Noname manuscript No.
(will be inserted by the editor)

MAT-Fly: an educational platform for simulating Unmanned Aerial Vehicles aimed to detect and track moving objects

Giuseppe Silano · **Luigi Iannelli**

Received: date / Accepted: date

Abstract The main motivation of this work is to propose a simulation approach for a specific task within the UAV (Unmanned Aerial Vehicle) field, i.e., the visual detection and tracking of arbitrary moving objects. In particular, it is described MAT-Fly, a numerical simulation platform for multi-rotors aircraft characterized by the ease of use and control development. The platform is based on Matlab® and the MathWorks™ Virtual Reality (VR) and Computer Vision System (CVS) toolboxes that work together to simulate the behavior of a drone in a 3D environment while tracking a car that moves along a non trivial path. The VR toolbox has been chosen due to the familiarity that students have with Matlab and because it allows to move the attention to the classifier, the tracker, the reference generator and the trajectory tracking control thanks to its simple structure. The overall architecture is quite modular so that each block can be easily replaced with others by simplifying the development phase and by allowing to add even more functionalities.

The simulation platform makes easy and quick to insert and to remove flight control system components, testing and comparing different plans when computer vision algorithms are in the loop. In an automatic way, the proposed simulator is able to acquire frames from the virtual scenario, to search for one or more objects on which it has been trained during the learning phase, and to track the target position applying a trajectory control addressing what is well-known in the literature as an image-based visual servoing problem.

Some simple testbeds have been presented in order to show the effectiveness and robustness of the proposed approach as well as the platform works. We released the software as open-source, making it available for educational purposes.

Keywords image-based visual servoing · 3D simulation scenario · MATLAB/Simulink · multi-rotors · educational · vision detection and tracking · UAV · software-in-the-loop · computer vision · trajectory control

Giuseppe Silano · Luigi Iannelli
Department of Engineering,
University of Sannio in Benevento,
Piazza Roma, 21; 82100 Benevento (Italy)
E-mail: {giuseppe.silano, luigi.iannelli}@unisannio.it

1 Introduction

Unmanned Aerial Vehicles (UAVs), although originally designed and supported for defense and military purposes (e.g., aerial attacks or military air covering), in the recent years gained an increasing interest and attention related to civilian use. Nowadays, UAVs are employed for several tasks and services like surveying and mapping [1], for spatial information acquisition and buildings inspection [2], data collection from inaccessible areas [3], agricultural crops and monitoring [4], manipulation and transportation or navigation purposes [5].

Many existing algorithms for the autonomous control [6] and navigation [7] are provided in the literature, but it is particularly difficult to make the UAVs able to work autonomously in constrained and unknown environments or also indoors. Thus, it follows the need for tools that allow to understand what it happens when some new applications are going to be developed in unknown or critical situations. Simulation is one of such helpful tools, widely used in robotics [8,9], whose main benefits are costs and time savings, enabling not only to create various scenarios, but also to carry out and to study complex missions that might be time consuming and risky in real world. Finally, bugs and mistakes in simulation cost virtually nothing: it is possible to crash a vehicle several times and thereby getting a better understanding of implemented methods under various conditions. Thus, simulation environments are very important for fast prototyping and educational purposes, although they may have some drawbacks and limitations, such as the lack of noisy real data or the fact that simulated models are usually incomplete or inaccurate. Despite the limitations, the advantages that the simulation provides are more, as like as to manage the complexity and heterogeneity of the hardware and applications, to promote the integration of new technologies, to simplify the software design, to hide the complexity of low-level communication [10].

Different solutions, typically based on external robotic simulators such as Gazebo [11], V-REP [12], AirSim [13], MORSE [14], are available to this aim. They employ recent advances in computation and computer graphics (e.g., AirSim is a photorealistic environment [7]) in order to simulate physical phenomena (gravity, magnetism, atmospheric conditions) and perception (e.g., providing sensor models) in such a way that the environment realistically reflects the actual world. In some cases those solutions do not have enough features that could allow to create large scale complex environments close to reality. On the other hand, when the tools provide such possibilities, they are difficult to use or they require high computing capabilities [13]. Definitely, it comes out that simulating the real world is a non trivial task, not only due to multiple phenomena that need to be modeled, but also because their complex interactions ask the user a notable effort for the learning and development phase. For all such reasons, a complete software platform that makes possible to test different algorithms for UAVs moving in a simulated 3D environment is more and more important both for the whole design process and educational purposes.

In this paper it will be presented a software architecture through which detection, tracking and control algorithms can be verified and validated all together in a 3D graphical environment. Due to the simple implementation and the limited possibility of interfacing it with dedicated tools, the proposed software architecture should be meant with an educational purpose, although it can be considered also as a starting point for the development of a software-in-the-loop (SIL) platform (see [6,15]) in the UAV framework. Furthermore, as highlighted in [16], the interactive learning approaches allow students to improve their technical knowledge and communication skills, giving them the experience of what they will encounter in a real world environment. Therefore, the platform can be seen as potentially endless: students, researchers and developers can expand the simulator functionalities by

modifying or integrating new agents dynamics, control algorithms or detection and tracking techniques for their own purposes.

In our paper the specific domain of interest regards the behavior of UAVs acting in accordance with the Image-Based Visual Servoing (IBVS) approach [17,18]. Most are the applications that use UAVs [19,20], either teleoperated remotely by a pilot (who decides the next action by looking at the camera images) from a Ground Control Station (GCS) or autonomously by following preprogrammed operations via onboard sensors, e.g., for sending data and video signals towards a Central Data Station (CDS) which stores the information for later processing. The availability of low-weight devices (e.g., Parrot AR.Drone 2.0 or more recent Parrot Bebop 2.0[1]), with low power consumption, makes cameras among the most suitable sensors, especially in GPS-denied environments [21].

Within the field of visual servoing, the camera configuration is distinguished in two main classes: *eye-in-hand* and *eye-to-hand*. In the first class the camera is rigidly attached to the UAV frame while in the second one the camera has a fixed orientation in the workspace thanks to some specific devices, e.g., a gimbal [22]. However, when the camera is placed on a mobile platform it can be rotated through a manipulator arm and such a third configuration is called *onboard-eye-to-hand* [23]. In this paper, the eye-in-hand configuration was adopted[2] for the drone using the ideal pinhole camera model [24] to emulate the camera in the simulation scenario.

The application we consider, that is an extension of our previous work [25], has been revised for making the UAV able to detect and track a specific object (a car) moving along a non trivial path. Compared to our previous work, a tracking algorithm has been added in the loop: the classifier is used to detect the target only at the first step or in case of a partial occlusion. Apart from such situations, a Continuously Adaptive Mean-Shift (CAMShift) tracking algorithm [26] is employed to follow the car along the path, thus reducing the computational burden and the possibility to lose the target during the tracking. Moreover, in this paper it has been proposed a novel procedure based on *ad hoc* Matlab scripts that automatically select the bounding box area (aka ROI, Region of Interest) of the target (the car) in each image, thus avoiding to pass through specific Matlab tools like *Training Image Labeler*. Those scripts also compare the performance of various classifiers selecting in an automatic way the best one among different features types (i.e., Haar, HOG, LBP) [27] and different number of training stages. Simple testbeds have been considered to analyze the effectiveness and the robustness of the platform. Finally, we published the software architecture as open-source [28] with the aim to share our result with other researchers and students that might use the platform for testing their algorithms and understanding how different approaches can improve the system performance, especially for educational purposes.

The paper is organized as follows. Section 2 explains the 3D simulation scenario and its functionalities. The classifier learning phase and the vision based target detection algorithm are presented in Sec. 3 and 4, respectively. Section 5 briefly describes the UAV model while numerical results and control algorithms are reported in Sec. 6. Finally, Section 7 concludes the paper.

[1] Lightweight commercial UAV platforms that counts with a 14 Mega pixels fisheye lens camera and a simple to use Software Development Kit (SDK).

[2] It is in line with many commercial Micro Aerial Vehicles (MAVs) devices such as the above mentioned Parrot devices.

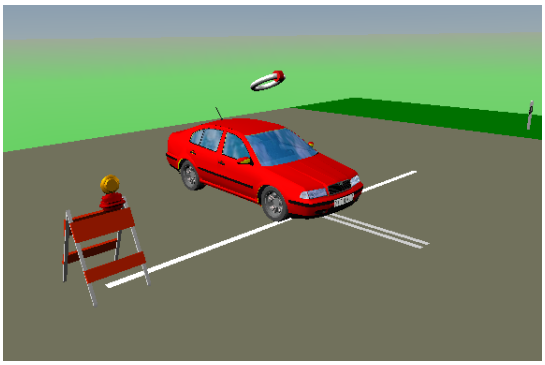


Fig. 1 Initial frame of the considered scenario in the paper. The visualization of forces and other similar features have been disabled in order to improve the classifier performance. While, the current steering angle visualizer has been hold to monitor the car movements along the path.

2 System Description

In order to simulate a scenario as much similar as to the real world, the Matlab Virtual Reality (VR) Toolbox has been used. The toolbox allows to visualize, in a 3D environment, how complex dynamic systems behave when they interact with the surrounding scenario. Furthermore, thanks to animation recording functionalities, frames and videos from the scene can be acquired and used to implement an image based visual problem. Also, the tool makes it easy to add external observers (modeled as viewpoints) to monitor any moving object from different positions and orientations.

From such perspective, the tool has been used for simulating the interaction of a drone following a car. We started from one of the examples[3] available on the MathWorks platform (specifically the *vr_octavia_2cars* example) that describes a quite detailed dynamical model of a car moving along a non trivial path (see, Fig. 1). The example represents a standard double-lane-change maneuver conducted in two-vehicles configuration, where one engages the Electronic Stability Program (ESP) control while the other switches off such control unit when changing lane. From this, a simpler scenario was considered by removing one of the two vehicle configurations, i.e., the car without the ESP controller.

Then, an external observer has been added to the scheme for simulating the behavior of a drone that flies in the 3D scenario observing the car moving along the path. In Matlab VR an external observer has six Degrees of Freedom (DOFs): the spatial coordinates x, y, z, and the angles *yaw* (ψ), *pitch* (θ) and *roll* (φ). The whole process is the following: images are updated according to the position and the orientation of the Matlab VR external observer (the UAV) w.r.t. the car; such images are acquired and elaborated for getting the necessary information to detect and track the object (the car), and to run the control strategy designed for the tracking problem. The outputs of the control algorithm consists of the commands u_φ, u_θ, u_ψ and u_T that should be given to the drone in order to update its position (x_d, y_d and z_d) and orientation (φ_d, θ_d and ψ_d), see Fig. 2.

In Figure 3 the Simulink scheme employed for simulating the drone and the car dynamics is reported. The *esp_on* and the *coordinates_transformation* blocks compute the steering

[3] The list of ready-to-use scenarios that can be easily customized by using any common text editor or 3D creation tool is available in [29].

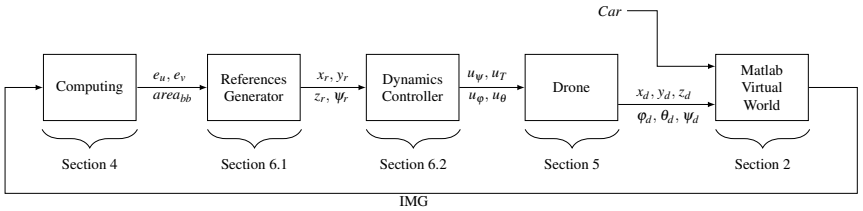


Fig. 2 The control scheme. Subscript d indicates the drone variables, while r indicates the references to the controller. For the reader, each block has been mapped with the section that describes its content.

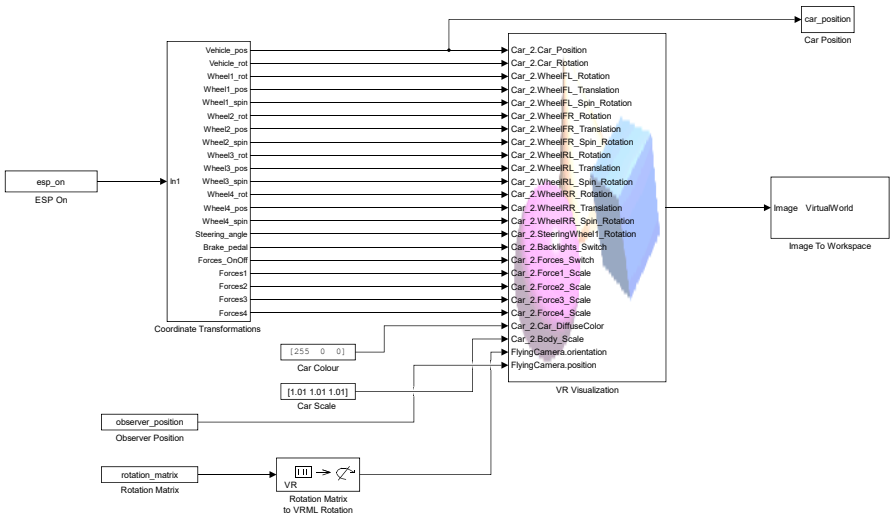


Fig. 3 Simulink scheme employed for simulating the drone and car dynamics in the virtual scenario.

angle, the linear velocity and the position of the car, and all forces needed to follow a given path. While, the *observer_position* and *rotation_matrix* blocks represent the aircraft position and orientation (it is expressed by using the direction cosine matrix [30] and the Rodrigues's formula [31]), respectively. The processed data are sent to the *VR Visualization* block that takes care of the drone and car movements in the simulated scenario.

Note that Matlab VR adopts a reference system slightly different from the classic fixed reference frame O_{FI}. Figure 4 illustrates such difference. In particular, axes are differently oriented and, furthermore, the virtual scenario reference system is centered in the car center of gravity, although the axes orientation is fixed. These differences are taken into account in all elaborations.

Finally, the Simulink scheme saves the current car position (x_{car}, y_{car} and z_{car}), used for comparing the drone and the car trajectories (see, Sec. 6.1), and frames of the virtual scenario observed from the drone point of view. Those frames will be used, as described in the next sections, for pattern recognition.

In Figure 5 the scheme of the overall software platform architecture is depicted. Colors highlight the different parts of the system: the classifier learning phase (in blue), the vision based target detection (in green), the flight control system (in yellow) and the Matlab VR

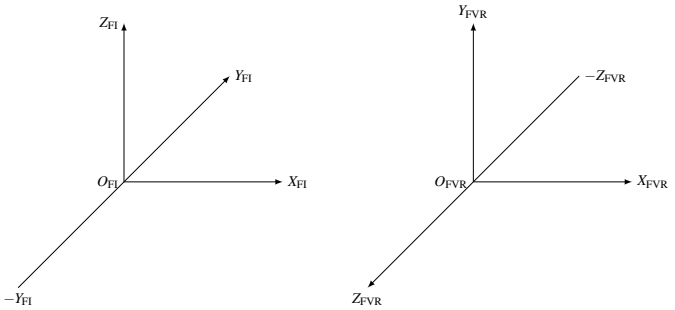


Fig. 4 The picture illustrates the classic fixed frame O_{FI} (left) and the corresponding virtual fixed O_{FVR} (right) reference system.

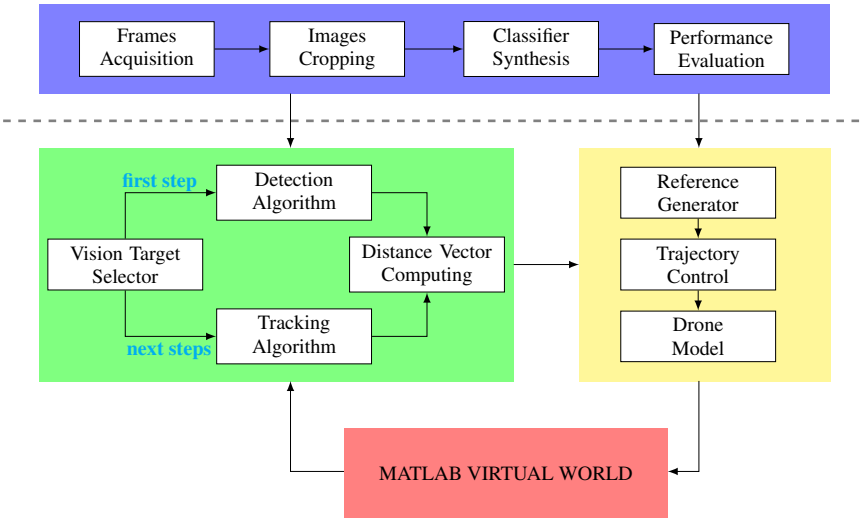


Fig. 5 Proposed software platform architecture. Arrows represent the data exchanged among blocks and how they interact each other.

toolbox (in red). A dashed line is used to separate the classifier learning phase from the rest of the scheme, since it does not take part directly in the simulation although its outputs are employed by the detection algorithm and reference generator for the object tracking (see Secs. 3 and 6).

To simplify the reuse of software components, the entire platform was designed by applying a modular approach: each part (the classifier learning phase, the vision based target detection, etc.) has been divided into smaller ones (e.g., the detection algorithm, the reference generator, the classifier synthesis, etc.), putting some effort in reducing their dependencies and thus making them ready to be used or replaced with others components. In such a way, different computer vision and control algorithms can be combined and tested evaluating their performance and how they can influence the simulation behavior, particularly for educational purposes.

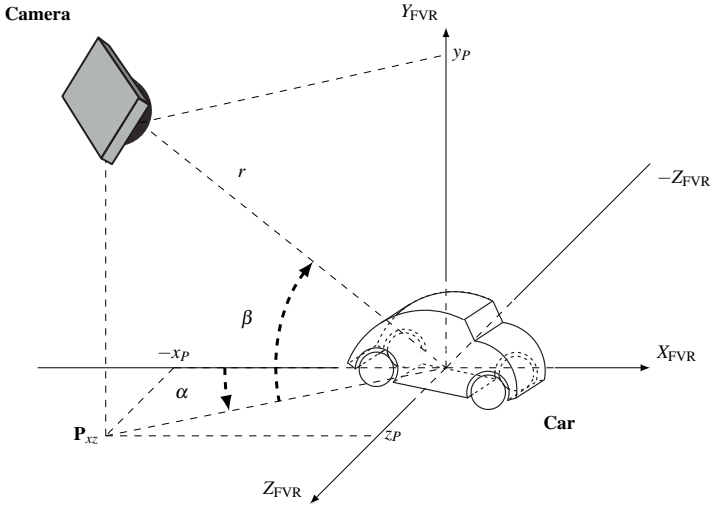


Fig. 6 Drone trajectory around the car parked in its initial state during the frames acquisition.

3 Classifier Learning Phase

The classifier learning phase is the most important part of the system: the object detection and tracking depend on it. Matlab scripts have been developed to automate the entire procedure, from the *frames acquisition* to the *classifier synthesis* and *performance evaluation*. To this aim, the learning process has been split into four parts, as it is depicted in Fig. 5: the frames acquisition, the image cropping, the classifier synthesis and the performance evaluation.

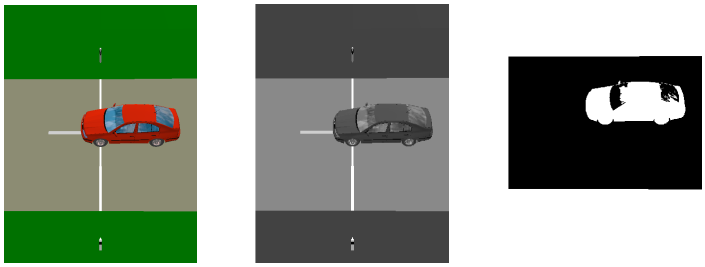
3.1 Frames acquisition

When going to train a classifier, a high number of images are needed. The images are divided into two groups: positive (that contain the target) and negative images. By following as suggested in [31], in particular we used 2626 positive images and 5252 negative images achieving a 1 : 2 ratio.

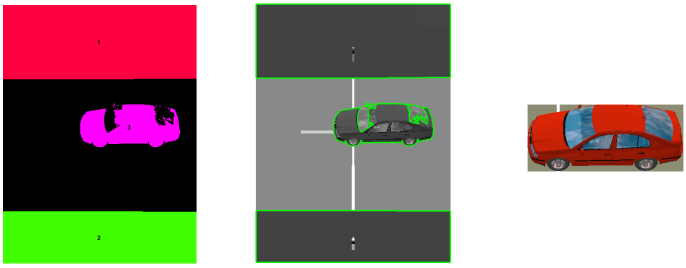
For collecting the images, we simulated the drone moving along a spiral trajectory around the car parked in its initial state (see, Fig. 6). The aircraft attitude and position have been computed for each frame so as described by the sphere surface equations,

$$\begin{cases} y = r\cos\beta \\ x = r\sin\beta\sin\alpha \\ z = r\sin\beta\cos\alpha \end{cases}, \tag{1}$$

where r, the sphere radius, is the distance between the car and the drone (assumed fixed and equal to 15 meters), used as reference for the trajectory generation (see, Sec. 6.1). Whereas, the angles $\alpha \in [0, 2\pi]$ and $\beta \in [0, \pi/2]$ allow to discriminate the drone position and orientation along the surface of the sphere, as depicted in Fig. 6. A video showing the results has been made available at the link `https://youtu.be/A70zed84zv0`.



(a) RGB. (b) Gray scale. (c) Bynary.



(d) Blobs labeled with pseudocolor. (e) Detected blobs. (f) Cropped image.

Fig. 7 Frames obtained by the images computing process. From the RGB file format (a) to the cropped image (f) the steps are reported, sequentially. Each figure is labeled according to the phase during which it has been obtained.

3.2 Image cropping

Following the approach proposed in [32], a Matlab script was developed to automatically select the bounding box area of the car. The image segmentation process was used to simplify and to change the representation: from RGB to binary (Figs. 7(a) and 7(c), respectively), passing through the gray scale (Fig. 7(b)). The result is a set of contours that make the image more meaningful and easier to analyze: each group of pixels in a region is similar w.r.t. some characteristics or computed properties, e.g., color (the red of the car, in our case), intensity, or texture, while adjacent regions are significantly different w.r.t. the same properties, thus allowing to easily detect the target.

To automatically select each group of pixels, the Balanced Histogram Thresholding (BTH) method [33] was chosen. Such method allows to separate the background from the foreground by dividing the image data into two main classes (see, Fig. 8) and by searching for the optimum threshold level.

Starting from the foreground binary image (see, Fig. 7(c)), the script uses the connected-component labeling algorithm [31] with a fixed heuristic (8-connected, in our case) for selecting individual blobs from the image. In Figure 7(d) such blobs are depicted with different colors and numbers. Then, after having extracted all images properties, the script chooses the blob (bounded and overlapped in Fig. 7(e)) that meets criteria in terms of size and intensity (fixed to match the target properties) in order to obtain a unique bounding box surrounding the target. In Figure 7(f) its size has been used for cropping the target within the frame.

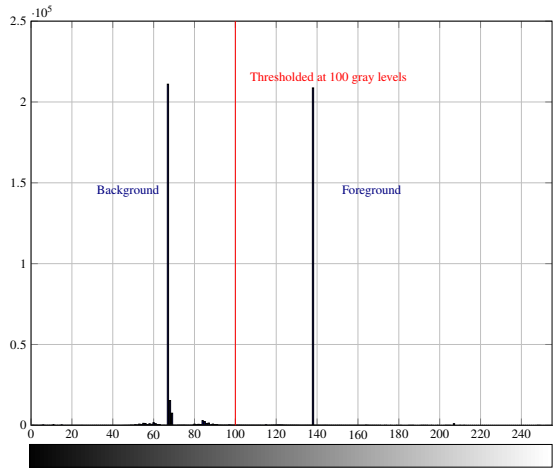


Fig. 8 Histogram of the image data. The red line, the grayscale threshold, divides the graph into two parts: the background and the foreground. The gray gradient bars indicate the associated color to each x-value, from 0 to 255.

Finally, the script makes as output a MAT-file containing, for each positive image, the suitable ROI components, i.e., the bounding box centroids, its width and height. Such file is employed during the classifier synthesis (see, Fig. 5) to design the target detector.

The proposed approach allows to completely and automatically detect the target (the car), decreasing the learning phase time and avoiding to pass through the specific Matlab tool *Training Image Labeler*. In Figure 7, for a single sample frame, all image processing steps, from the acquired frame to the cropped image, are depicted. Such steps allow to better understand how the algorithm works.

For the specific considered image set, the script was able to automatically detect the ROIs with an error of 8.18%: the target was not recognized only for 215 frames out of 2626 positive images, and the first ROI loss appeared after 1791 frames.

3.3 Classifier synthesis

The Viola & Jones algorithm [34] has been used to recognize the car along the path. Although the algorithm was originally designed and developed for face detection problem, it can be trained to detect any object [35] by using different features types (Haar, HOG, LBP) [27] and training stages.

For the considered testbed, the Haar features have been used for designing the classifier. Although Haar and LBP features are often used to detect faces due to their fine-scale textures while the HOG features are often used to detect objects (e.g., peoples and cars), they resulted more useful for capturing the overall shape of the target (see, Fig. 9) even if much longer time was needed during the training phase.

(a) Test image. (b) Haar. (c) Haar corrected. (d) HOG.

(e) HOG corrected. (f) LBP. (g) LBP corrected.

Fig. 9 Detection results obtained by using Haar, HOG and LBP features types. The false alarm and true positive rates has been fixed equal to 0.001 and 0.995, respectively, while the number of training stages was chosen equal to 4. Two different target models were considered: the "corrected" and "uncorrected" version.

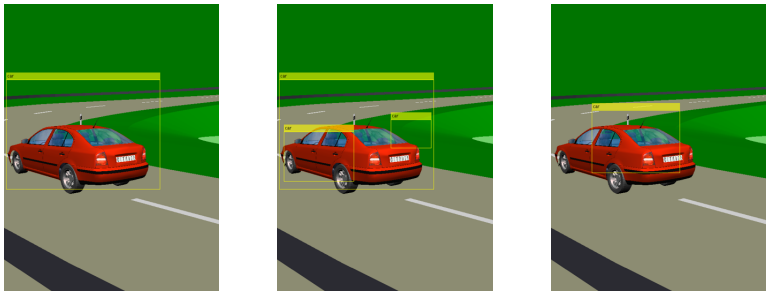
3.4 Performance comparison

In order to choose the best classifier for the proposed testbeds, a Matlab script was developed to analyze the performances for different false alarm and true positive probabilities, number of training stages, and features types. Such script is part of the proposed software platform (see, Fig. 5), and allows quite easily to compare results coming from different classifiers.

Two different models have been considered in designing the detector. The first one uses the ROIs automatically extracted from the virtual scenario, as described in Sec. 3.2, while the second one "corrects" those ROIs through the Matlab tool *Training Image Labeler*.

In Figure 9 the detection results obtained for the considered sample frame by using the Harr, HOG and LBP features type are depicted. In all revelations, the car is only partially detected in spite of the high number of images employed in the learning process. Except in same cases, there are no revelation errors: different bounding boxes are detected in the image. This is probably due to the several image view points used during the learning process and the absence of photorealism in the collected frames. As described in [36], the reality gap (i.e., realistic geometry, textures, lighting conditions, camera noise and distortion) direct influences the performances of computer vision algorithms. On the other hand, they introduce enough "useful noise" to help the detection. Many tests have been conducted in order to assess the true performance of the classifier.

As shown in Figs. 9(b) and 9(c), the detection results are very similar for both models. Thus, it is a good approximation to consider the "uncorrected" ROIs instead of the "corrected" version in the classifier design. Such approximation allows to save time during the training phase thus avoiding to pass through the Matlab tool *TrainingImageLaebeler*.



(a) Maximum bounding box.

(b) All bounding boxes.

(c) Average bounding box.

Fig. 10 Bounding box selection algorithm. The detection results are obtained by using the Haar cascade features type. The maximum (left) and average (right) bounding boxes are computed by using the result obtained from detection (center).

4 Vision Based Target Detection

The vision-based target detection phase (see, Fig. 5) is divided into four parts: the *vision target selector*, the *detection* and *tracking algorithms* and the *distance vector computing*. The vision target selector manages the switching from detection to tracking when recognizing the target: the detector (see Sec.3.3) is used only at the first step or in case of partial occlusion, otherwise a CAMShift tracking algorithm [31] is employed to follow the car along the path. Then, recognized the target, the distance vector computing block deals with generating the references for the drone trajectory control measuring the distance between the image and bounding box centroids.

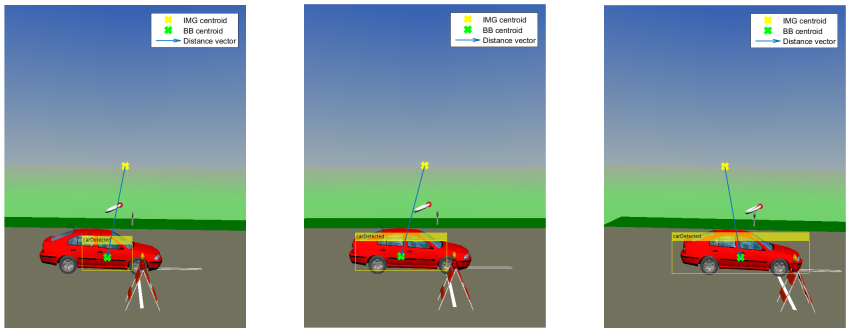
4.1 Vision target selection and tracking

Due to multiple target revelations (as described in Sec. 3.4), a Matlab script was used for obtaining a unique bounding box surrounding the target (the car). The script computes the maximum (Fig. 10(a)) and the average (Fig. 10(c)) bounding boxes, as shown in Fig. 10. The maximum approach put more trust in the detection results, while the average approach tries to filter out the revelation errors. The "good" choice depends on the particular used classifier and on the amount frames employed during the training phase. In our case study, the maximum bounding box has been chosen to figure out the image-based visual problem.

Whereas, the CAMShift algorithm was used to follow the car along the path. This algorithm performs target tracking by searching for its probability distribution pattern in a local adaptive size window whose initial size window is the output of the bounding box selection script. Although it does not guarantee the best performances, the algorithm supplies reliable and robust results [37].

In Figure 11 three consecutive frames produced as output by the tracker are reported. The frames show how the algorithm works exploiting low sensitiveness w.r.t. any change in the object appearance (e.g., shape deformation, scale, and illumination changes or camera motion), compared with detection.

Then, the distance vector between the image (u_{img}, v_{img}) and the bounding centroids (u_{bb}, v_{bb}) is computed, as depicted in Figs. 11 and 12. The vector aims to provide the ref-



(a) (b) (c)

Fig. 11 Three consecutive frames produces as output by the CAMShift tracking algorithm. The image and the bounding box centroids as well as the distance vector among centroids are reported.

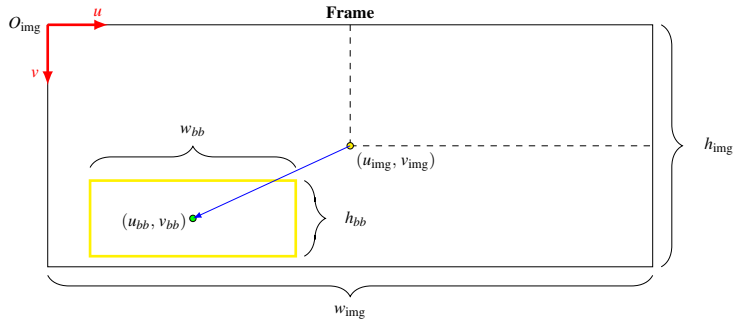


Fig. 12 The scheme illustrates the information extracted by the frames. In blue the distance vector and in yellow the bounding box are reported, respectively. The image $(u_{\text{img}}, v_{\text{img}})$ and bounding box (u_{bb}, v_{bb}) centroids are also represented.

erence signals (e_u and e_v) to the drone trajectory control [21] (see Sec. 6.1), so to move the drone in such a way that the car bounding box center overlaps with the image centroid.

5 Drone Dynamical Model

In our case study, we considered a drone with four rotors in a plus configuration [38]. However, the modular approach used to develop the simulation architecture allows to simulate any multi-rotors aircraft and configuration making the software platform particularly useful for educational purposes.

The design of a high performance attitude and position controller requires often an accurate model of the system. We here recall the commonly used dynamical model of a quadrotor [30] and, by following usual approaches, we introduce two orthonormal frames: the fixed-frame O_{FI} (where FI stands for Fixed Inertial), also called inertial (or reference) frame, and the body-frame O_{ABC} (where ABC stands for Aircraft Body Center) that is fixed in the aircraft center of mass and is oriented according to the aircraft orientation (attitude), see Fig. 13.

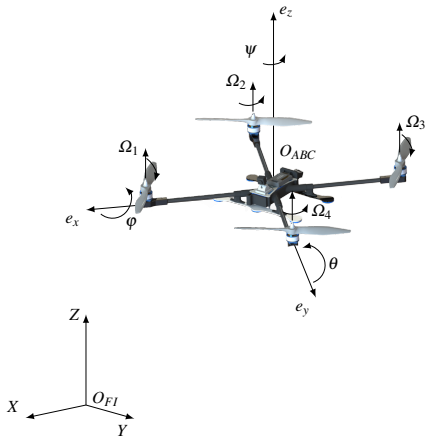


Fig. 13 Drone in the body-frame (O_{ABC}) and the fixed-frame (O_{FI}) reference systems. Forces exerted from each rotor, spin directions and propeller velocities, Ω_i, are also reported.

The translational dynamic equations of the aircraft can be expressed in the inertial frame as follows:

$$m\ddot{\xi} = -mgE_z + u_T R(\varphi,\theta,\psi)E_z, \tag{2}$$

where g denotes the gravity acceleration, m the mass, u_T the total thrust produced by the rotors, $\xi = \begin{pmatrix} x & y & z \end{pmatrix}^\top$ the drone position expressed in the inertial frame, $E_z = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}^\top$ is the unit vector along the Z-axis, while $R(\varphi,\theta,\psi)$ is the rotation matrix from the body to the inertial frame and it depends on the attitude $\eta = \begin{pmatrix} \varphi & \theta & \psi \end{pmatrix}^\top$ (i.e., Euler angles roll, pitch and yaw, respectively) that describes the body-frame orientation according to the ZYX convention [38]. Conversely, the rotational dynamics can be expressed as

$$I\dot{\omega}_B = -\omega_B \times I\omega_B + \tau, \tag{3}$$

where '$\times$' denotes the vector product, $\omega_B = \begin{pmatrix} \omega_x & \omega_y & \omega_z \end{pmatrix}^\top$ is the vector of the angular velocity expressed in the body-frame, $I = \mathrm{diag}(I_x, I_y, I_z)$ is the inertia matrix of the vehicle w.r.t. its principal axis, and $\tau = \begin{pmatrix} u_\varphi & u_\theta & u_\psi \end{pmatrix}^\top$ is the control torque vector obtained by actuating the rotors speeds according to the rotors configuration and the vehicle shape.

At low speeds and around the hovering state the simplified dynamic model consists of six second order differential equations obtained from balancing forces and momenta acting on the drone, where $c_\bullet$ and $s_\bullet$ denote $\cos(\bullet)$ and $\sin(\bullet)$ functions, respectively:

$$I_x\ddot{\varphi} = \dot{\theta}\dot{\psi}(I_y - I_z) + u_\varphi \tag{4a}$$
$$I_y\ddot{\theta} = \dot{\varphi}\dot{\psi}(I_z - I_x) + u_\theta \tag{4b}$$
$$I_z\ddot{\psi} = \dot{\theta}\dot{\varphi}(I_x - I_y) + u_\psi, \tag{4c}$$

$$m\ddot{x} = u_T\left(c_\varphi s_\theta c_\psi + s_\varphi s_\psi\right) \tag{5a}$$
$$m\ddot{y} = u_T\left(c_\varphi s_\theta s_\psi - s_\varphi c_\psi\right) \tag{5b}$$
$$m\ddot{z} = u_T c_\theta c_\varphi - mg. \tag{5c}$$

	Sym.	Value	Unit
Mass	m	0.65	kg
Distance to center of gravity	l	0.23	m
Thrust factor	b_f	$7.5 \cdot 10^{-7}$	kg
Drag factor	b_m	$3.13 \cdot 10^{-5}$	kg m
Inertia component along x-axis	I_x	$7.5 \cdot 10^{-3}$	kg m^2
Inertia component along y-axis	I_y	$7.5 \cdot 10^{-3}$	kg m^2
Inertia component along z-axis	I_z	$1.3 \cdot 10^{-3}$	kg m^2

Table 1 Drone parameters' values.

Equations (4)–(5) represent the nominal model used for designing the control law in [30] and here described in Sect. 6.2. However a more detailed model should be considered when simulation has to be employed as part of the control design process. Thus we introduced further details for catching more realistic behaviors writing the model inputs as

$$u_T = b_f \left(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2\right), \tag{6}$$

and

$$\tau = \begin{pmatrix} u_\varphi \\ u_\theta \\ u_\psi \end{pmatrix} = \frac{b_f}{b_m} \begin{pmatrix} b_m l \left(\Omega_2^2 + \Omega_4^2\right) \\ b_m l \left(\Omega_1^2 + \Omega_3^2\right) \\ -\Omega_1^2 + \Omega_2^2 - \Omega_3^2 + \Omega_4^2 \end{pmatrix}, \tag{7}$$

where Ω_i, $i \in \{1,2,3,4\}$, are the actual rotors angular velocities expressed in rad s^{-1}, l is the distance from the propellers to the center of mass, while b_f and b_m are the thrust and drag factors, respectively. Further details can be found in [30] or [38]. Table 1 reports the drone parameters' values.

6 Flight Control System

With the aim of illustrating a control design methodology exploiting the IBVS approach, we started considering the flight control system described in [21] and [30] that uses a *reference generator* and an *integral backstepping* (IB) *controller* to figure out the drone trajectory tracking problem. The reference generator extracts the information from the images to generate the path to follow, while the integral backstepping (IB) controller starts from such references to compute the needed drone command signals. Figures 14 and 16 describes the whole control scheme.

6.1 Reference Generator

The reference generator is decomposed into two parts: the attitude and the position controller, both illustrated in Fig. 14. The attitude controller tunes the yaw (ψ_r) and the pitch (θ_r) angles trying to overlap the image (u_{img}, v_{img}) and the bounding box (u_{bb}, v_{bb}) centroids (see, Fig. 12) keeping the roll (φ_r) angle equal to zero. Such angles are later used by the position controller for varying the drone reference position x_r, y_r, and z_r considering the drone initial position (x_{init}, y_{init} and z_{init}) and orientation (φ_{init}, θ_{init} and ψ_{init}) and its movements in the virtual environment when tracking the target. In our case study, we assumed the vehicle

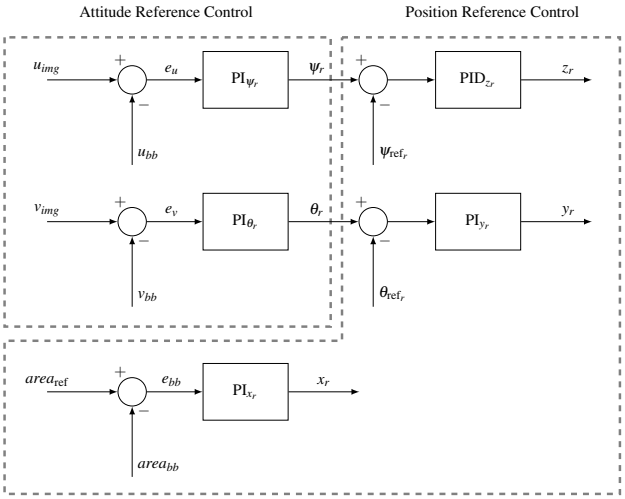


Fig. 14 The reference generator scheme referred to virtual reference system (O_{FVR}). The obtained heuristic PID gains are: $K_{P_{\psi_r}} = 1 \cdot 10^{-5}$, $K_{I_{\psi_r}} = 1 \cdot 10^{-3}$, $K_{P_{\theta_r}} = 1 \cdot 10^{-5}$, $K_{I_{\theta_r}} = 1 \cdot 10^{-3}$, $K_{P_{x_r}} = 1 \cdot 10^{-6}$, $K_{I_{x_r}} = 6 \cdot 10^{-6}$, $K_{P_{y_r}} = 1 \cdot 10^{-2}$, $K_{I_{y_r}} = 1 \cdot 10^{-2}$, $K_{P_{z_r}} = 15$, $K_{I_{z_r}} = 57.5$ and $K_{D_{z_r}} = 3.75$.

starts flying 4 meters over the ground with a distance of 15 meters w.r.t. the car along the x-axis in the O_{FVR} reference system.

Regarding the controller implementation, the considered control architecture is based on control loops that are nothing but PID (proportional-integral-derivative) controllers. These are a standard solution in the literature for the quadorotor controllers design [39]. The PI_{ψ_r} and PI_{θ_r} outputs are the ψ_r and the θ_r angles, respectively, used by the IB controller for computing the command signals (see, Fig. 16) and the y_r reference value (see, Fig. 14).

Whereas, the values ψ_{ref_r} and θ_{ref_r}[4] (see, Fig. 14) represent the references attitude that the vehicle should assume during the target tracking. Finally, the error signal e_{bb}, obtained as the difference between the bounding box ($w_{bb} \cdot h_{bb}$, aka $area_{\mathrm{bb}}$ in Fig. 14) and the reference[5] areas ($area_{\mathrm{ref}}$), is used to tune the distance x_r.

Figure 15 reports the trajectories followed by the car and the drone during the simulation, while a further video has been made available at `https://youtu.be/qAtndBIwdas` for showing the results of the proposed approach. In such video, the quadrotor dynamics has been neglected to highlight how the reference generator works.

6.2 Integral Backstepping controller

The integral backstepping of [30] has been used as the trajectory controller for the path tracking. We decided to employ this controller in particular for educational purposes: it was among the papers that first described how to design a multi-rotors aircraft controller, especially for a quadrotor. Moreover, the chosen integral backstepping controller performs ro-

[4] In our case study, we supposed that the drone should maintain a flat orientation as much as possible during the tracking. Therefore the angles ψ_{ref_r} and θ_{ref_r} have been set equal to zero.

[5] The reference value is given by the sample mean of ROIs collected during the learning process.

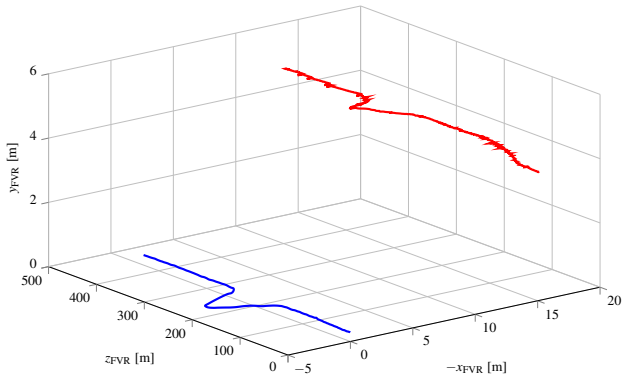


Fig. 15 The car (in blue) and the reference path (in red) described during the simulation when neglecting the drone dynamics.

bustness against external disturbances (offered by backstepping) and sturdiness w.r.t. model uncertainties (given by the integral action).

Starting from the reference generator's outputs, the IB controller computes the orientation ($\varphi_{\text{ref}_{IB}}$ and $\theta_{\text{ref}_{IB}}$) that the drone should assume to follow the reference path (x_r and z_r). The $\varphi_{\text{ref}_{IB}}$ and $\theta_{\text{ref}_{IB}}$ reference angles are computed as:

$$\theta_{\text{ref}_{IB}} = \frac{m}{u_T}\left[\left(1-c_1^2+\lambda_1\right)e_x + (c_1+c_2)\,e_{x_{IB}} - c_1\lambda_1\int_0^t e_x(\tau)\mathrm{d}\tau\right] \tag{8a}$$

$$\varphi_{\text{ref}_{IB}} = -\frac{m}{u_T}\left[\left(1-c_3^2+\lambda_2\right)e_z + (c_3+c_4)\,e_{z_{IB}} - c_3\lambda_2\int_0^t e_z(\tau)\mathrm{d}\tau\right], \tag{8b}$$

with

$$e_{x_{IB}}(t) = \lambda_1\int_0^t e_x(\tau)\mathrm{d}\tau + c_1 e_x(t) + \dot{e}_x(t) \tag{9a}$$

$$e_{z_{IB}}(t) = \lambda_2\int_0^t e_z(\tau)\mathrm{d}\tau + c_3 e_z(t) + \dot{e}_z(t), \tag{9b}$$

and

$$e_x = x_r - x_d \tag{10a}$$

$$e_z = z_r - z_d, \tag{10b}$$

where (c_1, c_2, c_3, c_4, λ_1 and λ_2) are positive constants. For the considered motivating examples, the following values have been chosen: $\lambda_1 = 0.025$, $\lambda_2 = 0.025$, $c_1 = 2$, $c_2 = 0.5$, $c_3 = 2$ and $c_4 = 0.5$.

Figure 16 shows the scheme describing as the control system works.

6.3 Numerical results

The overall system has been simulated in Matlab and the results illustrate in a direct way how the system performs (the video is available at `https://youtu.be/b8mTHRkRDmA`). In

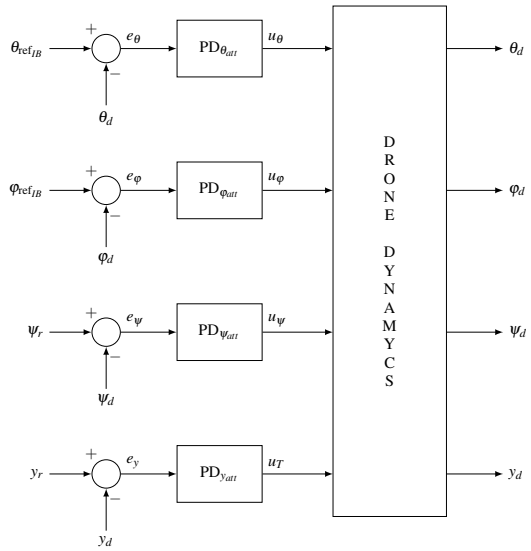


Fig. 16 The drone trajectory controller. All variables are expressed in the virtual reference system (O_{FVR}). Here we recall the heuristic control gains employed into the simulation scenario: $K_{P_{y_{att}}} = 1000$, $K_{D_{y_{att}}} = 200$, $K_{P_{\varphi_{att}}} = 8$, $K_{D_{\varphi_{att}}} = 4$, $K_{P_{\theta_{att}}} = 12$, $K_{D_{\theta_{att}}} = 4$, $K_{P_{\psi_{att}}} = 10$ and $K_{D_{\psi_{att}}} = 4$.

particular it is possible to see, from the drone point of view, how the quadrotor figures out the target tracking for the entire path without ever losing the car: although the vehicle increases its speed during the double-lane change maneuver, the drone is ready enough to capture the dynamic change. Also, after a few seconds of simulation, the quadrotor tilts around the x-axis due to the increasing of the roll angle (drones flies in an eye-in-hand configuration) and the car seems to face a climb.

A further scenario (the video is available at `https://youtu.be/RjXBtPqZZBc`) has been considered to prove the effectiveness and the robustness of the proposed approach as well as the easiness with which the software platform can be customized adding more than one vehicle into the virtual environment. As we can see, the detection and tracking algorithms are able to detect and track the car along the path even if another vehicle with a different color (yellow, for the considered example) is involved in the simulation. In particular, the tracker is able to resize the search window during the experiment avoiding to lose the target until the simulation stops.

Finally, the video at `https://youtu.be/m43Zadq-6XM` shows how the modular approach used in developing the software platform makes easy to change the world scenario in a few steps without the need to redesign the overall architecture. For the considered example, the *vr_octavia_2cars* example has been replaced with *vr_octavia* one keeping everything else unchanged. In the middle of the simulation (at 26 s) the car speed becomes much higher than the drone speed causing an excessive UAV rolling. Due to the eye-in-hand configuration, the target comes out of the camera view and it is lost.

Those results demonstrated as the system works and the limit of the eye-in-hand configuration, as well. Anyhow, the software platform allowed to test the complex system composed by computer vision and control algorithms interacting among them and with the moving objects dynamics.

The experiments as well as the software platform were developed with the 2015b release of Matlab, equipped with Computer Vision System and Virtual Reality toolboxes, but it is compatible with any Matlab successive release. The code is specific for the use case study, but it can be easily and quickly customized to work with any aircraft in the simulation framework.

7 Conclusions

In this work a well-know computing environment (Matlab) has been used to implement the simulation of a complex virtual scenario, in order to show the effects of computer vision and control strategies aimed to detect and track moving objects. In this way, it has been proven the effectiveness and easy to use of the approach for educational purposes, so that interested students might work in a known environment developing their own algorithms in an easy way. Nevertheless, in our opinion the work could constitute the first step towards the development of a more structured platform aimed for the software-in-the-loop of such kind of applications. The idea is to rely on more flexible and dedicated solution like V-REP [12] or Gazebo [40].

Furthermore, this paper illustrated how to expand the functionalities of the proposed platform modeling and integrating any vehicle into the simulation framework. Also, how the potentialities of the overall approach aimed at developing the system in a modular way can be used for facilitating the reuse of software modules.

We published the software as open-source [28] with the aim to share our result with other researchers that might use the platform for testing their algorithms and understanding how different IBVS methodologies and non-linear control techniques can improve the system performances.

References

1. D. Scaramuzza, M. C. Achtelik, L. Doitsidis, F. Friedrich, E. Kosmatopoulos, A. Martinelli, M. W. Achtelik, M. Chli, S. Chatzichristofis, L. Kneip, D. Gurdan, L. Heng, G. H. Lee, S. Lynen, M. Pollefeys, A. Renzaglia, R. Siegwart, J. C. Stumpf, P. Tanskanen, C. Troiani, S. Weiss, and L. Meier, "Vision-Controlled Micro Flying Robots: From System Design to Autonomous Navigation and Mapping in GPS-Denied Environments," *IEEE Robotics Automation Magazine*, vol. 21, no. 3, pp. 26–40, 2014.
2. S. Choi and E. Kim, "Image acquisition system for construction inspection based on small unmanned aerial vehicle," *Lecture Notes in Electrical Engineering*, vol. 352, pp. 273–280, 2015.
3. F. Fraundorfer, L. Heng, D. Honegger, G. H. Lee, L. Meier, P. Tanskanen, and M. Pollefeys, "Vision-based autonomous mapping and exploration using a quadro," in *IEEE International Conference on Intelligent Robots and Systems*, 2012, pp. 4557–4564.
4. D. Anthony, S. Elbaum, A. Lorenz, and C. Detweiler, "On crop height estimation with UAVs," in *IEEE International conference on Intelligent Robots and Systems*, 2014, pp. 4805–4812.
5. M. Blosch, S. Weiss, D. Scaramuzza, and R. Siegwart, "Vision based MAV navigation in unknown and unstructured environments," in *IEEE International conference on robotics and automation*, 2010, pp. 21–28.
6. D. F. de Castro and D. A. dos Santos, "A Software-in-the-Loop Simulation Scheme for Position Formation Flight of Multicopters," *Journal of Aerospace Technology and Management*, vol. 8, no. 4, pp. 431–440, 2016.
7. M. Mancini, G. Costante, P. Valigi, T. A. Ciarfuglia, J. Delmerico, and D. Scaramuzza, "Toward Domain Independence for Learning-Based Monocular Depth Estimation," *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1778–1785, 2017.
8. A. Tallavajhula and A. Kelly, "Construction and validation of a high delity simulator for a planar range sensor," in *IEEE Conference on Robotics and Automation*, 2015, pp. 1050–4729.

9. R. Diankov and J. Kuffner, "Openrave: A planning architecture for autonomous robotics," Robotics Institute, Pittsburgh, PA, Tech. Rep. 79, 2008.
10. A. Elkady and T. Sobh, "Robotics middleware: A comprehensive literature survey and attribute-based bibliography," *Journal of Robotics*, 2012.
11. N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," in *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, vol. 3, 2004, pp. 2149–2154.
12. E. Rohmer, S. P. N. Singh, and M. Freese, "V-REP: a Versatile and Scalable Robot Simulation Framework," in *Proceedings of The International Conference on Intelligent Robots and Systems*, 2013, pp. 1321–1326.
13. S. Shah, D. Dey, C. Lovett, and A. Kapoor, "AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles," in *Field and Service Robotics*, M. Hutter and R. Siegwart, Eds. Springer International Publishing, 2018, pp. 621–635.
14. G. Echeverria, N. Lassabe, A. Degroote, and S. Lemaignan, "Modular open robots simulation engine: MORSE," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 46–51.
15. M. A. Day, M. R. Clement, J. D. Russo, D. Davis, and T. H. Chung., "Multi-uav software systems and simulation architecture," in *International Conference on Unmanned Aircraft Systems*, 2015, pp. 426–435.
16. S. Khan, M. H. Jaffery, A. Hanif, and M. R. Asif, "Teaching Tool for a Control Systems Laboratory Using a Quadrotor as a Plant in MATLAB," *IEEE Transactions on Education*, vol. 60, no. 99, pp. 1–8, 2017.
17. F. Chaumette and S. Hutchinson, "Visual servo control. I. Basic approaches," *IEEE Robotics Automation Magazine*, vol. 13, no. 4, pp. 82–90, 2006.
18. ——, "Visual servo control. II. Advanced approaches," *IEEE Robotics Automation Magazine*, vol. 14, no. 1, pp. 109–118, 2007.
19. J. Aleotti, G. Micconi, S. Caselli, G. Benassi, N. Zambelli, M. Bettelli, and A. Zappettini, "Detection of Nuclear Sources by UAV Teleoperation Using a Visuo-Haptic Augmented Reality Interface," *Sensors*, vol. 17, no. 10, 2017.
20. F. J. Perez-Grau, R. Ragel, F. Caballero, A. Viguria, and A. Ollero, "Semi-autonomous teleoperation of UAVs in search and rescue scenarios," in *2017 International Conference on Unmanned Aircraft Systems*, 2017, pp. 1066–1074.
21. J. Pestana, J. L. Sanchez-Lopez, S. Saripalli, and P. Campoy, "Computer vision based general object following for GPS-denied multirotor unmanned vehicles," in *IEEE American Control Conference*, 2014, pp. 1886–1891.
22. A. Muis and K. Ohnishi, "Eye-to-hand approach on eye-in-hand configuration within real-time visual servoing," in *Advanced Motion Control. The 8th IEEE International Workshop on*, 2004, pp. 647–652.
23. V. Lippiello, B. Siciliano, and L. Villani, "Eye-in-hand/Eye-to-hand Multi-Camera Visual Servoing," in *Proceedings of the 44th IEEE Conference on Decision and Control*, 2005, pp. 5354–5359.
24. B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control*. Springer-Verlag, 2009, ISBN, 978-1846286414.
25. G. Silano and L. Iannelli, "An educational simulation platform for GPS-denied Unmanned Aerial Vehicles aimed to the detection and tracking of moving objects," in *IEEE Conference on Control Applications*, 2016, pp. 1018–1023.
26. G. R. Bradski, "Computer vision face tracking for use in a perceptual user interface," *Intel Technology Journal*, 2nd Quarter 1998.
27. E. Rosten and T. Drummond, *Machine Learning for High-Speed Corner Detection*. Springer Berlin Heidelberg, 2006, pp. 430–443.
28. G. Silano, "MAT-Fly GitHub Repository," 2018. [Online]. Available: https://github.com/gsilano/MAT-Fly
29. MathWorks, "Virtual Reality World and Dynamic System Examples," Mathworks Online documentation resources. [Online]. Available: https://goo.gl/rtEx3S
30. S. Bouabdallah, P. Murrieri, and R. Siegwart, "Towards Autonomous Indoor Micro VTOL," *Autonomous Robots*, vol. 18, no. 2, pp. 171–183, 2005.
31. P. Corke, *Robotics, Vision and Control: Fundamental Algorithms in MATLAB*. Springer, 2011.
32. I. Analyst, "Image Segmentation Tutorial," MathWorks File Exchange. [Online]. Available: https://goo.gl/mn8dhJ
33. A. dos Anjos and H. R. Shahbazkia, "BI-LEVEL IMAGE THRESHOLDING - A Fast Method," in *Proceedings of the First International Conference on Bio-inspired Systems and Signal Processing*, vol. 2. SciTePress, 2008, pp. 70–76.
34. P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1, 2001, pp. 511–518.

35. Y. Xu, G. Yu, X. Wu, Y. Wang, and Y. Ma, "An enhanced viola-jones vehicle detection method from unmanned aerial vehicles imagery," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 7, pp. 1845–1856, 2017.
36. P. Martinez-Gonzalez, S. Oprea, A. Garcia-Garcia, A. Jover-Alvarez, S. Orts-Escolano, and J. Garcia-Rodriguez, "Unrealrox: an extremely photorealistic virtual reality environment for robotics simulations and synthetic data generation," *Virtual Reality*, 2019.
37. N. M. Artner and W. Burger, "A Comparison of Mean Shift Tracking Methods," in *2017 International Conference on Unmanned Aircraft Systems*, 2008, pp. 197–204.
38. B. L. Stevens, F. L. Lewis, and E. N. Johnson, *Aircraft control and simulation: dynamics, controls design, and autonomous systems*. John Wiley & Sons, 2015.
39. T. N. Dief and S. Yoshida, "Review: Modeling and Classical Controller Of Quad-rotor," *International Journal of Computer Science and Information Technology & Security*, vol. 5, no. 4, pp. 314–319, 2015.
40. C. E. Agüero, N. Koenig, I. Chen, H. Boyer, S. Peters, J. Hsu, B. Gerkey, S. Paepcke, J. L. Rivero, J. Manzo, E. Krotkov, and G. Pratt, "Inside the Virtual Robotics Challenge: Simulating Real-Time Robotic Disaster Response," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 494–506, 2015.