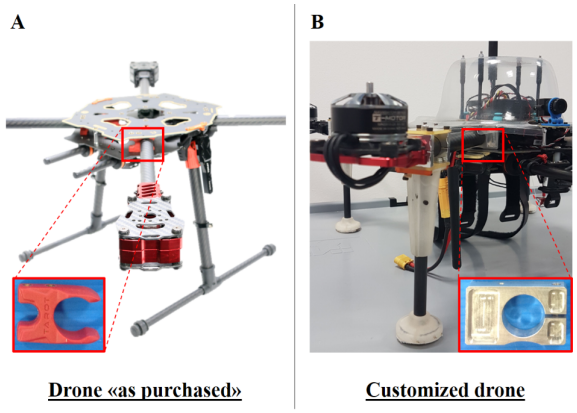


Figure 3. Structural modifications to enhance a mechanical stability. Panel A. Tarot 650 Sport quad-copter as purchased (picture from the web). The detail shows the V-shape connection between the main frame and the arm. Panel B. Drone customized by substituting the joints with an aluminum clamp and by providing a new fixed landing support placed at arm extremity.

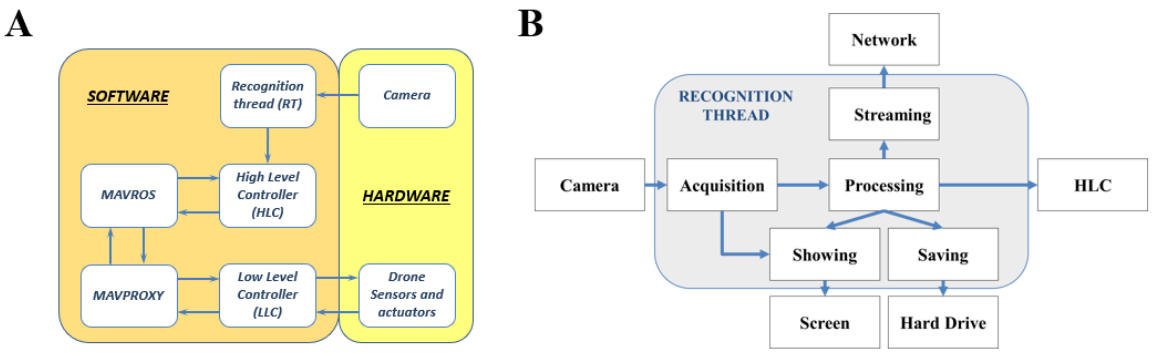


Figure 4. Multi-thread software structure. Panel A. Software architecture and interface with the hardware components. The HLC manages the robotic platform with a state machine that exploits the information given by drone sensors and actuators through the LLC and the MAVROS/MAVPROXY modules and data from the recognition thread that elaborates the images acquired by the online camera. Panel B: Multi-thread structure of the vision system.

3. Software

We built our software with a multi-thread architecture, shown in Figure 4a, where arrows indicate the directions of the directionality of data exchange. The High Level Controller (HLC) manages the navigation of the drone through a State Machine, a tool suitable to develop a modular system. Such a design allows separating the different operations the drone needs to perform (e.g., takeoff, tracking, landing) so that each activity can be tested independently without being affected by side effects due to reciprocal unknown relationships. For this application, the State Machine is implemented with the SMACH module (SMACH, 2019) in the Robot Operating System platform (ROS), an open-source set of software libraries and tools created for robotic applications (Koubaa, 2017) and employed in previous similar works (Chan et al., 2019; Spurný et al., 2019). The HLC provides the flight commands and manages the procedural state of the robotic platform, using camera data, which are elaborated by Recognition Threads (RT), and information from the drone sensors and motors managed by the Low Level Controller (LLC) implemented in the FCB Pixhawk 1 through the open-source ArduCopter software - *V3.6.12 Quad* (ArduPilot, 2019). The LLC, in turn, transfers

data to the on-board computer through the MAVPROXY and MAVROS software packages that communicate with the HLC (Mavros-MavLink, 2019).

The following sections deliver a full overview of the vision module and the State Machine, specifically designed for the MBZIRC Competition.

3.1. The Recognition Threads

The RT processes the images from the digital camera and, based on dedicated algorithms, it provides the HLC the main information about the target, as illustrated in Section 3.1.3. The vision system is divided into the following sub-activities: acquisition, processing, showing, recording and streaming (Figure 4b). The Acquisition thread acquires the image through the OpenCV instructions and stores it in a public variable, which can be read from other threads. The Processing thread is the most crucial part and will be detailed later in this Section. It reads the public variable from the Acquisition thread, processes it and stores the result in its own variables for a later use by other threads. The Showing thread reads the outcomes from the Acquisition and Processing threads and, with OpenCV commands, shows the raw and processed images on a monitor connected to the on-board computer. The Recording thread saves all the frames in a folder on the TX2 memory (USB memory stick, eMMC or microSD) so that we could review and examine, later in the laboratory, the actions performed by the drone and the results of the recognition threads through the single images or through a video obtained by converting the saved frames. We chose to save the frames and not the composite video for two reasons: 1) to better understand what the drone has seen frame-by-frame and 2) to not overload both the CPU and GPU. Streaming thread reads the processed images and streams the images over the network through our in-house server-client Python program based on an UDP-socket.

This modular architecture facilitates the development and the optimization processes of each individual phase and allows us to activate only what is needed so as not to overload the system resources. For instance, laboratory tests required only the acquisition, processing, and showing tasks, because it is possible to connect the on-board PC to a screen. Occasionally, we also activated the recording thread to better debug the software code. We employed all threads, except for the showing one, during the outdoor experimental sessions. In contrast, during the Competition we activated only the acquisition, processing, and streaming threads for maximizing the system performance.

3.1.1. The Processing thread

As mentioned above, the processing thread is the most crucial part of the RT. We developed an approach to improve the traditional technique based on color thresholding and shape recognition. This algorithm processes in real-time the 1080p at 30 fps video, captured by the camera.

The first step of our algorithm consists of applying a slight Gaussian filter to the frame to remove some noise (Figure 5b). Then, the image is converted from BGR color space to the HSV color space, in which we filter all pixels with colors different from the target's, obtaining a binary image (Figure 5c). We carefully tune this filter to exclude most of the image that does not have the right color, even if this means a partial filtering of the target itself. The aim of this stage is, primarily, to locate inside the frame any region containing the target, rather than providing an accurate detection. Additionally, erosion and dilation filters are applied to the binary image in order to cancel out the potential noise created by the color threshold procedure (Figure 5d) (Digiacomo et al., 2020; Solem, 2012; Szeliski, 2010).

Then, the OpenCV function "findContours", which was developed based on a previous work (Suzuki & Abe, 1985), recognizes all contours in the binary image (Figure 5e), the output list of such contours is first surveyed to filter all those that are too small or have perimeter over area ratio too different from that of the reference target (Figure 5f). For instance, in a 1920 px × 1080 px image, a 14-cm radius air balloon, positioned at 13 m from the camera has a diameter of approximately 28 pixels. Since we judged a distance of 13 m sufficient for the navigation, we filter all the contours having an area of less than 600 pixels wide (we rounded the actual 615.44 area of 14 pixel radius). Using the same threshold and proportions, the RT is able to recognize the competition balloons

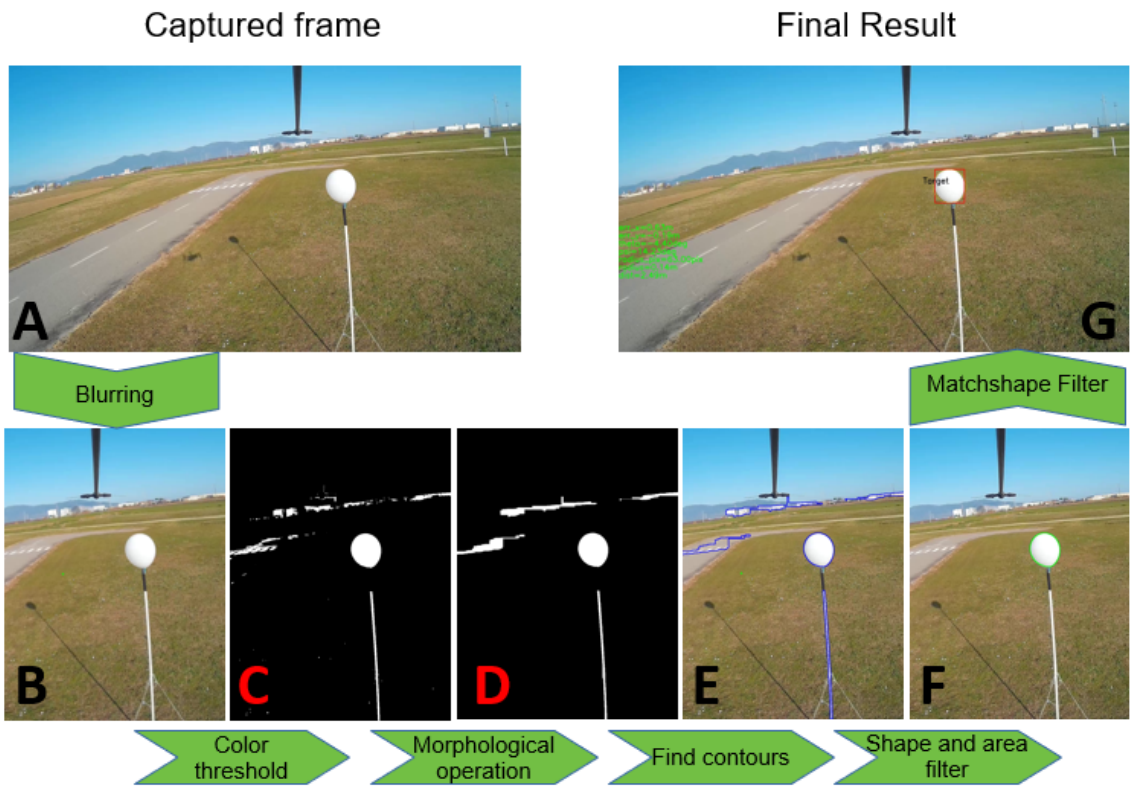


Figure 5. Recognition threads flow chart. Panel A: frame captured by the camera. For space needs, panels report only a ROI around the target but algorithms are applied on the entire frame. Panel B: frame blurred with Gaussian filter. Panel C: binary image obtained with the color threshold filter. Only pixels in a specific range (H [0–255], S [0–48] and V [194,255]) are drawn in white while others are black. Panel D: Morphological operations are applied to the binary image to reduce noise. Panel E: Drawn in blue, above the original frame, all the outlines returned by the OpenCV "findcontours" function applied on the binary image. Panel F: Drawn in green, the only contour that passed the filter on the area and the area/perimeter ratio. Panel G: Surrounded by a red bounding box, the contour that passed the "matchshape" filter.

(45-cm diameter) at a distance of about 35 m. A last filter is applied to the list of shapes that passed the previous stages (Figure 5g). Each contour is checked by a built-in OpenCV function "matchshape" that compares the contour with a reference silhouette (e.g., circle or air balloon silhouettes). This procedure also recognizes surrounding objects of color similar to the targets, and so requires a preliminary calibration to properly set the color threshold limits. In order to finely tune the set of colors that the target assumes under a particular light condition, we developed a Python program that acquires and processes the images from the camera, showing the result with a practical Graphical User Interface (GUI). This program can be run directly on the TX2 with a connected monitor, or remotely through a Ground Station over the network, depending on specific needs. Panels A&C or B&D in Figure 6 present two examples of the windows on the computer screen: panel A (or B) shows the result of the processing thread algorithm, and panel C (or D) illustrates the result of the first two stages (color conversion and color threshold) including six trackbars. These trackbars allow to easily select the actual values for ($[H_{\min}$–$H_{\max}]$, $[S_{\min}$–$S_{\max}]$ and $[V_{\min}$–$V_{\max}]$) of the framed target. The selected values are automatically saved as inputs for the RT. Moreover, this process also allows checking the geometrical characteristics of the balloon (e.g., its radius), by comparing the distance

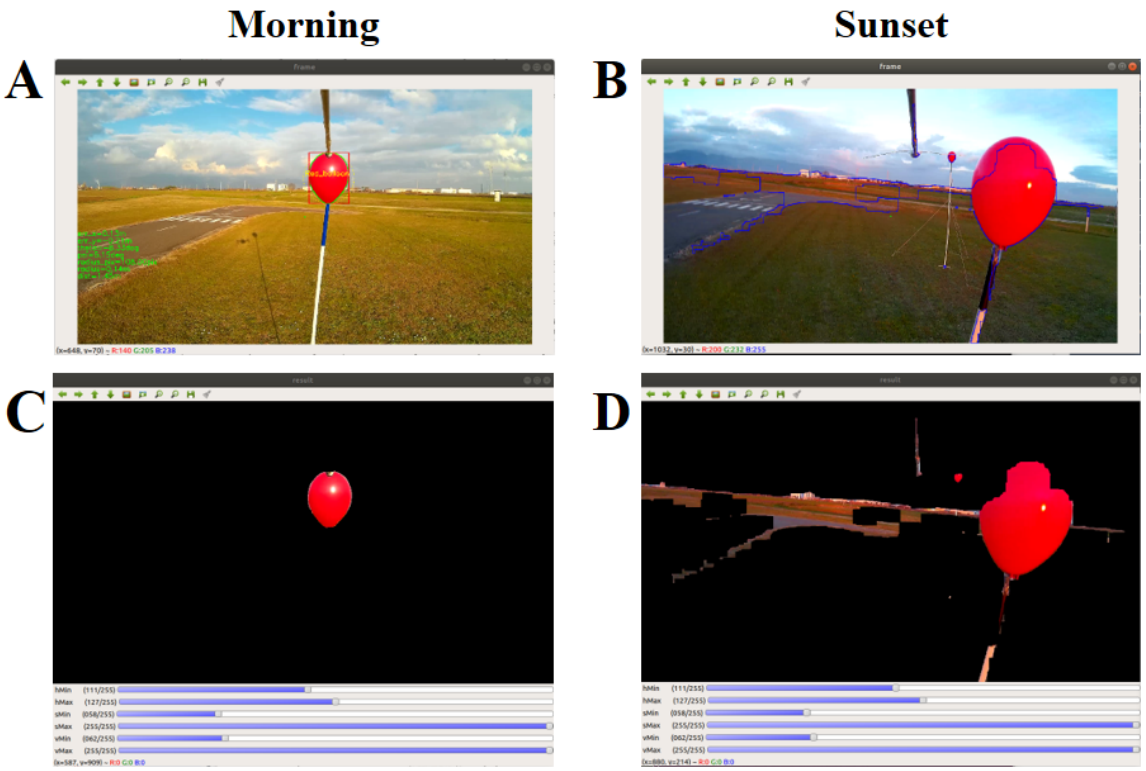


Figure 6. An example of the calibration applied on the same type of balloons at different times of the day (morning - Panels A, C vs. sunset - Panels B, D). Panels A, B reports the frames processed by the whole algorithm, while Panels C, D only the result of color threshold filter mask applied to the frames along with the trackbars for setting the HSV parameters. The correct parameters for the morning, calibrated with the white light of the sun, are unusable for the evening, when the red light of the sunset turns the scene red.

obtained by the processing thread with the real distance measured between the camera and the target.

The calibration process is necessary every time the ambient light conditions change significantly, since since they can modify the color appearance of the objects (for instance, a yellow balloon appears as orange at sunset). Thus, the same object is associated to different sets of colors depending on the ambient light that illuminates it. Figure 6 reports an example of such a situation, comparing two images framed during morning and sunset tests. Trackbars in panels C (morning) and D (sunset) are set on the same HSV values. As it can be observed, the threshold limits selected before the morning flight tests are not adequate for the sunset flight tests, because some pixels on the background, colored by the ambient light, are included in the HSV color threshold, whereas some balloon pixels are excluded. As a result, the balloon framed in Panel B (sunset) is not recognized as a target.

The calibration phase helps to quickly change the threshold limits during field tests according to the actual light condition (morning/evening) and makes the code not only precise, but also flexible, since it enables recognizing targets with different colors by simply tuning the HSV parameters.

3.1.2. The Memory Class

The most distinctive feature of the recognition process consists in the capability of avoiding false positives (FP), potential reasons for wasting time during the competition.

In this perspective, the last step of our vision "supervisor" that cuts out FPs and chooses the target to be reached, in case there is more than one balloon in the same frame. Following the object-oriented

programming paradigm, this "supervisor" was designed as a class, called "memory class" (MC), aimed to track over time the presence and the main attributes of targets in the frames (viz., radius and the graphic coordinates of the target center). To meet this goal, the MC maintains a list to which it appends a history object (HO) every time a new target is discovered. A HO is composed by 2 attributes: 1) a queue with, at most, the last three sets of info (position and radius) about the balloon associated with this HO; 2) a reliability index (RI), which indicates how much the associated queue is likely to be a true target.

An MC object is instantiated at the beginning of the flight and the RT interacts with it as follows. First, for each frame examined by the RT, all the contours that have passed the match-shape control are given as an input to the MC. Then, each contour is compared with those in the MC's HO list: if there is a similar entity in the memory, a situation that occurs if the centers of the two objects are close enough and their radii are comparable, the contour is appended in the queue and the related RI is increased by 1 unit up to a maximum $RI_{\max}$ (set to 20, see Section 5.1). Otherwise, a new HO is created, its RI is initialized to 1 and the target information is appended to the queue. Therefore, the RIs of the HOs that are not considered in the previous step are decremented by 1 unit and the HOs whose RIs are 0 are removed from the list.

In order to identify the target to be reached, the MC selects the HO with the highest RI among all those that have an RI greater than a threshold value, $RI_{\min}$, selected to distinguish a true positive (TP) from a FP (in our case $RI_{\min}$ set to 5, see Section 5.1). If 2 (or more) HOs have the same RI, the algorithm chooses the HO associated with the greatest radius, that usually corresponds to the nearest target. The most recent values of the balloon, related to the selected HO, are read from the queue and represent the output of the RT.

This procedure is important not only for separating TPs from FPs, but also for determining the first target to be reached, identified by the higher RI. Two different TPs in a frame can be distinguished not only for the different distance values, but also because they are generally associated to different RIs.

Section 5.1 reports in detail the reasons behind the development of the MC based on experimental results and the empirical evidence about the determination of $RI_{\min}$ and $RI_{\max}$ values.

3.1.3. Target features from the RT

The RT shares information with the HLC, namely the features of the identified target that are later employed in the control laws (see Section 3.2). The first parameter is the distance (d_N) from the camera to the target, estimated as:

$$d_N = \frac{r_w \cdot R_{\text{ref}} \cdot f_l}{s_w \cdot R_{\text{ob}}} \tag{1}$$

in which r_w is the resolution of the camera in width, R_{ref} is the reference radius for the target, f_l is the focal length of the camera, s_w is the sensor width and R_{ob} is the radius of the observed target.

RT also estimates the relative distances, along the y_D-/z_D axes, between the center of the body frame and the center of the target: δ_{T_x} and δ_{T_y} respectively.

$$\delta_{T_x} = \frac{s_w \cdot d_N \cdot \delta x}{f_l \cdot r_w}, \qquad \delta_{T_y} = \frac{s_h \cdot d_N \cdot \delta y}{f_l \cdot r_h} \tag{2}$$

in which r_h is the resolution of the camera in height, s_h is the sensor height and δ_x and δ_y are the differences in pixel from the center of the image (Figure 7).

3.2. The High Level Controller

Figure 8 illustrates the programmed states that combine the information from the MAVROS/MAVPROXY (flight commands to/from the drone and data from on-board sensors) and the RT (data from the vision module).

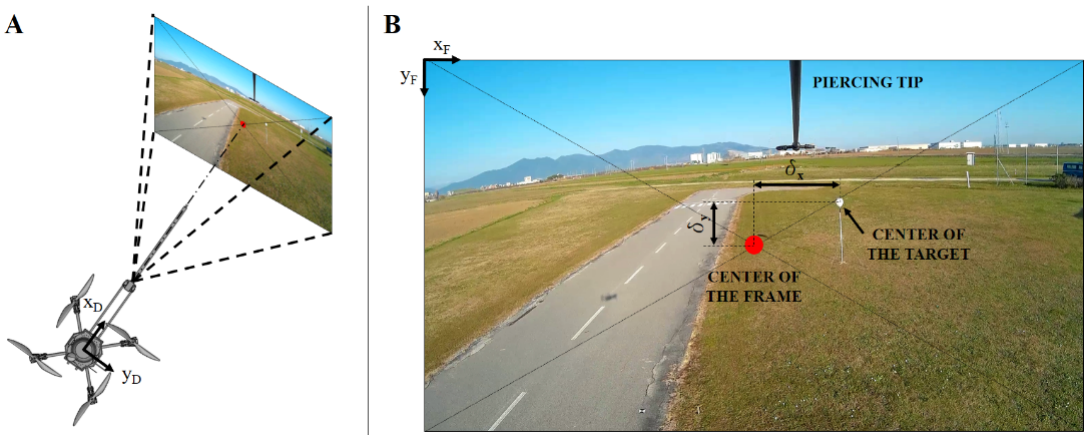


Figure 7. Qualitative depiction of the drone capturing a frame in the surrounding environment (Panel A). Panel B shows the main features (δ_x and δ_y) extracted from the frame.

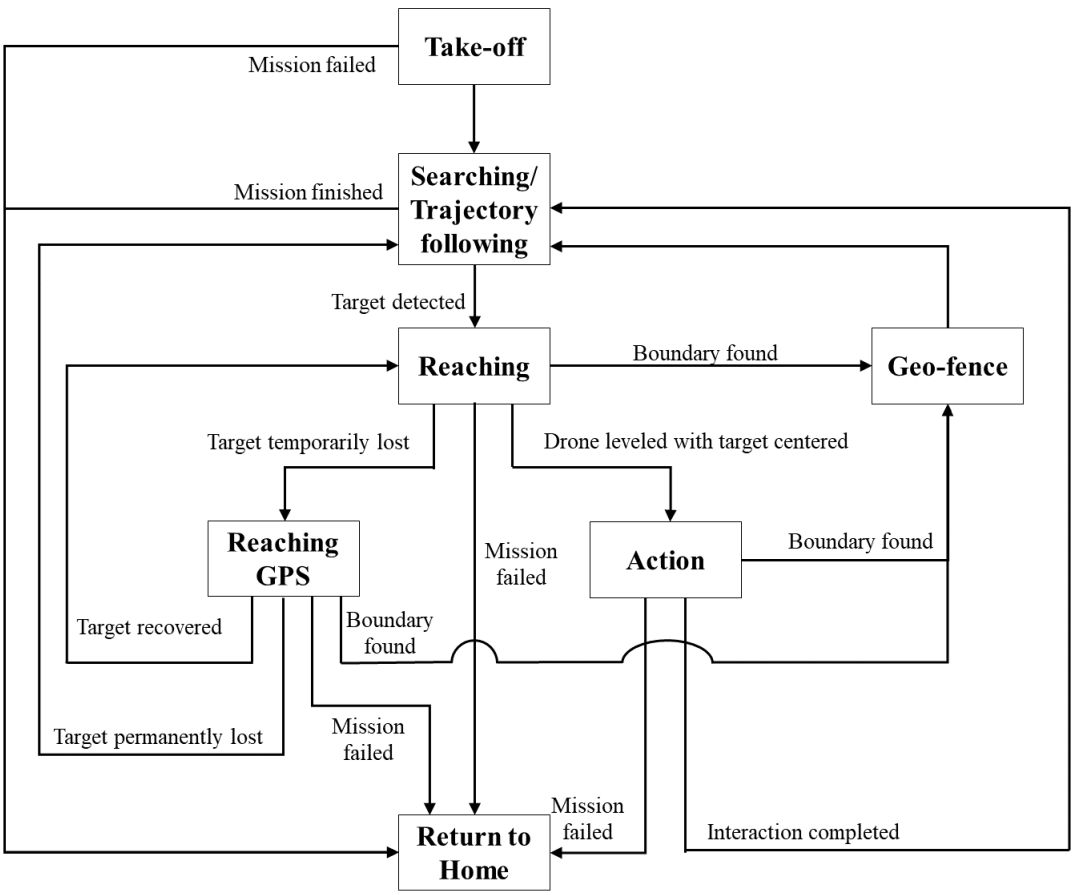


Figure 8. Main State Machine of the High level controller (HLC).

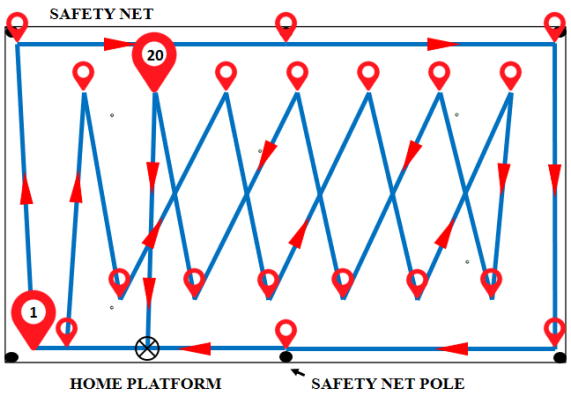


Figure 9. Waypoints: pre-set GPS points that define the searching trajectory of the drone.

Once the HLC is triggered, the drone in GUIDED mode automatically takes off from the home position to reach the preset altitude of the searching mission, before switching in AUTO mode and following the prescribed trajectory implemented by the LLC (*Searching/Trajectory following State*). The trajectory is defined *a priori* with Mission Planner, a ground station application (jDrones, 2016), as 20 pre-set GPS points (waypoints), selected by our team the day before the challenge on a satellite map of the arena. Figure 9 qualitatively shows the locations of the waypoints for covering the entire arena. While following the trajectory, if the RT detects a target, the HLC changes the LLC mode to GUIDED and the state to the *Reaching State*. This latter state aims at approaching the target up to a longitudinal distance equal to 1.5 m from it. At this distance, in order to prevent any potential collision, the HLC switches into the *Action State* and the drone attacks the target, consisting in bursting the colored balloon, according to different control laws (see Section 3.2.1.2). Both the *Reaching State* and the *Action State* are performed through a velocity control system in GUIDED mode, as detailed in the following Section 3.2.1.

During the *Action State*, if the target disappears from the scene, different scenarios open up, as depicted in Figure 10. In order to identify the actual situation, the drone moves about 1 m backward so that the RT can collect a new information related to the just processed area. At this stage, only if the RT detects no more targets, is the attack successfully verified: The drone abandons the target area, changes its altitude and resumes the pre-programmed path in AUTO mode searching for new targets. In contrast, if the RT finds a target in the same area, the HLC performs a check to verify the actual completion of the activity by evaluating the relative distance from it. The check is considered successfully passed if this distance is more than 2 m: in this case, the HLC sets a flag for a different target and the previous interaction is considered completed so that the drone can resume the pre-programmed trajectory in AUTO mode. If the distance is less than 2 m, the check is not passed and the system recognizes a void interaction so that the drone repeats the final approach and interaction. This circumstance (a void interaction) can occur if the target shifts out from the camera view before the drone steps back or if the drone gets too close to the target during the interaction so that the colored pixel in the image cannot be identified as a balloon.

Moreover, a possible issue during the *Reaching State* relates to a temporary disappearance of the target from the camera field of view or a void identification due to disturbances, light conditions, or wind gusts. In such cases, the RT cannot give new information about the target, but the task is not considered complete. Thus, the HLC activates the *Reaching GPS State*, giving the drone instructions to move towards the most recent target GPS coordinates through a position control system in GUIDED mode. The GPS coordinates are calculated and stored during the *Reaching State*, starting from the actual drone position and from the RT data about the locked target just before losing it, as illustrated in Section 3.2.1.1. If the target is recovered during the *Reaching GPS State*, the RT

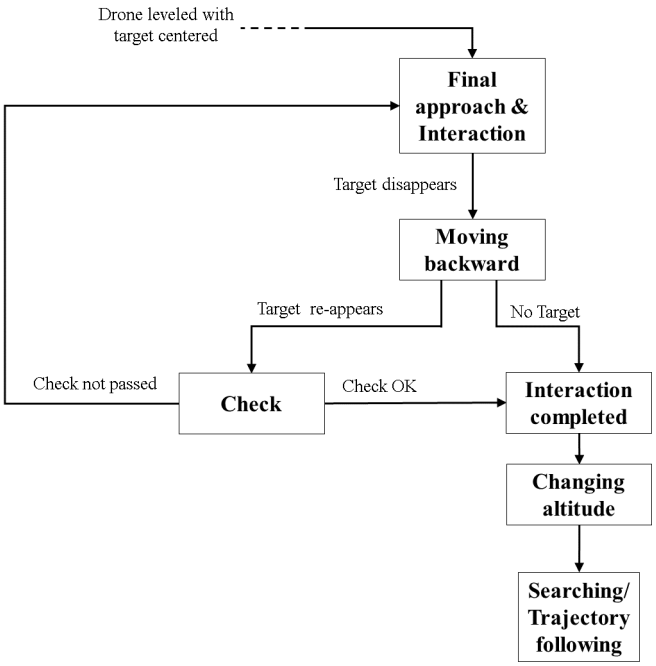


Figure 10. Subflow related to the *Action State* for interacting with the targets.

shares the related data with the HLC and the drone switches to the *Reaching State* again; otherwise the target is considered permanently lost and the drone resumes the *Searching/Trajectory-following State*.

While running the *Reaching*, the *Action* and the *Reaching GPS States*, the HLC also monitors the real-time position of the drone. During these states, if the drone is chasing a target erroneously detected outside the arena, it could move towards the borders, namely the safety net. In this case, comparing the drone position with the target position and the distance from the borders, as detailed in Section 3.2.1.3, the HLC switches into the *Geofence State*, a customized state that was specifically developed without using the standard procedure implemented in ArduCopter (Stevens & Atkins, 2016). Also, during this state, the LLC is set in GUIDED mode and follows a consequential scheme of velocity/position control laws, detailed in Section 3.2.1.3.

Finally, we designed an additional emergency procedure that is autonomously activated at any state in case of system issues: the HLC stops the current process and gives the drone instructions to head back to the home position (*Return To Home State*). During this state, the HLC no longer checks the data from RT and the drone is blind so that the emergency procedure can not be interrupted, even if the RT continues to run in order to stream the video, until the landing is completed. In addition to this safety procedure, the automatic flight can be ended by the pilot if the drone is out of control because of software/hardware anomalies: in such situations the pilot simultaneously takes over the control of the drone and stops both the HLC and the RT by using the pre-programmed stick on the transmitter that sets the mode of the LLC to STABILIZE.

3.2.1. Control laws for the HLC

This section is devoted to the main control laws in terms of positioning and velocity for the *Reaching State*, *Action State* and *Geo-fence State*.

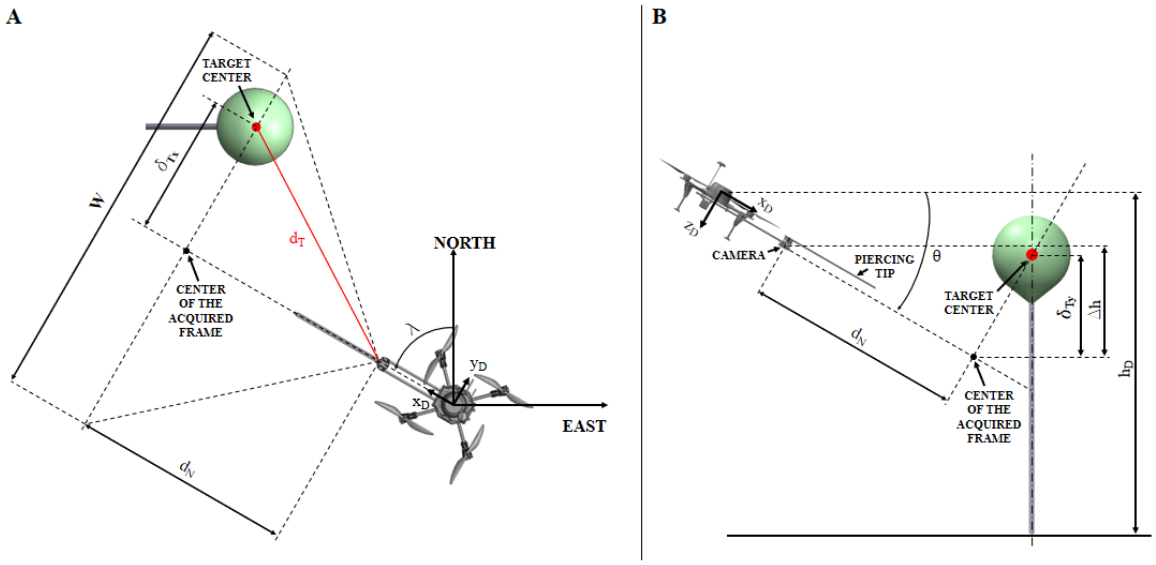


Figure 11. Alignment with the target. Panel A. View from a plane passing through the camera and angled by ϑ (pitch angle) with respect to the horizontal line. Panel B. Vertical alignment with compensation of the pitch angle. Note. Table 1 reports the description of the parameters employed in the Figure.

3.2.1.1. Reaching State During the *Reaching State*, in order to compute the actual GPS position of the target for the *Reaching GPS State*, the HLC collects and stores, instant by instant, the GPS coordinates of the drone (latitude - α_D, longitude - β_D and altitude - h_D) and other information from the RT related to the target (δ_{T_x}, δ_{T_y} and d_N, defined in Section 3.1.3).

As shown in Figure 11a, the planar distance between the origin of body frame of the drone (x_D, y_D, z_D) and the center of the target (d_T) is calculated from geometrical considerations:

$$d_T = \sqrt{d_N^2 + \delta_{T_x}^2} \tag{3}$$

The GPS coordinates of the target (latitude - α_T, longitude - β_T, and altitude - h_T) are, thus, estimated as

$$\begin{cases} \alpha_T = \alpha_D + \frac{d_T}{k} \cos\lambda \\ \beta_T = \beta_D + \frac{d_T}{k} \sin\lambda \\ h_T = h_D - \delta_{T_y} \end{cases} \tag{4}$$

in which k equals $1.11 \times 10^5\,\mathrm{m\,rad^{-1}}$, is a constant to convert the coordinates from the body frame to the GPS reference frame, and λ is the bearing angle.

Before computing the velocity component commands for the LLC, the HLC elaborates the RT data and makes a correction to the drone vertical alignment with the target because of its positioning error induced by the drone pitch angle (ϑ), which is more pronounced at high speeds (Figure 11b). Therefore, we estimated a corrected $\delta_{T_y}^{\mathrm{corr}}$ as:

$$\delta_{T_y}^{\mathrm{corr}} = \delta_{T_y} + \Delta h = \delta_{T_y} + d_N \sin\vartheta \tag{5}$$

Moreover, once estimated $\delta_{T_y}^{\mathrm{corr}}$, the HLC elaborates the yaw velocity component ($\dot{\psi}$) and linear velocity components (v_x, v_y, v_z), using specific boundaries for the highest/lowest values according to specific values $|v|_{\mathrm{min/max}}$ below the limits imposed by the competition rules (see Table 1).

The velocity components are calculated as follows: v_x is modulated as proportional to the relative distance between the actual target distance d_N and a safety distance from it ($d_{\min}$):

$$v_x = k_x(d_N - d_{\min}), \qquad k_x = \frac{v_{x_{\max}}}{d_{\max}} \tag{6}$$

where $d_{\max}$ is the maximum longitudinal distance, an experimental constant optimized for limiting the forward speed (see Table 1).

In order to define the control law to ensure the lateral alignment of the drone with the target, we establish two different cases based on the numerical value of δ_{T_x}. If δ_{T_x} is greater than a selected threshold value, we use the yaw rotation as the quicker maneuver to complete the target centering; on the contrary, during the experimental tests, we found a better accuracy using the v_y velocity component.

Equations 7 and 8 summarize this rationale:

$$\dot{\psi} = \begin{cases} 0 & \text{if } |\delta_{T_x}| \leq 1\,\text{m} \\ -k_\psi \operatorname{sgn}[\delta^{px}_{T_x}]|\delta^{px}_{T_x} - \delta^{px}_{T_{x_{\min}}}| + \dot{\psi}_{\min} & \text{otherwise} \end{cases} \tag{7}$$

$$v_y = \begin{cases} k_y \delta_{T_x} & \text{if } |\delta_{T_x}| \leq 1\,\text{m} \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

in which k_y is an experimental constant while k_ψ is calculated as follows:

$$k_\psi = \frac{\dot{\psi}_{\max} - \dot{\psi}_{\min}}{\delta^{px}_{T_{x_{\max}}} - \delta^{px}_{T_{x_{\min}}}} \tag{9}$$

$\delta^{px}_{T_{x_{\min}}}$ and $\delta^{px}_{T_{x_{\max}}}$ are the lowest and highest errors in pixel along the x-axis for the camera. $\dot{\psi}_{\min}$ and $\dot{\psi}_{\max}$ are the lowest and highest yaw rates.

Finally, v_z is set proportional to $\delta^{\text{corr}}_{T_y}$:

$$v_z = -k_z \delta^{\text{corr}}_{T_y}, \qquad k_z = \frac{v_{z_{\max}}}{\delta_{T_{y_{\max}}}} \tag{10}$$

in which $\delta_{T_{y_{\max}}}$ is the maximum vertical distance between the drone and the target, a constraint imposed for limiting the vertical speed component.

Using this approach, the target is, almost invariably, in the camera field of view, during the whole *Reaching State*. However, Equation 10 takes into account only the relative vertical position of the drone and the target, but it does not monitor the altitude of the drone, which datum is important to prevent a collision with the uneven ground (lower bound - $h_{\min}$), or an out-of-range altitude (upper bound - $h_{\max}$) as prescribed by the competition (MBZIRC, 2020). In order to find a trade-off between safety, precision for the vertical alignment with the target, and complexity of the control law, we designed an adaptive controller based on a dedicated algorithm. The HLC modified the control law in real time, based on the v_z velocity component calculated in Equation 10, the actual flight altitude (h_D), and the data received by the RT ($\delta^{\text{corr}}_{T_y}$). If the drone is at an altitude h_D in a sub-range between the $(1-\eta)h_{\max}$ and $(1+\eta)h_{\min}$, v_z does not require a further correction. η is a tolerance selected during the experimental tests (see Table 1) in order to ensure enough space to decrease and reverse the vertical velocity (both near $h_{\max}$ and $h_{\min}$). The HLC commands the same velocity either if the target is above the drone and the drone is close to $h_{\min}$, or if the drone is close to $h_{\max}$ and the target is beneath. On the contrary, if the drone is close to $h_{\min}$ and the target is beneath the drone, the HLC reduces v_z based on the altitude gap between h_D and $h_{\min}$; likewise, if the drone is close to $h_{\max}$ and the target is above it, the HLC reduces v_z based on relative measurement between h_D and $h_{\max}$. Finally, if the drone is below the $h_{\min}$, the HLC activates a safety condition, increasing the altitude with a positive v_z.

3.2.1.2. Action State We designed different strategies for the *Action State*, aiming at maximizing the reliability to attack the target. In contrast to the requirements for the *Reaching State*, the *Action State* requires a higher precision level, not only to successfully accomplish the task, but also to prevent any failed interaction and to avoid any potential collision with the target holders. Therefore, specific control laws were designed for this state, and specific proportional coefficients were tuned during the flight tests (see Table 1).

Similarly to the *Reaching State*, a correction for δ_{T_y} is introduced in order to balance the pitch angle of the drone, according to the previously outlined Equation 5.

The velocity component along the x-axis is defined similarly to Equation 6:

$$v_x = k_x^a(d_N - d_{\min}^a) + v_{x_{\min}} \tag{11}$$

in which k_x^a is an experimental coefficient (see Table 1), $d_{\min}^a$ is the minimum distance at which the drone contacts the target and $v_{x_{\min}}$ is the lowest speed command value sensed by the drone, according to the LLC, and experimentally confirmed during flight trials.

However, as already discussed, Equation 11 is not sufficient to properly manage the position of the drone close to the target for the interaction, since it does not take into account all the different possible conditions when approaching the target (e.g., different light conditions that affect the accuracy of the RT data or a wind gust that compromises the approach). Thus, we designed a piecewise linear function. We used as $d_{\max}^a$ the $d_{\min}$ numerical value used for the *Reaching State*, $l_p = 60\,\text{cm}$ as the length of the piercing tip, while $v_{x_{\max}}$ is highest allowed speed for the *Action State*.

With this solution, the HLC gives the drone the highest velocity along the x_D-axis when the robot is distant from the target (up to 1.5 m to it) and uses the lowest velocity only for performing the interaction. Moreover, a safety condition is implemented to move back the drone and avoid a potential collision with the pole holding the target if the relative distance is less than an arbitrary threshold d_{pole} selected during the on-field tests (see Table 1).

The control law to ensure the lateral alignment manages v_y and $\dot{\psi}$ that are strictly connected to the δ_{T_x} given by the RT. We use a rationale similar to that pursued for the *Reaching State*, but with specific proportional coefficients (see Table 1).

Finally, v_z is calculated with Equation 10. However, we provide a slightly different algorithm to set a specific v_z associated to small $\delta_{T_y}^{\text{corr}}$ numerical values. This allows centering the target with a reduced error, as measured during the experimental sessions (see Table 1).

3.2.1.3. Geofence State As stated earlier, the HLC checks the Geofence conditions during the other states to prevent any potential collision with the safety net or the frame poles.

We model the arena as a rectangle, whose boundaries are defined by imaginary straight lines between four virtual points, identified with the safety net poles, with coordinates $(\hat{x}_i,\ \hat{y}_i)$ in a Cartesian reference frame $(\hat{X} - \hat{Y})$ centered in the drone home position on the starting platform (Figure 12a).

A first preliminary analysis concerns the real-time estimation of the shortest distance (d) between the drone position $(\hat{x}_D,\ \hat{y}_D)$ and the closest boundary. It is calculated as

$$d = \frac{|a\hat{x}_D + b\hat{y}_D + c|}{\sqrt{a^2 + b^2}} \quad \text{in which:} \quad \begin{cases} a = \hat{y}_2 - \hat{y}_1 \\ b = \hat{x}_1 - \hat{x}_2 \\ c = \hat{x}_2\hat{y}_1 + \hat{y}_2\hat{x}_1 \end{cases} \tag{12}$$

In this case the critical condition is related to the distance necessary to stop the drone flying at its maximum speed. For this reason, if d is less equal to a threshold d_{lim} (from experiments), the highest numerical value for the v_x component, calculated above for other states (Equations 6, 11), is proportionally scaled to d:

$$d \leq d_{\text{lim}} \quad \wedge \quad v_{x_{\max}} = 0.5d \tag{13}$$

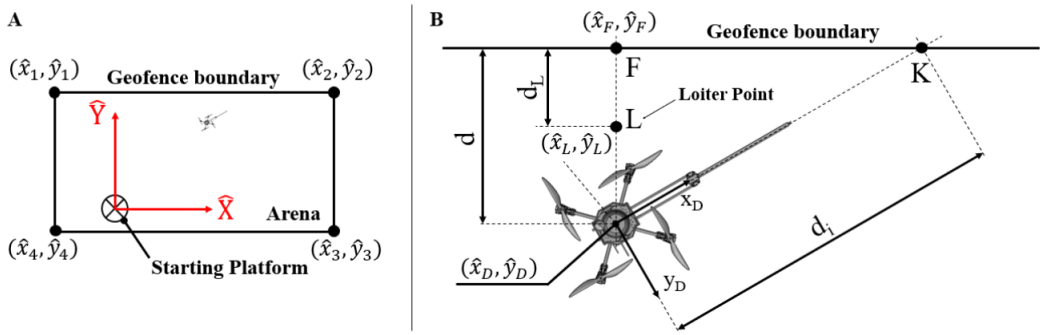


Figure 12. The *Geo-fence State*.Panel A: definition of the Geofence boundaries and identification of the reference frame centered on the starting platform. Panel B: Geometrical analysis for the *Geo-fence State*. Note. Table 1 reports the description of the parameters employed in the Figure.

However, since this case can also occur if the boundary is behind or beside the drone, this condition is not sufficient to enable the *Geo-fence State*. This is why we also consider the distance d_i from the real-time drone position to the intersection point between the boundaries and the direction of the drone head. This strategy allows the HLC to identify the boundary towards which the drone is moving. We enable the *Geo-fence State* if

$$d_N \geq d_i + \Delta d \quad \vee \quad d_i \leq d_{i_T} \tag{14}$$

in which Δd is a tolerance for the boundary distance, and d_{i_T} is a threshold value, both tuned during flight tests (see Table 1).

In both the cases presented in Equation 14, the *Geo-fence State* is activated and the drone gradually slows down and stops at the Loiter point L (Figure 12b). The Loiter point is located on a segment, perpendicular to the closest Geofence boundary, that connects such a boundary with the actual position of the drone. Being F $(\hat{x}_F, \hat{y}_F)$ the point on the Geofence Boundary, we have:

$$\hat{x}_F = \frac{b(b\hat{x}_D - a\hat{y}_D) - ac}{a^2 + b^2}, \qquad \hat{y}_F = \frac{a(a\hat{y}_D - b\hat{x}_D) - bc}{a^2 + b^2} \tag{15}$$

The position of the Loiter point $(\hat{x}_L, \hat{y}_L)$ with respect to the Geofence boundary (viz., F-point), is determined by fixing the experimental constant d_L (see Table 1):

$$\hat{x}_L = \hat{x}_F + \text{sgn}\,[\hat{x}_D - \hat{x}_F]\frac{d_L}{1 + \left(\frac{b}{a}\right)^2}, \qquad \hat{y}_L = \hat{y}_F - \frac{b}{a}(\hat{x}_F - \hat{x}_L) \tag{16}$$

At the Loiter point, the HLC performs a yaw rotation equal to the angle between the actual heading direction and the direction for the next waypoint, so that the camera can focus again inside the arena and the drone can resume its *Searching/Trajectory following State*. In parallel, the GPS coordinates of the objects potentially located outside of the arena are stored in order to be discarded later if the RT found it again.

4. Simulation tools

In this Section, we present the simulation tools that we used to develop our systems for the 2020 MBZIRC Competition. At an early stage, our main aim was to reproduce the behavior of the LLC to better manage the drone dynamics during the standard flight phases (i.e., take-off, hovering, trajectory following) and to study its response to single or multiple flight commands: in view of this, we designed a customized simulator named I-SIMC, as described later in this Section. At a later stage, a different simulation strategy by means of a Software In The Loop (SITL) Simulator was

crucial to safely test the operation strategy implemented into the HLC, reducing the development time and risk of accidents: SITL is able to simulate the main aspects of the LLC without using the actual drone hardware (ArduPilot Dev Team, 2020). Hence, in order to reproduce the behavior of the firmware installed on the FCB, we used the ArduPilot SITL simulator with the same version (*ArduCopter V3.6.12 Quad*). It is worth to note that both simulation tools are very powerful to debug the control strategies independently by the experimental conditions (as light conditions, wind gusts or disturbances).

Before running the experimental tests, we simulated the interaction between the HLC and the RT in laboratory using two open-source software platforms: Mission Planner (MP) and SITL simulator. We test the system at our laboratory in Livorno (Italy), using actual air balloons. As in the real system, the RT elaborates the images from the camera and, if a target is detected, it sends the related info to the HLC. Starting from these data and from the sensor data related to the actual condition of the drone, as simulated by the SITL and shared by MAVPROXY, the HLC selects the correct state and sends the computed commands to the SITL by means of MAVPROXY. During the simulations, the balloons are properly moved inside the laboratory in order to test the most of the functions described in Sections 3.1 and 3.2, from the *Take-off State* to *Return to Home State*. Figure 13 shows some pictures taken during this type of test in our laboratory in Livorno (Italy). During the first steps of the preparation for MBZIRC 2020, as previously mentioned, we developed the Integrated SImulator for MultiCopters (I-SIMC), a custom simulator composed of different development environments. I-SIMC combines the 3D physical environment modeled by the Virtual Robot Experimentation Platform software (V-Rep, Coppelia Robotics GmbH, Switzerland) with an *in silico* model of the system dynamics implemented with Matlab/Simulink (The MathWorks, Inc., USA), in place of the LLC. These two components share data with ROS libraries (Figure 14).

We reproduced a detailed CAD model of our drone and included it into the 3D virtual environment in V-Rep, where the simulated FCB primary sensors (e.g., gyroscopes, accelerometers) are combined to the 3D drone model. The simulated drone communicates/receives information to/from the Simulink

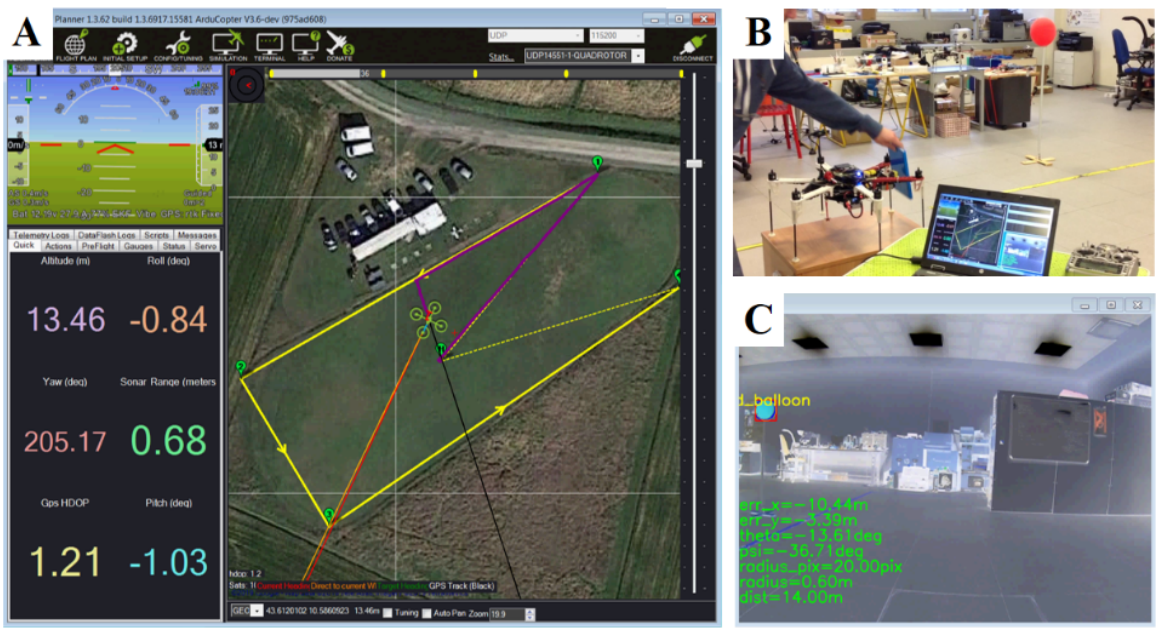


Figure 13. Pictures taken during a simulation test at our laboratory in Livorno (Italy). Panel A. Mission Planner graphics to visualize the simulated trajectory and sensor data from SITL. Panel B. Laboratory trials to frame a target with the on-board camera. Panel C. Showing thread graphics from the RT with the processed frames.

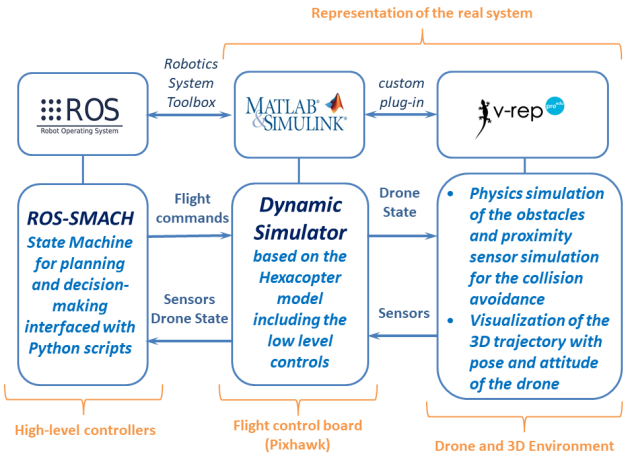


Figure 14. I-SIMC Architecture with which we simulated the aerial platform.

model to follow a programmed trajectory. In Simulink/Matlab, the data from V-Rep are processed by a nonlinear mathematical model that represents the drone dynamics: the vehicle is seen as a six-degrees-of-freedom system described by the Newton-Euler equations of motion of a rigid body affected by external forces and torques. In order to implement this formulation, two reference frames are used: an absolute inertial frame (x, y, z) and a body referenced frame connected to the drone and centered in its center of gravity (x_B, y_B, z_B). The relative angular position between these reference frames is described by the Euler angles (ϕ, θ, ψ).

Building on these general concepts, the translation dynamics can be expressed as

$$m\ddot{\xi} = F_g + T \tag{17}$$

in which m is the mass of the drone, $\xi = \{x, y, z\}$ is the position vector, F_g is the gravitational force and T is the total thrust from the rotors (in number of n). According to the Renard's formulation (Kannerworff, 1992):

$$T = \sum_{i=1}^{n} T_i = K_T \sum_{i=1}^{n} \omega_i^2, \qquad K_T = c_T \rho d^4 \tag{18}$$

in which T_i is the thrust provided by a single propeller, ω_i is the rotor angular velocity and K_T is the thrust coefficient of the propeller. This constant is estimated through $c_T = 0.1$, a factor specific of the propeller type (Kannerworff, 1992), the air density $\rho = 1.2\,\mathrm{kg\,m^{-3}}$ and d as for the diameter of the propeller.

By neglecting the torque due to the angular acceleration of the propellers and considering the propeller inertia matrix J_b as diagonal, the rotational dynamics of the system is expressed as:

$$J_b\dot{v} + v \times (J_b v) + J_{\text{gyro}} = \tau_B \tag{19}$$

in which J_b=$\{J_x, J_y, J_z\}$ is estimated through the 3D CAD file, $v = \{p, q, r\}$ is vector of the angular velocity, $v \times (J_b v)$ is the centrifugal force, $J_{\text{gyro}} = \{-J_r \dot{q}\omega_r, J_r \dot{p}, 0\}$ is the gyroscopic contribution given by the rotational inertia of the blades (J_r) and their total angular velocity (ω_r).

The external torques are expressed by $\tau_B = \{\tau_\phi, \tau_\vartheta, \tau_\psi\}$, estimated by following the Renard's (Kannerworff, 1992):

$$\begin{cases} \tau_\phi = \frac{d}{2} \sum_{i=1}^{n} K_T \omega_i \sin\left(\frac{2\pi(i-1)}{n}\right) \\ \tau_\vartheta = \frac{d}{2} \sum_{i=1}^{n} K_T \omega_i \cos\left(\frac{2\pi(i-1)}{n}\right) \\ \tau_\psi = Q_T \sum_{i=1}^{n} (-1)^{i-1} \omega_i^2, \qquad Q_T = c_Q \rho d^5 \end{cases} \tag{20}$$

where $c_Q = 0.05$ is the torque factor, specific of the propeller type.

A distribution model is included to compute the required angular speed of each rotor (ω_i) by means of trigonometrical relationships depending on the geometrical location of the rotors themselves (Sidea et al., 2014). Starting from the control signals related to the vertical thrust (U_{th}) and the attitude variations in pitch, roll and yaw (U_ϑ, U_ϕ, U_ψ), we have:

$$\omega_i = U_{th} + U_\vartheta \cos\left(\frac{2\pi(i-1)}{n}\right) + U_\phi \sin\left(\frac{2\pi(i-1)}{n}\right) + (-1)^i \times U_\psi \tag{21}$$

We managed the dynamics of the drone with PID controllers using gains that were tuned following an optimization process consisting of smoothing the overshoot and the oscillations in order to converge to the desired behavior and to reproduce the performance of the real LLC reported during preliminary flights.

4.1. Simulating obstacle avoidance

The I-SIMC simulator was mainly used in order to create a robust control strategy also in case of obstacles since, when the 2020 Competition was announced, rules included also this specific scenario. We simulated the autonomous navigation of our drone using V-Rep in a large number of scenarios. Walls, gates and columns were employed to tune the control laws (Figure 15). Although the final rules of the Competition did not include obstacle avoidance tasks, our approach improved the optimization of the control laws, giving also the opportunity to investigate the dynamics of the UAV in presence of objects, that could have also resembled, in a way, the targets of the Challenge.

In the I-SIMC, we used a control law for v_x similar to that presented in Equation 6, but we modulated it as proportional to the distance from the obstacle, estimated by proximity sensors in place of the target distance given by the camera. This allowed us to pre-test and tune different control

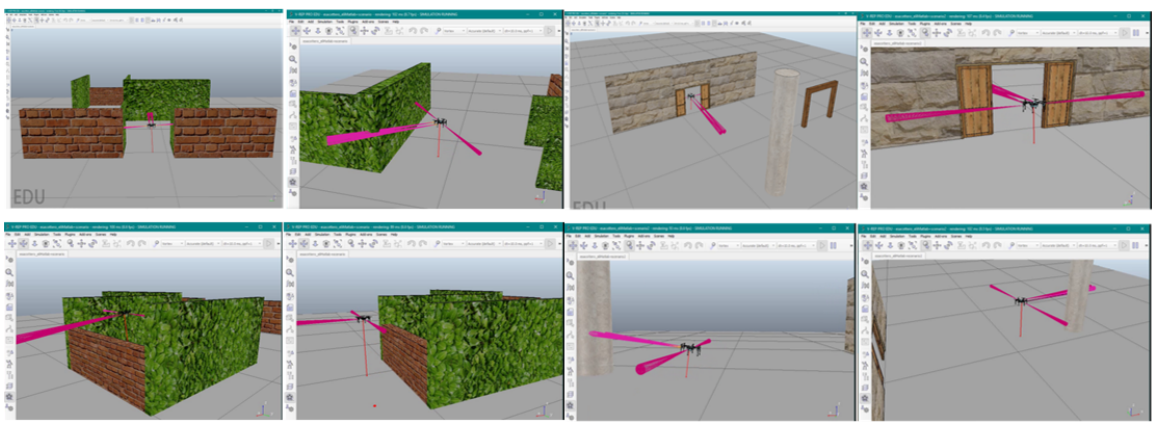


Figure 15. Screenshots from V-Rep simulator: the drone faces different types of obstacles (e.g., walls, gates), controlling the navigation by means of the on board sensors, whose spatial range is represented by the purple cones originated in the drone body.

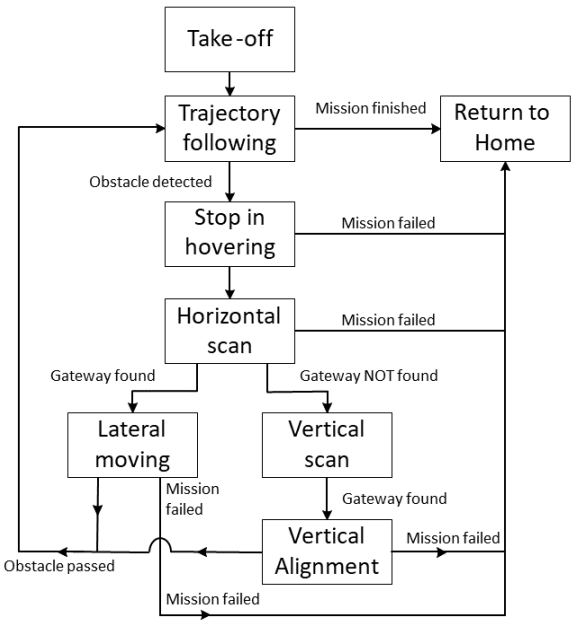


Figure 16. State Machine for Obstacle Avoidance tested with I-SIMC.

laws in view of the one used for the *Reaching State* in order to take into account the inertia of the drone along the axis of principal direction of the motion.

The scenes in V-Rep were also designed to test different skills of the HLC: 1) to find an obstacle and keep a safe distance, as planned with a target during the *Reaching State*; 2) to analyze the surrounding flight environment and to estimate the relative position from the unsafe areas, as planned for the *Geofence State*; 3) to autonomously make decisions (e.g., change of altitude, correct direction) to define and follow a trajectory adequate to the specific mission.

As for the I-SIMC simulator, we designed a dedicated state machine for the navigation in scenarios with obstacles (Figure 16) to better understand the acting of the main State Machine presented in Section3.2. According to this specific strategy, the drone automatically takes off from the home position to reach the preset altitude, before following a prescribed trajectory inside the V-Rep scene (*Searching/Trajectory following State*). While following the path, if an obstacle is detected by the sensors, the drone starts the *Stop in hovering State* and the v_x component, computed in real-time as a function of the actual distance from the obstacle, is sending by the HLC to the Simulink model. After the drone reaches a threshold distance to the obstacle (i.e., 1.5 m), it hovers to characterize the surrounding objects. Firstly, the HLC enables an *Horizontal scan State* and, if a gateway is found during the scanning, it plans a new trajectory. Otherwise, the HLC enables a *Vertical scan State* in order to find an alternative way and, according to the data collected during both the scanning procedures, it decides the best path for avoiding the obstacle. The transversal velocity components (v_y, v_z) are constant values, equal to 0.5 m/s, selected by the HLC and sent to the Simulink model until the obstacle is detected, before the drone can resume the prescribed path.

Finally, we tested into the I-SIMC the safety state, as described in Section 3.2, to simulate emergency events (e.g., low battery level).

5. Experimental tests on the field

We carried out the experimental tests in licensed flight fields in Pisa (Italy) in order to test and optimize the control strategies and assessing the overall drone performance.

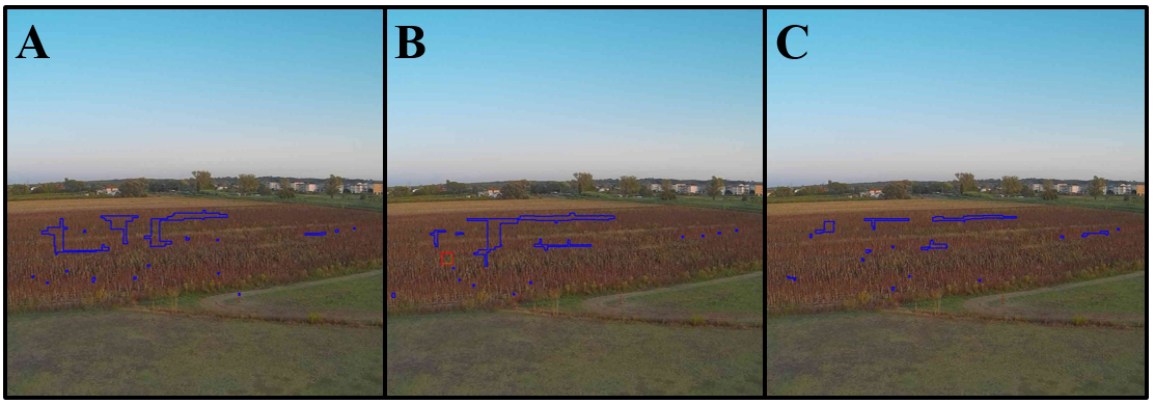


Figure 17. Example of the color-threshold side effect generated in 3 consecutive frames during a test flight in search of red balloons on a reddish brown field (Panels A–C). Blue contours are those after applying color threshold, and erosion and dilatation filters. The red bounding-box is the only boundary that passed the match-shape filter. In Panel C the red-bounding box is removed to show the randomness of the phenomenon.

5.1. Optimizing the Processing thread

During the developmental stages, we noticed that, occasionally, the RT described in Section 3.1, returned a false positive (FP), a potential issue in view of the correct execution of the drone mission.

FPs are randomly generated when a frame contains pixel areas whose set of colors straddles the threshold limit in the color space. As an example, Figure 17 depicts three consecutive frames captured during an experimental trial on the field in which the RT searches for red-colored targets. In all the three images the contours that pass the color threshold are drawn in blue. The shape and size of such contours are unpredictable and they quickly change frame-by-frame. This randomness could lead to contours that are accepted by the match-shape control and that are labeled with red bounding box generating a false positive (Figure 17b).

The randomness with which these outlines appear in the frames along with the low probability of a false positive, allowed us to develop and add to our vision algorithm the "memory class" (MC), described in Section 3.1.2, which analyzes all the contours approved by the match-shape function. The idea is to examine multiple frames to discriminate true positives (TPs), whose outlines remained similar in terms of shape, position and size in a set of consecutive frames. These features, indeed, are not possessed by false positives.

We used a probabilistic approach assuming: 1) the camera is not framing any target; 2) there is a FP in the N^{th}-frame ($f^{[N]}$), as it occurs in Figure 17b. Let $P(E_1)$ be the probability that a FP is generated in $f^{[N]}$. Experimentally, we noticed that this probability is very low (below 10^{-2}). Similarly, $P(E_2)$ is the probability that in the consecutive frame ($f^{[N+1]}$) a FP is generated in the vicinity. If this second FP is not present in the vicinity, this could be related to another target, which is not of interest at the moment. Given the absence of any target in the frames, the two probabilities can be considered independent. Accordingly, the probability that two FPs are generated in the same image portion in two consecutive frames ($P(f^{[2]})$) can be calculated as

$$P(f^{[2]}) = P(E_1) \times P(E_2) \tag{22}$$

By extending this rationale to N consecutive frames, the global probability will be

$$P(f^{[N]}) = \prod_{i=1}^{N} P(E_i) \tag{23}$$

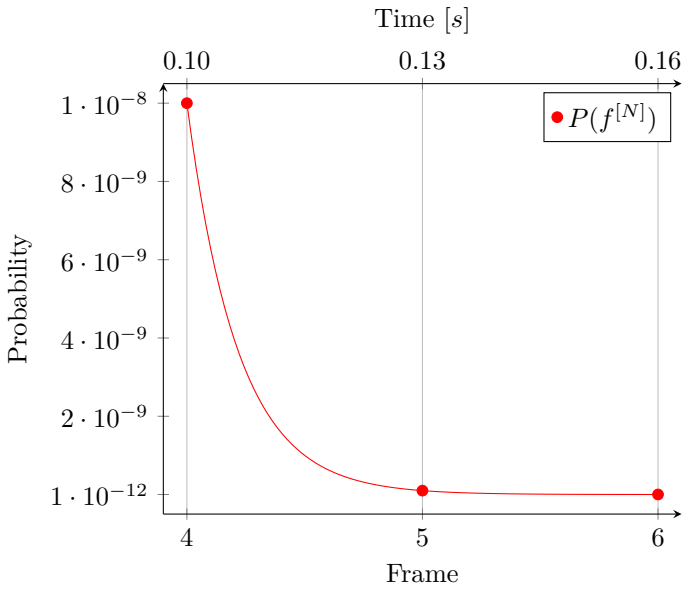


Figure 18. Plot of the $P(f^{[N]})$ probability function in the range 4-6 frame. On the vertical axis the probability that FPs appear in consecutive frames is reported as a function of the number of consecutive frames (bottom horizontal axis); on the top horizontal axis the corresponding time spent for the acquisition of subsequent frames (using a 30-fps camera).

These hypotheses can be extended by considering, more realistically, only a Region of Interest (ROI) of the frame. Since we experimentally found that $P(E_i) < 10^{-2}$, the global probability $P(f^{[N]}) \to 0$ with $N \to \infty$. Figure 18 plots a detail of the Probability function (Equation 23) and shows that, already between 4 and 6 frames, the Probability value decreases from 10^{-8} to 10^{-12}.

Therefore, the MC strategy consists in considering an object as a TP only if it shows up in the same portion of consecutive frames for a substantial number of times. This number that corresponds to $RI_{\min}$, must be high enough to annihilate the probability of generating a FP but, at the same time, as small as possible to reduce the delay for identifying a potential actual target. It is worth to notice that each frame approximately corresponds to 30 ms to 40 ms and to about 30 cm to 40 cm traveled at the maximum speed allowed by the competition rules.

Several tests were carried out to determine the $RI_{\min}$, both in the laboratory and in the field. The starting value was set at 2 (the minimum value possible) and we progressively increased it until we complied the desired requirements. In our application the optimal value fell between 4 and 5 and, cautiously, we selected $RI_{\min} = 5$. In general, $RI_{\min}$ deeply depends on the robotic platform in which this software is implemented: in fact, this parameter can change based on system features, i.e., the processor power, the fps and resolution of the camera or the maneuverability and speed of the drone.

During the early stages of the development of MC, we dealt with another problem: the appearance of a "Ghost balloon" due to the uncontrolled increase of the RI index, as described in Section 3.1.2. To better understand this phenomenon, we can consider, as an example, a set of 180 frames: the first 90 frames (from $f^{[1]}$ to $f^{[90]}$) depict a target, while the remaining frames (from $f^{[91]}$ to $f^{[180]}$) are black images. According to the rules described in Section 3.1.2, without introducing a $RI_{\max}$, the RI increases up to its maximum value (90) in $f^{[90]}$, and starts decreasing from frame $f^{[91]}$ onward down to 0. The relative HO will be eliminated from the MC only when the RI equals 0, which occurs after frame $f^{[180]}$.

The MC continues to return the data related to the last identified target, also after this is not present anymore in the image, for a period that depends on the maximum value achieved by the

RI. If the balloon is recognized in N frames, its position will be the output of the MC for the next $N - RI_{\min}$ frames from its disappearance in the image. Following the previous example, the MC will continue to return the same data from $f^{[91]}$ to $f^{[175]}$ with $RI_{\min} = 5$, even if there is no target in these frames, generating a "Ghost balloon".

This side-effect can be seen both as an advantage and an issue for the navigation. The advantage can be explained as follows: if the well-established target (HO with a very high RI) temporarily disappears from the FoV, e.g., because the piercing tip obstructs the view or light reflections dazzle the camera, the drone can move along the same direction identified with the last saved information as long as the RI is sufficiently high ($RI \geq RI_{\min}$). Remaining in the *Action State*, the probability of a new proper recognition increases along the way and prevents an incorrect maneuver (viz., switching the HLC to the *Searching/Trajectory following State*). In contrast, the drone will continue to navigate towards the balloon after the interaction, although it is no more framed by the camera, especially if the RI is high. This is dangerous because although the velocity components during the *Action State* are limited, the proximity between the drone and the balloon support can cause the two to collide.

In order to balance the positive and the negative effects of the "Ghost balloon" phenomenon, we upper-limited the values that the RI can assume with the $RI_{\max}$ parameter. A too high value implies that the drone continues to navigate "blindly" towards a theoretical target for a considerable time interval. In contrast, a too low value could give the loss of a target due to a temporary issue in the recognition process.

$RI_{\max}$ is correlated to the number of frames processed per second, depending on the acquisition speed of the camera or on the computational power of the on-board computer. Based on our experimental tests in the laboratory, we observed that our system takes about 35 ms to 40 ms (processing time) to acquire and process one image. The difference ($RI_{\max} - RI_{\min}$) is the maximum value of frames containing a "Ghost balloon" and, consequently, the corresponding time delay is the product between this difference and the processing time. These considerations and the experimental evidences of the flight tests suggested us to select $RI_{\max} = 20$. Thus, the difference ($RI_{\max} - RI_{\min}$) becomes equal to 15 that corresponds approximately to 0.5 s, an adequate time to safely fly during all the states of the HLC.

5.2. Optimizing the HLC

The optimization of the HLC was driven by large number of flight trials, using Mission Planner not only to plan the trajectory, based on the waypoints, and to track the drone during the mission, but also to monitor the correct operation of the sensors on board of the Pixhawk.

We performed flight tests by progressively increase their complexity from the scenario and programming code standpoints.

As for the scenario, we initially placed only one target aiming to verify the correct sequence of the operations implemented into the HLC and of the flight commands sent to the LLC. Once this check was considered successful, we placed multiple targets both in the flight area, in order to test the interaction strategy, and outside a defined perimeter to test the safety measures for discarding targets placed, on purpose, in a not-allowed flight zone. In order to test the sensitivity and reliability of our system, we placed the targets randomly and at different heights from the ground. Moreover, since the effective balloon radius was defined just a few days before the Competition (balloon radius of 22.5 mm), we used small targets (common party balloons radius of 12 mm) in order to finely tune the control laws. We performed our tests also with different weather conditions: sunny days were useful to calibrate the RT algorithms to deal with different target luminosity, while windy days helped to optimize the drone configuration and the stability of the LLC and HLC systems. Figure 19 reports some pictures taken during the experimental tests. Concerning the programming code, at an early stage we tested the actual integration of the LLC with MAVROS and SMACH packages due to a simplified HLC, that implemented only the *Searching/Trajectory following State* and the emergency procedure (i.e., the *Return to home State*). Then, we updated the HLC code by adding the *Reaching State* and a fictitious *Action State*, that consisted in hovering in front of the target, at a

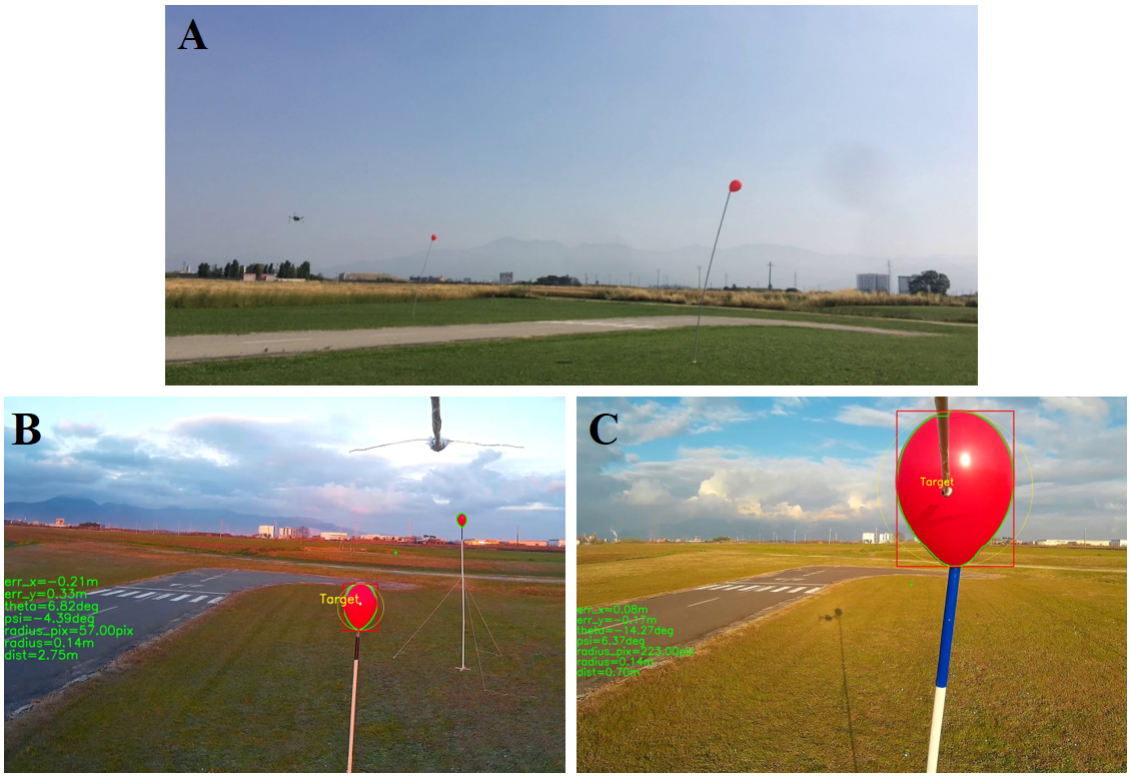


Figure 19. Pictures taken during the experimental tests on the fields. Panel A. Randomly placed balloons during a windy day. Panel B. Targets at different heights framed during the *Reaching State*. Panel C. A target framed just before the interaction during the *Action State*. Note: the depiction of different piercing tip designs depends on the developmental stage of the project. The pictures in panels B-C contains the main numerical features of the locked target (see also Section 3.1.3).

safety distance from it, for about 30 s, since the Competition regulation did not define the interaction phase yet. At the later stage, after the publication of the final version of the Competition regulation, we tested the full state machine of the HLC, including the control laws to approach and interact with (aka pierce) the targets, considering a further corrective experimental factor (δ) for the vertical position of the target in order to exactly center the piercing tip with respect to the balloon.

During all these steps, we were able to analyze and test progressively the updates introduced in the HLC code in order not only to optimize the sequence of all flight phases, but also to select the appropriate parameters for the control laws, as described in Section 3.2.1. The final coefficients are listed in Table 1.

We found these on-field trials important to deal with phenomena difficult to simulate in laboratory (e.g., environmental conditions) and also to discover and solve issues that were not being considered during the design phases and lab tests. An example is the key role of both of the vertical and horizontal alignments to the target that, in case of a wrong positioning, may lead to potential safety issues (i.e., collisions of the drone with the target holder). It is important to stress that, as mentioned earlier, all the tests on the field were carried out with small balloons, thus assessing the reliability and efficiency of our methodology. Moreover, the balloons during the tests were linked to flexible poles and were free to move following the wind. In such circumstances, in the presence of wind with speeds between 10 km/h and 20 km/h, the algorithm generated greater errors when estimating the distance of the target, up to 50 cm depending on the different orientation of the balloon due to the

Table 1. Parameters from the experimental sessions on the field.

Parameter	Symbol(s)	Value
Lowest velocities implemented by the LLC	$v_{x_{\min}}$, $v_{y_{\min}}$, $v_{z_{\min}}$	$0.11\,\mathrm{m\,s^{-1}}$
Highest/lowest velocity imposed by the control algorithm	$\lvert v\rvert_{min/max}$	$2.5\,\mathrm{m\,s^{-1}}$
Minimum distance from a target during the Reaching State	$d_{\min}$	1.5 m
Maximum selected distance from a target to enable the Reaching State	$d_{\max}$	15 m
Proportional Coefficient for Y-control law	k_y	$0.35\,\mathrm{s^{-1}}$
Lowest errors in pixel along the x-axis for the camera	$\delta^{px}_{T_{x_{\min}}}$	192 px
Highest errors in pixel along the x-axis for the camera	$\delta^{px}_{T_{x_{\max}}}$	810 px
Lowest yaw rate during the Reaching State	$\dot{\psi}_{\min}$	$0.25\,\mathrm{rad\,s^{-1}}$
Highest yaw rate during the Reaching State	$\dot{\psi}_{\max}$	$1\,\mathrm{rad\,s^{-1}}$
Maximum vertical distance between the drone and the target imposed by the control algorithm during the Reaching State	$\delta_{T_{y_{\max}}}$	3 m
Tolerance tuned during the experimental tests for the min/max height	η	0.15
Proportional Coefficient for X-control law tuned during the experimental tests for the Action State	k^a_x	$0.5\,\mathrm{s^{-1}}$
Minimum distance at which the drone interacts with the target during Action State	$d^a_{\min}$	0.7 m
Highest X-velocity imposed by the control algorithm during Action State	$v_{x_{\max}}$	$0.2\,\mathrm{m\,s^{-1}}$
Lowest yaw rate during the Action State	$\dot{\psi}_{\min}$	$0.1\,\mathrm{rad\,s^{-1}}$
Highest yaw rate during the Action State	$\dot{\psi}_{\max}$	$0.4\,\mathrm{rad\,s^{-1}}$
Vertical Error during the Action State (occurred during tests)	-	3 cm
Tolerance for the Geo-fence distance tuned during flight tests	Δd	1 m
Minimum distance from a boundary to enable the Geo-fence State tuned during flight tests	d_{i_T}	2 m
Threshold for the distance value limit near a Geofence boundary tuned during flight tests	d_{lim}	5 m

wind. In these cases, the success rate for the interactions was about 80%. Regarding the results achieved during the tests on the field for the final tuning of the control system, we achieved an almost 95% rate of successful results, including detection, reaching and interaction, independently of the lighting conditions and with wind speed up to 10 km/h (i.e., a total of 19 popped balloons in 5 flights with 4 balloons randomly-positioned inside the field during one day). These results were achieved also due to earlier tests performed in the laboratory without external disturbances and noises (i.e., change of light or environment conditions) aimed at accurately setting the parameters of the vision algorithm. During these lab tests we achieved a 100% of successful and stable detection of the balloon at distance equal to or less than 15 m.

6. The MBZIRC Challenge

We participated to Challenge 1 of the 2020 MBZIRC Competition in Abu Dhabi, achieving the fourth place over 22 teams. The Challenge required the accomplishment of two tasks: first two drones had to autonomously detect and interact (aka pierce) with five randomly positioned fixed targets (Figure 20), while, to complete the Challenge, the drones had to track and capture a mobile target overflying the arena.

We assembled two autonomous drones following the design strategies described above. We planned our strategy by setting a sequential activation of the platforms consisting on the second drone taking off after the landing of the first one, in order to deal with the targets missed in first instance. This

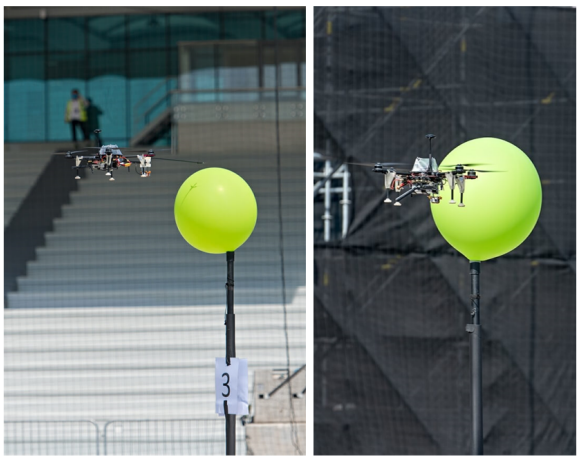


Figure 20. Drone approaching the targets during the 2020 MBZIRC Competition.

plan would have also helped avoiding any potential collision between the two platforms. According to this strategy, we were able to fully accomplish the first task of the Challenge in ∼365 s (including take-off/landing manoeuvres and a final check by overflying the arena). This result prevented the use of the second drone, saving additional time. As for the second task related to the flying target, we decided to compete in manual mode, collecting additional points thanks to our experienced pilot. Eventually, we scored the same amount of points as of other teams, but we did better than others by accomplishing the same tasks in a less span of time.

7. Lesson learned

In this Section we summarize the main lessons learned in the MBZIRC competition, in order to give the readers a tool set for the implementation of such strategies and solutions in applications not limited to the goals of the challenge.

The first important aspect concerns the structure of the drone. Most drones are commercialized as "closed" packages in which both hardware and software components cannot be customized for specific applications (e.g., a competition). Aiming at competing to MBZIRC 2020, we decided to develop a modular platform able to be quickly customized to face unexpected last-minute situations.

From a hardware standpoint, we wanted to fabricate a modular platform by integrating dedicated designed modules to a commercial drone structure. First, we designed four legs made of Acrylonitrile Butadiene Styrene (ABS), manufactured with our in-house 3D printer (Figure 3b) and carbon tubes, in order to replace the original two, retractable, T-shaped supports of the commercial drone (Figure 3a). Additionally, we replaced the original arms joints made of plastic with new optimized aluminium rigid clamps. These two structures enabled a stability during autonomous flight maneuvers, such as take-off and landing, and under unfavorable weather conditions (e.g., wind gusts) or bumpy ground surfaces.

Concerning the specific task related to the interaction with the targets, we decided to endow the drone with a piercing tip with three sharp needles at the end, whose high aspect ratio facilitated the success of the bursting action. From a static balance standpoint, the addition of a cantilever beam carrying both piercing tip and camera moved the drone's center of gravity forward and required a compensating relocation of the electronic components and battery as a counterweight. Compared to other design choices made by other teams (e.g., cuneiform appendices to attack targets), this simple and cheap solution proved efficient and allowed our team to successfully accomplish the task, while saving time.

From the software standpoint, we achieved a significant advancement by simulating the control environment, thereby approaching the multi-thread architecture through separated subsystems. This *divide and conquer* methodology helped to thoroughly debug our code for the demands of Challenge applications. In this respect, the software development was significantly simplified by the simulation tools described in Section 4: this process allowed us both to test new system features and to detect and fix several issues related to the RT and the HLC. In particular, using the simulators, the HLC code was well debugged before the on-field tests, eliminating most of the code performance issues and to reduce the probability of crashes. Later, experimental sessions on the field offered a final opportunity to make the software reliable and adaptable to a large array of boundary conditions that were difficult or impossible to simulate. Moreover, the multi-thread structure gives the system a great versatility: the HLC can operate with inputs from a different vision algorithm and the RT code can run with a different control software architecture or on different hardware.

The experimental sessions were crucial also for finding out some challenges peculiar to the competition scenario. A first issue to overcome was the uneven availability of the GPS/Wi-Fi signal across the arena. This particular aspect increased the level of difficulty for meeting the envisaged goal. As also reported in other previous studies (Bejuri et al., 2011; Laoudias et al., 2018; Stephenson et al., 2013), the partial availability of GPS or network signals and their low quality is one of the main challenges that the scientific community is working on in view of robotic implementations in underdeveloped areas or areas with disturbances (i.e., buildings and iron structures). We found the GPS sensor connected to the LLC particularly inaccurate, confirming what was also outlined in (Wang et al., 2008). We noted during the experiments on the field, for example, that the drone was flying in different positions than those *a priori* set by the waypoints and implemented in the HLC. Since we expected a similar, and possibly worse, situation during the competition, we fixed this issue by increasing the number of waypoints and by experimentally tuning our code to reduce the effects of the GPS errors. In particular, concerning the *Geofence State*, a state that demanded a high level of precaution, safety distances from the boundaries were increased by an additional tolerance (see Table 1). In contrast, an alternative solution could have been the employment of a real-time kinematic (RTK) satellite system, as other teams did following examples in literature (Bähnemann et al., 2019; Spurný et al., 2019; Stephenson et al., 2013). However, we decided not to implement such a solution and, on the contrary, to optimize our vision and navigation codes, to avoid a score penalization (MBZIRC, 2020). Moreover, by planning a strategy involving a sequential order of active drones, we made the accomplishment of the Challenge much easier and safer, preventing potential inter-platform collisions due to Wi-Fi losses or bad GPS signals.

A second software challenge concerned the reliability of the FCB on-board sensors and, particularly, of the altitude measurement computed by the LLC using the embedded barometer as primary reference. We noticed unwanted fluctuations that led to a substantial instability of the altitude position of the drone. Although this situation may appear marginal for standard flight manoeuvres, it deeply impacts the alignment of the drone with the target, potentially compromising a successful interaction. We solved this problem by connecting the LightWare SF20/C to the LLC, an additional sensor that provided a reliable reference altimeter at altitudes up to 25 m, under differing light conditions, and over varying different ground surfaces (grass, clods, asphalt). The efficacy of this solution led to abandoning the on-board barometer for the altitude computation.

Concerning the vision strategies, we developed an approach that demonstrated adaptability to the progressive unveiling of the competition rules. We designed our code to be reliable independently of the target features (e.g., size, color, shape) and, as previously mentioned in Section 5.1, the introduction of the Memory Class (MC) improved the accuracy and the stability of the drone during the approach and the interaction with the target. Specifically, we tested our code with targets smaller than those of the challenge and, thanks to the experiments on the field, with highly variable weather conditions that influenced the balloons movement and light condition. This methodology definitely successful, since the milder conditions experienced at the Challenge (viz., no wind gusts and larger targets) that led to the accomplishment of the tasks in a reduced amount of time and without the employment of a second drone.

8. Conclusions

In this paper we describe the system developed to participate to Challenge 1 of the 2020 MBZIRC Competition. Our team consisted of both specialists in robotics and computer science and Master students in Automation and Aerospace Engineering from Scuola Superiore Sant'Anna (Pisa, Italy). The competition was a remarkable opportunity for students to make an effective contribution to a real-world project, a learning experience to understand the importance of exploiting the trans-field synergies in a team, and an opportunity to compete with worldwide students coming from different countries. Moreover, the MBZIRC 2020 competition led to opening a new research line, dealing with aerial robotics, in our own laboratory.

During the preparation for the Challenge, we designed and developed a drone to find and attack randomly-positioned colored targets. We presented the main hardware/software design choices to successfully accomplish the challenge task, describing the process and the issues faced during the implementation in virtual and experimental scenarios.

We wanted to demonstrate how it possible to develop a modular system able to reliably perform complex tasks in unstructured outdoor scenarios. This aspect is crucial for adapting the system to the working conditions and for testing separately individual functionalities, uncovering possible faults or deficiencies. The robustness of our system largely depends, indeed, on the analysis of the failures encountered during the developmental stages.

This is, indeed, the key novelty of our work: a successful approach that exploits the synergistic use of vision and control algorithms and customized hardware solutions for a challenging application whose methodologies and outcomes can be also used as a helpful toolkit for other implementations not directly related to the competition goals. As a demonstration of this flexibility, the same hardware/software system architecture has been repurposed, with relatively few customizations in the processing thread, as an autonomous, aerial, robotic aerial platform for scanning archaeological ruins in the context of cultural heritage projects.

Acknowledgments

This project was supported by the Mohamed Bin Zayed International Robotics Challenge 2020 (www.mbzirc.com) organized by Khalifa University, Abu Dhabi, UAE. The authors want to acknowledge our experienced pilots Mr. Claudio Lattanzi and Mr. Andrea Capanna, and Dr. Roberto Lazzarini, Dr. Nicodemo Funaro, and Mr. Gabriele Favati for their technical support.

ORCID

Mario Milazzo https://orcid.org/0000-0001-8429-5544
Stefano Roccella https://orcid.org/0000-0002-2738-8821

References

Abughalieh, K. M., Sababha, B. H., & Rawashdeh, N. A. (2019). A video-based object detection and tracking system for weight sensitive UAVs. *Multimedia Tools and Applications*, *78*(7), 9149–9167. https://doi.org/10.1007/s11042-018-6508-1

ArduPilot. (2019). *Arducopter* [Apparatus]. Retrieved September 24, 2020, from https://ardupilot.org/copter/index.html

ArduPilot Dev Team. (2020). *SITL simulator (Software in the Loop)* [Computer software]. Retrieved September 24, 2020, from https://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html

Bachmann, R. J., Boria, F. J., Vaidyanathan, R., Ifju, P. G., & Quinn, R. D. (2009). A biologically inspired micro-vehicle capable of aerial and terrestrial locomotion. *Mechanism and Machine Theory*, *44*(3), 513–526. https://doi.org/10.1016/j.mechmachtheory.2008.08.008

Bähnemann, R., Pantic, M., Popović, M., Schindler, D., Tranzatto, M., Kamel, M., Grimm, M., Widauer, J., Siegwart, R., & Nieto, J. (2019). The ETH-MAV team in the MBZ International Robotics Challenge. *Journal of Field Robotics*, *36*(1), 78–103. https://doi.org/10.1002/rob.21824

Baloch, G., & Gzara, F. (2020). Strategic network design for parcel delivery with drones under competition. *Transportation Science*, *54*(1), 204–228. https://doi.org/10.1287/trsc.2019.0928

Bejuri, W. M. Y. W., Mohamad, M. M., & Sapri, M. (2011). Ubiquitous positioning: A taxonomy for location determination on mobile navigation system. *Signal & Image Processing: An International Journal*, *2*(1), 24–34. https://doi.org/10.5121/sipij.2011.2103

Beul, M., Nieuwenhuisen, M., Quenzel, J., Rosu, R. A., Horn, J., Pavlichenko, D., Houben, S., & Behnke, S. (2019). Team NimbRo at MBZIRC 2017: Fast landing on a moving target and treasure hunting with a team of micro aerial vehicles. *Journal of Field Robotics*, *36*(1), 204–229. https://doi.org/10.1002/rob.21817

Chan, W. P., Mizohana, H., Chen, X., Shiigi, Y., Yamanoue, Y., Nagatsuka, M., & Inaba, M. (2019). Multimodal sensing and active continuous closed-loop feedback for achieving reliable manipulation in the outdoor physical world. *Journal of Field Robotics*, *36*(1), 17–33. https://doi.org/10.1002/rob.21818

Coskun, M., & Ünal, S. (2016). Implementation of tracking of a moving object based on camshift approach with a UAV. *Procedia Technology*, *22*, 556–561. https://doi.org/10.1016/j.protcy.2016.01.116

Dang-Khanh Le, T.-K. N. (2015). A study on the modeling of a hexacopter. *Journal of the Korean Society of Marine Engineering*, *39*(10), 1023–1030. https://doi.org/10.5916/jkosme.2015.39.10.1023

Digiacomo, F., Afroz, A. S., Pelliccia, R., Inglese, F., Milazzo, M., & Stefanini, C. (2020). Head-mounted standalone real-time tracking system for moving light-emitting targets fusing vision and inertial sensors. *IEEE Transactions on Instrumentation and Measurement*, *69*(11), 8953–8961. https://doi.org/10.1109/tim.2020.2995233

Elliott, A. (2016). *Build your own drone manual: The practical guide to safely building, operating and maintaining an Unmanned Aerial Vehicle (UAV)*. Haynes Publishing.

Floreano, D., Zufferey, J.-C., Srinivasan, M. V., & Ellington, C. (Eds.). (2010). *Flying insects and robots*. Springer-Verlag. https://doi.org/10.1007/978-3-540-89393-6

Fum, W. Z. (2015, September). *Implementation of simulink controller design on Iris+ quadrotor* (Master's thesis). Naval Postgraduate School. Monterey, California.

Glock, K., & Meyer, A. (2020). Mission planning for emergency rapid mapping with drones. *Transportation Science*, *54*(2), 534–560. https://doi.org/10.1287/trsc.2019.0963

Gupta, S. G., Ghonge, M., & Jawandhiya, P. M. (2013). Review of unmanned aircraft system (UAS). *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, *2*(4). https://doi.org/10.2139/ssrn.3451039

Hassanalian, M., Abdelkefi, A., Wei, M., & Ziaei-Rad, S. (2017). A novel methodology for wing sizing of bio-inspired flapping wing micro air vehicles: Theory and prototype. *Acta Mechanica*, *228*(3), 1097–1113. https://doi.org/10.1007/s00707-016-1757-4

jDrones. (2016). *Ardupilot* [Apparatus]. Retrieved September 24, 2020, from https://ardupilot.org/

Kannerworff, L. (1992). *Progettiamo gli aeromodelli*. Aero Club d'Italia - Roma.

Khalilov, J., & Kutay, A. T. (2015). Interfacing Matlab/Simulink with V-REP for a controller design for quadrotor. *International Journal of Engineering Research and Reviews*, *3*(4), 42–49.

Koubaa, A. (Ed.). (2017). *Robot Operating System (ROS) The Complete Reference - Volume 2* (Vol. 707). Springer. https://doi.org/10.1007/978-3-319-54927-9

Laoudias, C., Moreira, A., Kim, S., Lee, S., Wirola, L., & Fischione, C. (2018). A survey of enabling technologies for network localization, tracking, and navigation. *IEEE Communications Surveys & Tutorials*, *20*(4), 3607–3644. https://doi.org/10.1109/comst.2018.2855063

Leuenberger, D., Haefele, A., Omanovic, N., Fengler, M., Martucci, G., Calpini, B., Fuhrer, O., & Rossa, A. (2020). Improving high-impact numerical weather prediction with lidar and drone observations. *Bulletin of the American Meteorological Society*, *101*(7), E1036–E1051. https://doi.org/10.1175/bams-d-19-0119.1

Mavros-MavLink. (2019). *MAVROS - MAVLink extendable communication node for ROS* [Computer software]. Retrieved September 24, 2020, from http://wiki.ros.org/mavros

MBZIRC. (2020). *Mohamed Bin Zayed International Robotics Challenge*. Retrieved September 25, 2020, from http://www.mbzirc.com/

Müller, M. (2020). *eCalc - the most reliable RC Calculator on the Web V2.02* [Computer software]. Retrieved September 24, 2020, from https://www.ecalc.ch/xcoptercalc.php

Nonami, K., Kendoul, F., Suzuki, S., Wang, W., & Nakazawa, D. (2010). *Autonomous flying robots: Unmanned aerial vehicles and micro aerial vehicles*. Springer Japan. https://doi.org/10.1007/978-4-431-53856-1

Pantalone, M. (2015). *Modellazione e simulazione di un quadricottero multirotore* (Master's thesis). Alma Master Studiorum, Università di Bologna, Scuola di Ingegneria e Architettura - Sede di Forlì, corso di Laurea in Ingegneria Aerospaziale. Bologna, Italy.

Persico, S. (2015, December). *Controllo visuale di un sistema robotico per la raccolta della frutta* (Master's thesis). Politecnico di Milano, Scuola di Ingegneria Industriale e dell'Informazione, Corso di Laurea Magistrale in Ingegneria Meccanica. Milan, Italy.

Sidea, A.-G., Brogaard, R. Y., Andersen, N. A., & Ravn, O. (2014). General model and control of an n rotor helicopter. *Journal of Physics: Conference Series*, *570*(052004). https://doi.org/10.1088/1742-6596/570/5/052004

SMACH. (2019). *SMACH, ROS-independent Python library* [Computer software]. Retrieved September 24, 2020, from http://wiki.ros.org/smach

Solem, J. E. (2012, July). *Programming computer vision with Python: Tools and algorithms for analyzing images*. O'Reilly Media, Inc, USA. https://www.ebook.de/de/product/18609832/jan_erik_solem_programming_computer_vision_with_python.html

Spurný, V., Báča, T., Saska, M., Pěnička, R., Krajník, T., Thomas, J., Thakur, D., Loianno, G., & Kumar, V. (2019). Cooperative autonomous search, grasping, and delivering in a treasure hunt scenario by a team of unmanned aerial vehicles. *Journal of Field Robotics*, *36*(1), 125–148. https://doi.org/10.1002/rob.21816

Stephenson, S., Meng, X., Moore, T., Baxendale, A., & Edwards, T. (2013). Network RTK for intelligent vehicles: Accurate, reliable, available, continuous positioning for cooperative driving. *GPS World*, *24*(2), 61–67.

Stevens, M. N., & Atkins, E. M. (2016). Multi-mode guidance for an independent multicopter geofencing system. *16th AIAA Aviation Technology, Integration, and Operations Conference*. https://doi.org/10.2514/6.2016-3150

Suzuki, S., & Abe, K. (1985). Topological structural analysis of digitized binary images by border following. *Computer vision, graphics, and image processing*, *30*(1), 32–46. https://doi.org/10.1016/0734-189x(85)90136-7

Szeliski, R. (2010). *Computer vision: Algorithms and applications*. Springer. https://doi.org/10.1007/978-1-84882-935-0

Tzoumanikas, D., Li, W., Grimm, M., Zhang, K., Kovac, M., & Leutenegger, S. (2019). Fully autonomous micro air vehicle flight and landing on a moving target using visual–inertial estimation and model-predictive control. *Journal of Field Robotics*, *36*, 49–77. https://doi.org/10.1002/rob.21821

Vanin, M. (2013, May). *Modeling, identification and navigation of autonomous air vehicles* (Master's Degree Project). Automatic Control School of Electrical Engineering, Kungliga Tekniska Högskolan. Stockholm, Sweden.

Wang, J., Garratt, M., Lambert, A., Wang, J. J., Han, S., & Sinclair, D. (2008). Integration of GPS/INS/Vision sensors to navigate unmanned aerial vehicles. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *37*(B1), 963–970.

Zhu, P., Wen, L., Du, D., Bian, X., Hu, Q., & Ling, H. (2020). *Vision meets drones: Past, present and future*. arXiv: 2001.06303 [cs.CV].

Zufferey, J.-C. (2008, April). *Bio-inspired flying robots: Experimental synthesis of autonomous indoor flyers*. EPFL Press, distributed by CRC Press, USA. https://doi.org/10.1201/9781439808115

How to cite this article: Cascarano, S., Milazzo, M., Vannini, A., Spezzaneve, A., & Roccella, S. (2021). Design and development of drones to autonomously interact with objects in unstructured outdoor scenarios. *Field Robotics*, *1*, 127–157.

Publisher's Note: Field Robotics does not accept any legal responsibility for errors, omissions or claims and does not provide any warranty, express or implied, with respect to information published in this article.