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

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 aircrafts characterized by 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 numerical simulator makes easy and quick to insert and to remove flight control system components, testing and comparing different plans, both for indoor and outdoor scenarios, when computer vision algorithms are in the loop. In an automatic way, the simulator is able to acquire frames from the virtual world, 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 in that way the image-based visual servoing problems.

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

Keywords Image-based visual servoing · 3D virtual reality environment · MATLAB Simulink · Multi-rotors · Educational · Vision detection and tracking · UAV · software-in-the-loop · Computer Vision System Toolbox · Virtual Reality Toolbox

Giuseppe Silano
Department of Engineering,
University of Sannio in Benevento,
Piazza Roma, 21; 82100 Benevento (Italy)
E-mail: {giuseppe.silano}@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 increasing interest and attention related to civilian use. Nowadays, UAVs are employed for several tasks and services like surveying and mapping [1], for rescue operations in disasters [2,3], for spatial information acquisition, buildings inspection [4,5], data collection from inaccessible areas, geophysics exploration [6,7], traffic monitoring [8], animal protection [9], agricultural crops and monitoring [10], manipulation and transportation or navigation purposes [11,12].

Many existing algorithms for the autonomous control [13] and navigation [14,15] 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 [16,17,18], 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. To that aim, 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 [19].

Different solutions, typically based on external robotic simulator such as Gazebo [20], V-REP [21], Webots [22], AirSim [23], are available to this purpose. They employ recent advances in computation and graphics (see, e.g., the AirSim photorealistic environment [14]) 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. 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 [24,13]) in the UAV framework. Furthermore, as highlighted in [25], 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. Finally, the open-source code [26], through which it was made available, makes the platform potentially endless: students, researchers and developers can expand the software platform functionalities modeling and integrating any vehicle in the simulation framework combining also machine learning techniques (e.g., reinforcement learning, deep learning, support vector machine, etc.) both for control and vision problems.

In our paper the specific domain of interest regards the behavior of UAVs acting in accordance with the Image-Based Visual Servoing (IBVS) [27]. Most are the commercial applications that use UAVs, 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 on-board sensors, 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, with low power consumption, makes cameras one of the most suitable sensors, especially in GPS-denied environments [28].

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 [29]. 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* [30]. Many commercial Micro Aerial Vehicles (MAVs), such as the Parrot AR.Drone 2.0 or more recent Parrot Bebop 2.0, adopt the eye-in-hand configuration also for advanced research tasks and to develop complex applications [28]. In this paper we assumed the same configuration for the drone taking into account the ideal pinhole camera model[1] [31] employed to simulate the scenario.

The application we consider, that is an extension of our previous work [32], has the objective of making the UAV able to detect and track a specific object (a car) moving on the ground. 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 partial occlusion. Otherwise, a Continuously Adaptive Mean-Shift (CAMShift) tracking algorithm [33] is employed to follow the car along the path. 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 car for each image, thus avoiding to pass through the specific Matlab tool *Training Image Labeler*, and also compare the performance of various classifiers selecting in an automatic way the best one among different features types (Haar, HOG, LBP) [34] and different number of training stages. Simple testbeds have been considered to analyze the effectiveness and the robustness of the platform.

The paper is organized as follows. Section 2 explains the virtual reality world 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 the control algorithms are reported in Sec. 6. Finally, Section 7 concludes the paper.

[1] The pinhole camera model describes the mathematical relationships between the coordinates of a point in a three-dimensional space and its projection onto the image plane of an ideal pinhole camera, where the camera aperture is described as a point and no lenses are used to focus light. The model does not include geometric distortions or blurring of unfocused objects caused by lenses and finite sized apertures. It also does not take into account that most practical cameras have only discrete image coordinates.

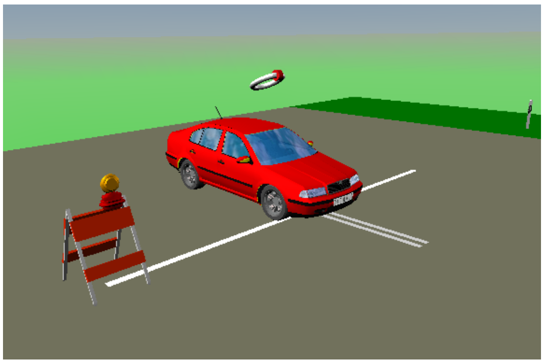


Fig. 1 Initial frame of the considered scenario in the paper. The visualization of forces and other similar quantities have been disabled in order to improve the classifier performance.

2 System Description

In order to simulate a scenario as much similar as to the real world, the Matlab Virtual Reality (VR) Toolbox (aka Simulink 3D Animation Toolbox) has been used. The toolbox allows the visualization, in a 3D world, of complex systems dynamic behavior while they interact with the surrounding scenario. It also provides ready-to-use environments [35], directly employable to simulate the reality. Such scenarios can be customized to create new ones by using any common text editor or 3D authoring tool. Moreover, the toolbox allows the interaction with the environment via force feedback steering wheels, mouse or other input hardware devices. The interaction with the virtual world is done by manipulating object properties, such as scale, position and rotation at simulation time. Whereas, thanks to offline animation recording functionality, frames and videos of the 3D environment can be acquired and used to implement the visual-based object tracking problem. Through the VR features, it is possible to navigate in the virtual world by zooming, panning moving, sideways, and rotating about a point of interest known as viewpoint. Such point can be used to observe any object in motion from different positions and orientations.

From such perspective, the tool has been meant for simulating the interaction of a drone following a car. We started from one of the examples available on the MathWorks platform (specifically the *vr_octavia_2cars* example) for describing a quite detailed dynamical model of a car moving along a non trivial path (see, Fig. 1). The employed scenario represents a standard double-lane-change maneuver conducted in two-vehicle configurations. The first one engages the Electronic Stability Program (ESP) control unit, the second one switches such control unit off, respectively. In the example two data sets containing the vehicle dynamics are sent in parallel to the virtual reality scene in order to drive the cars along the path. From that, first a simpler scenario was considered: we removed a vehicle configuration (the car without the ESP) by exploiting the model through the VRML97 EXTERNPROTO format[2] [36]. In such a way, the attention has been put in the UAV dynamics and control, exploiting the object oriented approach and minimizing the development time. However, the considered scenario is only one of the ready-to-use examples described on the MathWorks website (a full list is available, [35]), others can be customized by using the same criteria.

[2] It allows to create (or to delete) several identical vehicles as instances of a common 3D object.

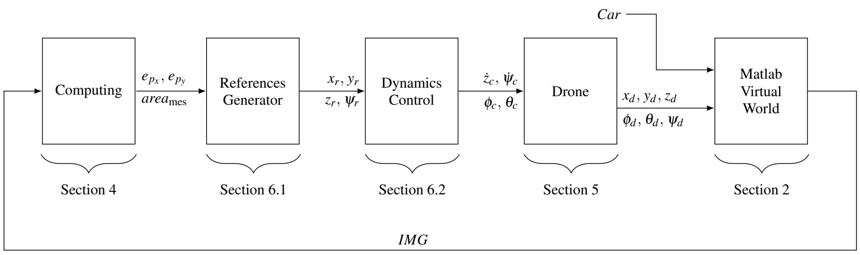


Fig. 2 The control scheme. Subscript c indicates the commands, r indicates references and d indicates the drone. Each block is mapped to the section in which it is described.

Thus, an external observer has been added to the scheme. In Matlab VR the external observer has six degrees of freedom (DOFs): the spatial coordinates x, y, z, and the angles *yaw* (ψ), *pitch* (θ) and *roll* (ϕ)[3], see Fig. 15. In this way it is possible to simulate the behavior of a drone that moves in the virtual environment and observes the car moving along the path. The whole process is the following: images are updated according to the position and the orientation of the Matlab VR external observer (the UAV) with respect to the car; such images are acquired and elaborated for getting necessary information in order to detect and track the object (the car), and to run the control strategy designed for tracking the moving object. The output of the control algorithm consists into the actuation commands (ψ_c, θ_c, ϕ_c and $\dot{z}_c$) that should be given to the drone in order to update its position (x_d, y_d and z_d) and orientation (θ_d, ψ_d and ϕ_d), see Fig. 2. Here a further low level feedback control loop is present, as explained in Sec. 6.

The scheme depicted in Fig. 3 shows the overall platform architecture. Colors are employed to highlight the different parts of the system that make it up: the classifier learning phase (in blue), the vision based target detection (in green), the flight control system (in yellow) and the Matlab VR (in red). Conversely, the dashed line allows to separate the learning phase from the simulation scenario. Each part of platform will be properly described in the next sections.

The entire architecture was designed by using a modular approach. The system components are split into smaller parts, called modules (e.g., the detection algorithm, the reference generator, the classifier synthesis, etc.), that can be independently created and then used into the simulator making it low in coupling and high in cohesion. In this way, every block can be easily replaced with another being careful to maintain the same interface (inputs and outputs). Such characteristics allow to combine and to test different computer vision and control algorithms, evaluating performances and working, for educational purposes.

In Figure 4 the Simulink scheme employed to simulate the virtual scenario is reported. The car dynamics are described in the block *esp_on*. At each time step, they allow to simulate the steering angle, the brake system intensity, the linear velocity and position of the car, and all forces needed to follow a given path. The data processed by the *Coordinate Transformations* and *VR Visualization* blocks take care of the car movements in the virtual environment. Conversely, the *observer_position* and the *rotation_matrix* blocks represent the aircraft position (x_d, y_d and z_d) and the direction cosine matrix [37], respectively. Such matrix defines the drone attitude (ψ_d, ϕ_d and θ_d) in the virtual reality world by using the Rodrigues's formula

[3] Keeping in mind that in the VRML format the viewpoint orientation is defined in the form of 4-element (axis angle): the VRML rotation. So, the camera direction must be converted into this format before sending it to the virtual scene.

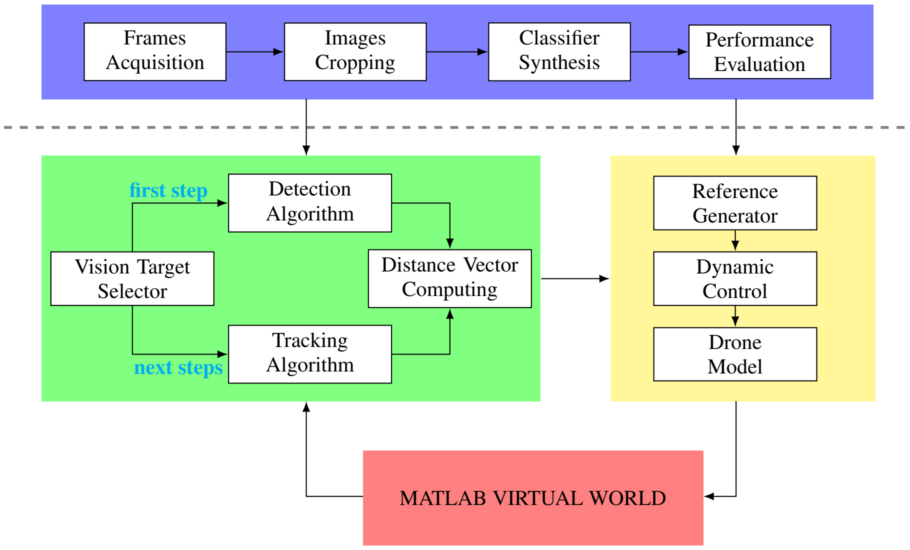


Fig. 3 Proposed software platform architecture. The blue block on top represents the Classifier Learning Phase (Sec. 3); the green block includes the Vision Based Target Detection (Sec. 4); the yellow block covers the Flight Control System (Sec. 6). Finally, in red the Matlab Virtual World block (Fig. 4). The dashed line temporally separates the learning phase from the simulation scenario.

[38]. Both blocks are updated by the regulator at each frame (time step) during the entire simulation. Finally, the vehicle properties such as color (*car_colour*) and scale (*car_scale*) are set.

$$\begin{aligned}\mathbf{u}^b &= R_x(\phi)\cdot R_y(\theta)\cdot R_z(\psi)\cdot \mathbf{u}^r \\ &= \begin{bmatrix} c_\theta c_\psi & c_\theta s_\psi & -s_\theta \\ -c_\phi s_\psi + s_\phi s_\theta c_\psi & c_\theta c_\psi + s_\psi s_\theta s_\phi & s_\phi c_\theta \\ s_\theta s_\psi + c_\phi s_\theta c_\psi & -s_\phi c_\psi + c_\phi s_\theta s_\psi & c_\phi c_\theta \end{bmatrix} \end{aligned} \tag{1}$$

where

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\phi & s_\phi \\ 0 & -s_\phi & c_\phi \end{bmatrix},\ R_y(\theta) = \begin{bmatrix} c_\theta & 0 & -s_\theta \\ 0 & 1 & 0 \\ s_\theta & 0 & c_\theta \end{bmatrix},\ R_z(\psi) = \begin{bmatrix} c_\psi & s_\psi & 0 \\ -s_\psi & c_\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Equation (1) describes the direction cosine matrix and its components, where $c_\bullet$ and $s_\bullet$ denote $\cos(\cdot)$ and $\sin(\cdot)$ functions, respectively. The superscripts b and r indicate the *body reference* (O_{ABC}) and the *inertial reference* (O_{FI}), respectively (see, Sec. 5). Note that Matlab VR adopts a reference system slightly different form the classic fixed reference frame O_{FI}. Figure 5 illustrates such difference: the reference system on the left is the classic fixed reference frame O_{FI}, while the on the right the reference system adopted in the virtual world (O_{FVR}). In particular, axes are differently oriented and, furthermore, the virtual world 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.

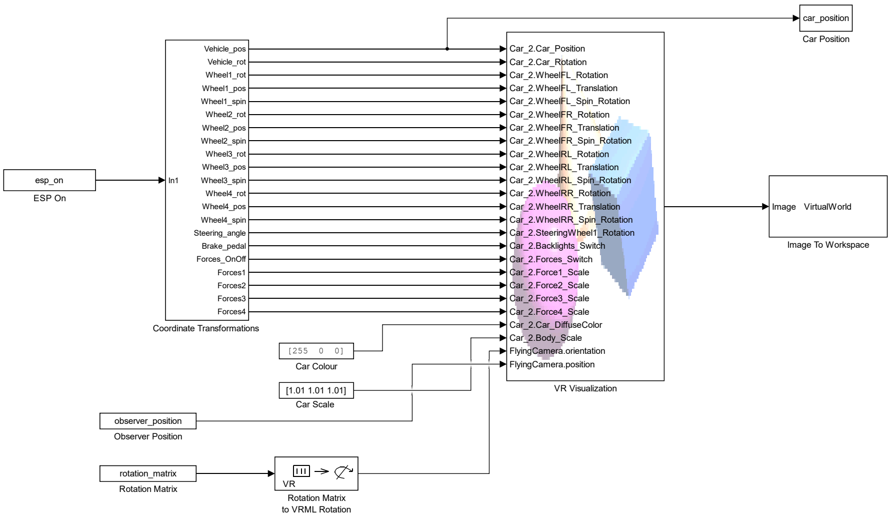


Fig. 4 Virtual world Simulink scheme.

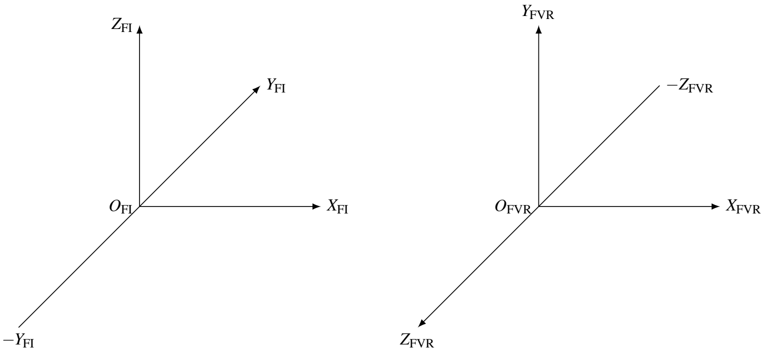


Fig. 5 The picture illustrates the classic fixed frame O_{OFI} (left) and the corresponding virtual fixed O_{FVR} (right) reference system.

Finally, the Simulink scheme saves the current car position (x_{car}, y_{car} and z_{car}), then 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, employed for the pattern recognition.

Starting the simulation is quite simple, so as customizing it: is enough to run the *main* script into the *Object_detection_and_tracking_simulator* repository. The software platform starts working in synchronous mode enabled: the script is automatically triggered each simulation step (0.05 seconds). Both control and image processing times are neglected in the considered use cases. Such characteristic is not a limitations: any time they can be hypothesized and added in the loop studying how they influence the simulation behavior.

3 Classifier Learning Phase

The classifier learning phase is the most important part of the system, from it depends the object detection and tracking as well as the flight control system. Matlab scripts have been developed to automate the entire procedure: from the frames acquisition to the classifier synthesis and performance evaluation. To that aim, the learning process has been split in four parts, as is depicted in Fig. 3: 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 [39], in particular we used 2626 positive images and 5252 negative images achieving a $1:2$ ratio.

To this aim, 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,

$$\begin{cases} y = r\cos\beta \\ x = r\sin\beta\sin\alpha \\ z = r\sin\beta\cos\alpha \end{cases}, \tag{2}$$

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 as well as the acquired data have been made available [40,41].

The TIF (Tagged Image File) format was used for the frames. It is a flexible, adaptable file format for handling images and data within a single file. It also includes the header tags (e.g., size, definition, image-data arrangement, applied image compression) and defines the image geometry allowing to edit and re-save images without losing quality.

3.2 Image cropping

Starting from the approach proposed in [42], a Matlab script was developed to automatically select the bounding box area of the car. The image segmentation has been employed to simplify and to change the representation: from RGB[4] to binary (Figs. 8(a) and 8(c), respectively), passing through the gray scale (Fig. 8(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 with respect to some characteristics or computed property, e.g., color (the red of car, in our case), intensity, or texture. Adjacent regions are significantly different with respect to the same properties by allowing to detect the target.

The Balanced Histogram Thresholding (BTH) method [43] was employed to divide the background from the foreground images, in whose is contained the target. The BHT splits

[4] The RGB is an additive color model in which red, green and blue light are added together in various ways to reproduce a broad array of colors. The name of the model comes from the initials of the three additive primary colors: red, green, and blue.

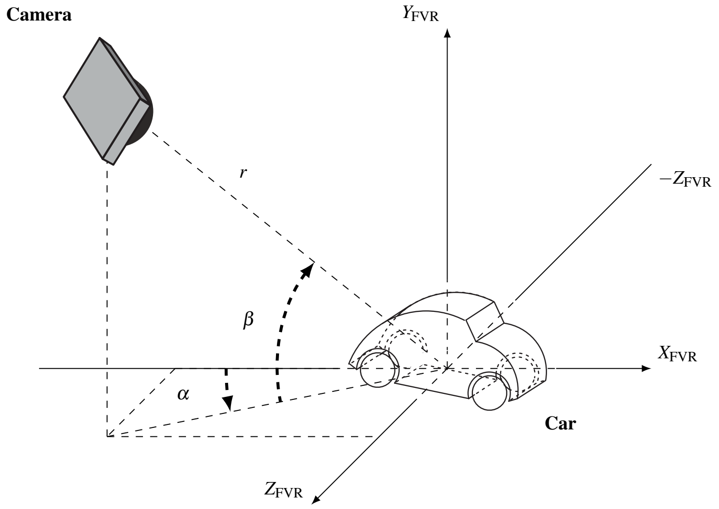


Fig. 6 Drone trajectory around the car parked in its initial state during the frames acquisition.

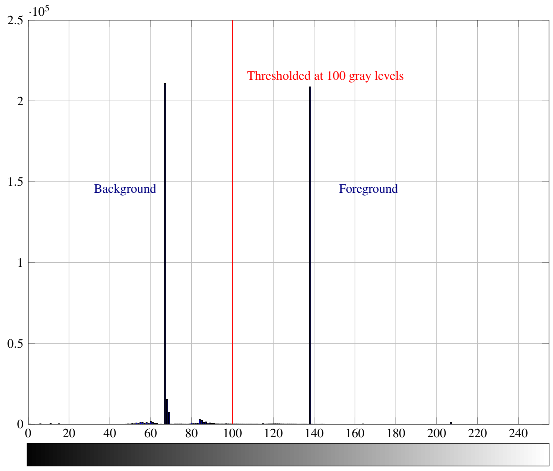
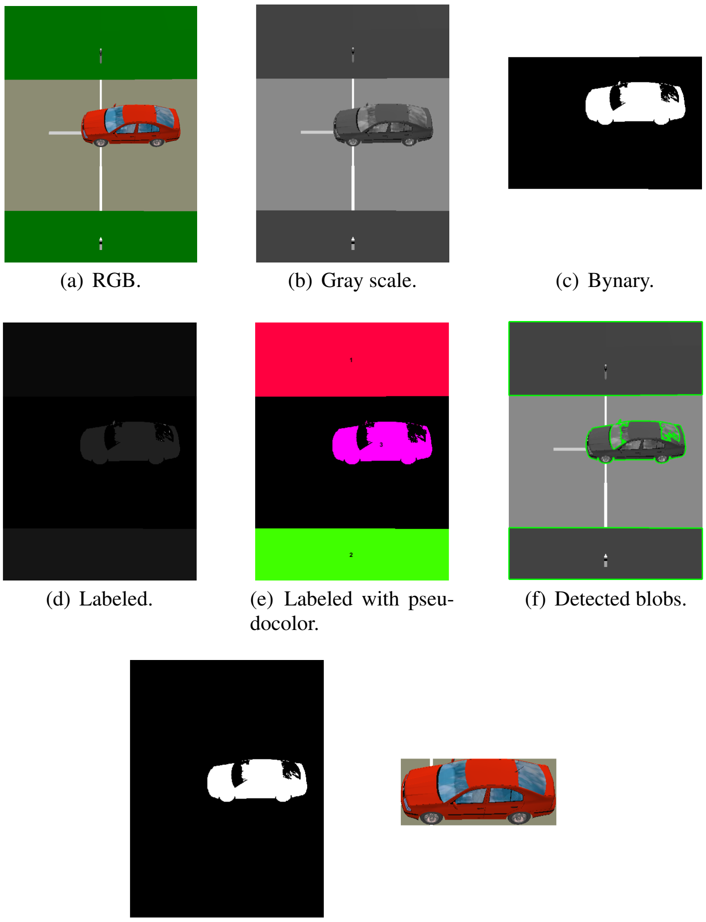


Fig. 7 Histogram of image data. The red line, the grayscale threshold, splits the graph in two parts: background and foreground (in whom is contained the target). The gray gradient bars indicate the associated color to each x-value, from 0 to 255.

the image data in two main classes, the backgrounds and the foregrounds (see, Fig. 7), searching for the optimum threshold level. In particular, the method weighs the histogram, checking which of the two sides is heavier, and removes weight from the heavier side until it becomes the lighter. It repeats the same operation until the edges of the weighing scale meet.

Starting from the foreground binary image (see, Fig. 8(c)), the script identifies individual blobs (Fig. 8(d)) by looking for pixels are connected to each other. Then, by using the connected-component labeling algorithm [44], a graph is constructed from the inputs data



(a) RGB.

(b) Gray scale.

(c) Bynary.

(d) Labeled.

(e) Labeled with pseudocolor.

(f) Detected blobs.

(g) Selected blob.

(h) Cropped image.

Fig. 8 Frames derived from the images computing process. From the RGB file format (a) to the cropped image (h) the steps are reported, sequentially. Each figure is labeled in according to the suitable phase in whom is obtained.

containing vertexes and connecting edges. The algorithm traverses the graph by labeling vertexes based on connectivity and relative values of their neighbors [45] in according to a fixed heuristic (8-connected, in our case). In Figure 8(e) different colors have been assigned to each blob in order to clearly show the distinct. In addition, numbers (e.g, from one to the number of blobs) were used to identify blobs and distinguish them from one another.

Finally, extracted all images properties, the script selects the blobs (bounded and overlapped in Fig. 8(f)) that meet criteria in terms of size and intensity. In Figures 8(g) and 8(h) the chosen blob and the derived cropped image (obtained from the initial frame by using the extracted ROI value) are shown, respectively.

Thus, 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 8, for a single sample frame, all image processing steps, from the acquired frame (see, Sec. 3.1) to the cropped image, are depicted. Such steps allow to better understand how the algorithm works. For the considered image set, the script was able to automatically detect the ROI with an error of 8.18%: on 2626 positive images, the target is not recognized in only 215 frames and the first ROI loss appears at 1791th sample.

Finally, a MAT-file containing for each positive image the suitable ROI components $((x_{bb} - w_{bb})/2$, $(y_{bb} - h_{bb})/2$, w_{bb} and h_{bb}, see Fig. 14) was made in output. Such file was employed during the classifier synthesis (see, Fig. 3) to design the target (the car) detector.

3.3 Classifier synthesis

Collected the ROIs, for the overall positive images set, the classifier has been designed by using different features types (Haar, HOG, LBP) [34] and training stages, evaluating its performance for each configuration. The Viola & Jones algorithm [46], in particular its practical implementation by employing the Matlab cascade object detection function *trainCascadeObjectDetector*, has been chosen to that aim. Although the algorithm was originally designed and developed for face detection, it is possible to extend its use to any object taking care to employ a lot of training data to reach the convergence. Indeed, thanks to its fastness and robustness, many and various are the implementations [47,48] in which the algorithm finds application (e.g., the open-source computer vision library OpenCV, [49]).

The MathWorks Computer Vision System (CVS) Toolbox™ was used during the classifier synthesis. The toolbox supports several approaches to recognize a target in any image or video, including the Viola & Jones algorithm. Moreover, it employs Haar-like features and a cascade of ready-to-use and easy to customize pretrained classifiers [50].

Starting from the ROI components, we employed the attentional cascade training scheme [51] by using different numbers of cascade layers to obtain an accurate detector of the target. As depicted in Fig. 9, the scheme creates a model of the target from its positive and negative images set expressed by an eXtensible Markup Language (XML) file. Such model was used as input for the function *trainCascadeObjectDetector*, then to synthesized the classifier tuning the function parameters as follows:

- *ObjectTrainingSize*: the objects size employed during the training activity. They are in the form of a vector with two elements: rows and columns (800 × 600, in our case). All negative and positive images were resized to the specified values before starts the training.
- *NegativeSamplesFactor*: the negative samples gain factor directly related to the detection performance. In our case study, it has been fixed equal to 2 by achieving a 1: 2 ratio.
- *FalseAlarmRate*: the false positive result rate (*FalsePositiveRate*) accepted for each layer of the attentional cascade. A false positive result represents a negative sample classified as the target. The parameter value can be a number in the interval $(0,1]$. It has been chosen equal to 0.001 in the testbed examples.
- *TruePositiveRate*: the positive result rate correctly detected for each layer. The parameter value can be a number in the interval $(0,1]$, and it was set equal to 0.995 (the default value).

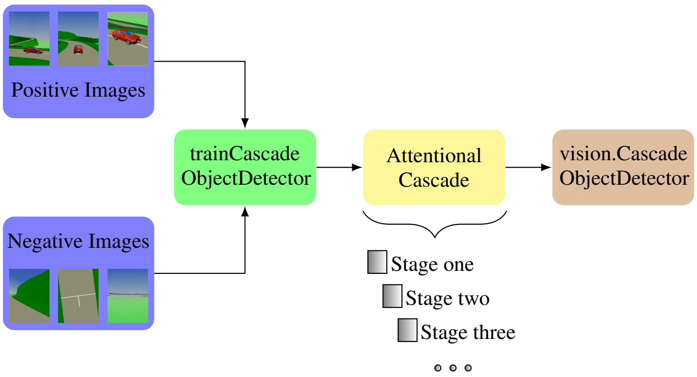


Fig. 9 The attentional cascade training scheme. The positive images include the car, while the negative images depict only the virtual scenario.

- *FeatureType*: the feature type used for training. The accepted types are “Haar” [46], “LBP” [52] and “HOG” [53]. For the proposed motivating examples, “Haar” was used as feature type.
- *NumCascadeStages*: number of cascade stages employed to train the classifier. They are related to the training time in a direct way. In our case, we chose 4 stages for the Viola & Jones detector.

Finally, the Viola & Jones classifier is build through the *vision.CascadeObjectDetector* function. However, the proposed detector is one of possible solutions described in the literature [54,55], so other detectors could be employed to this aim.

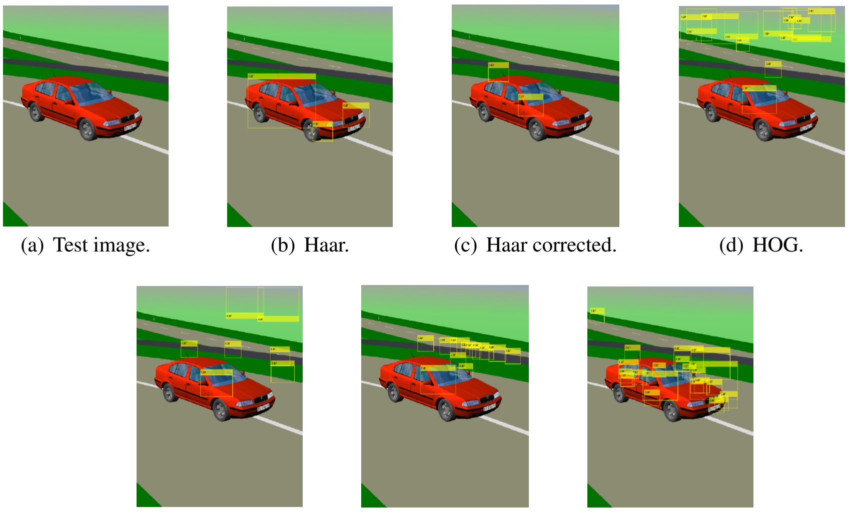
The function parameters must be always specified for achieving an optimal functioning detector, from it depends the entire process: a bad detection result causes without doubt the inability to follow the car.

Although Haar and LBP features are often used to detect faces, because they work well for representing fine-scale textures, while the HOG features are often used to detect objects (e.g., people and cars), due to the non photorealism of the Matlab VR frames, the Haar features are resulted more useful for capturing the overall shape of the target (see, Fig. 10). Unfortunately, training a detector by using Haar features takes much longer.

3.4 Performance comparison

In order to choose the best classifier for the proposed testbeds, a Matlab script for performance analysis was developed. It is part of the proposed software platform (see, Fig. 3), and allows in easy way to compare different classifiers by showing the detection results.

Two different models have been considered to synthesize the detectors. The first one obtained by using the ROIs automatically extracted from the virtual scenario, as described in Sec. 3.2, the second one obtained by “correcting” that ROIs through the Matlab tool *Training Image Labeler*. Such models show the effectiveness of the proposed approach by analyzing how the revelation error influences the performance. The ROIs have been corrected in 215 positive images.



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

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

Fig. 10 Detection results obtained by using the Haar cascade, histogram of oriented gradients (HOG) and local binary patterns (LBP) features types. Two different target models are considered: the "corrected" and "uncorrected" version.

As discussed in Sec. 3.3, the classifiers were built by tuning the feature types and the training stages and then tested on a test images set[5] in order to evaluate their performance. In Figure 10 the detection results obtained for a single sample image are depicted. Moreover, the script is able to compare classifiers made by using the same feature type (Haar, HOG, LBP) but different function parameters (e.g., *NumCascadeStages*, *TruePositiveRate*, etc.). Such behavior increases the platform potentiality and its use for educational purposes.

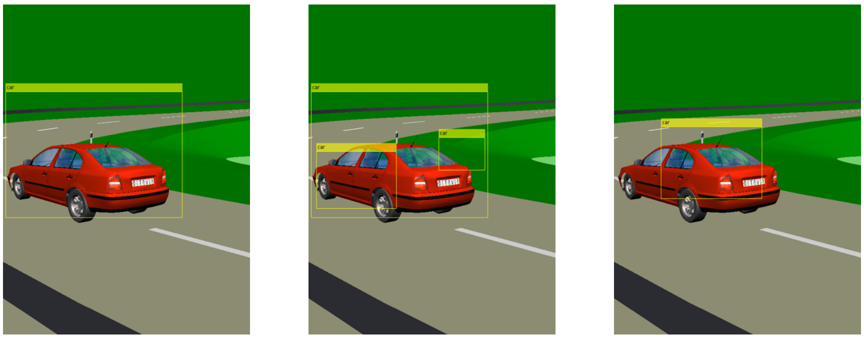
In all detections, the car is only partially detected in spite of high images number used in the learning process. Except in same cases, there are no revelation errors: different bounding boxes are detected in the image. That is probably caused by the several image view points used in the learning process and by the absence of photorealism in the images. 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.

In Figs. 10(b) and 10(c), the detection results shown appear similarly. 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 of the detector avoiding to pass through the Matlab tool *TrainingImageLaebeler*, time related in a direct way to the images number.

4 Vision Based Target Detection

The Viola & Jones algorithm [46] has been used for recognizing the car along the path. Such algorithm, despite requiring more computational time than other [56] (e.g., SIFT and SURF

[5] The images have been chosen as varied as possible: different backgrounds, illumination conditions, view points, etc., so as described in [45].



(a) Maximum bounding box. (b) All bounding boxes. (c) Average bounding box.

Fig. 11 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, whereas in the middle all revelations are reported.

algorithms [54,55]), it ensured the best performance for the considered case study. Much time has been dedicated to the learning process (see, Sec. 3) in order to detect the target when it was in different positions. For all such reasons, the resulting classifier model was used in the vision based target detection.

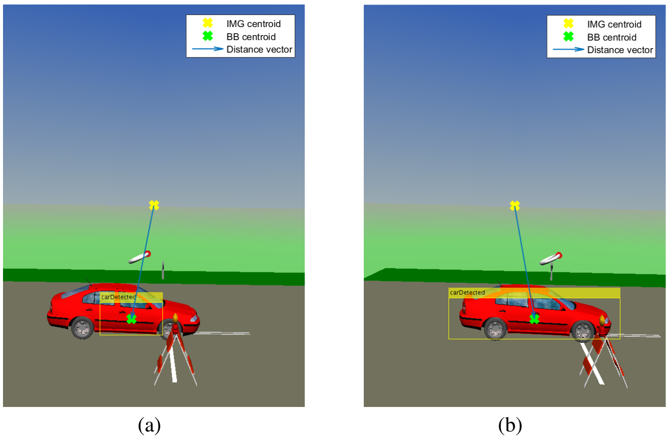
The target detection is divided in three parts: the bounding box selection script, the tracking algorithm and the distance vector computing. The classifier and the tracking algorithm are mutually exclusive: the detector is used only at the first step or in case of partial occlusion, otherwise a CAMShift tracking algorithm is employed to follow the car along the path. Detected the target, the bounding box selection script deals with reducing the revelations number into the frame. Conversely, when the tracking algorithm turns on, the box obtained by the detector is employed as initial search windows (see, Sec. 4.2). In both cases, the distance vector between the image and bounding box centroids is computed (see, Sec. 6) and used to generate references for the drone trajectory control (see, Sec. 4.3).

4.1 Bounding box selection

To avoid multiple revelations as described in Sec. 3.4, a bounding box selection script was build. The recognition of different bounding boxes requires an algorithm in order to obtain a unique box that surrounds the target (the car). The Matlab script computes the maximum and the average bounding box, as shown in Fig. 11. The maximum approach put more trust in the detection results while the average approach tries to filter out the revelations. By considering the time spent during the classifier design and the obtained results, the maximum bounding box was chosen as trade-off solution to obtain a unique bounding box surrounding the target.

4.2 Tracking algorithm

The CAMShift algorithm performs the target tracking by searching for its probability distribution pattern in a local adaptive size window within the frame. To this aim, the maximum



(a) (b)

Fig. 12 CAMShift tracking of a red colored target (the car in the virtual scenario) on a images sequence. The yellow box corresponds to the boundaries of the tracked area. Its dimension increase from (a) to (b) with the flow of time. The image and the bounding box centroids as well as the distance vector are also depicted.

bounding box (the output of the bounding box selection script) has been used as heuristic for the initial size window.

By exploiting the tracker less sensitive respect to any change in the object appearance, slow and gradual in the reality, better performance than detection are guaranteed. In Figure 12 an example of a color tracking via the CAMShift is shown. Moreover, the figure depicts how the algorithm works by adapting the size window in order to perform the tracking. In such a way, it allows to avoid the target appearance variations problem (e.g., shape deformation, scale and illumination changes or camera motion), that instead characterize the detection.

At each time step, the image is converted from the RGB to the HSV (Hue Saturation Value) color space (see, Fig. 13). The HSV [57] separates the color information (hue) from saturation (how concentrated the color is) and from brightness. The hue corresponds to the intuitive notion of color. It is invariant respect to light conditions, so it can be exploited for color segmentation under non-uniform illumination, thanks to its brightness independence. Conversely, the saturation is used to describe how pure a color is or how much white is added to a pure color (its intensity), while the value (V) refers to the lightness or darkness of it.

From such information, the *vision.HistogramBasedTracker* function, included in the CSV toolbox, was used to track the car along the path. It employs the histogram of pixel values to identify the tracked object (the car), a graphical representation of the tonal distribution in the digital image (the number of pixels for each tonal value), and the initial search window obtained from the bounding box selection phase (see, Sec. 4.1). Once the initial region of interest is selected in the current frame, the window is moved in the next image according to the distance vector between the center of the search window and the calculated centroid (see, Fig. 12). As described in the bounding box selection phase (see, Sec. 4.1), also the tracker function outputs a bounding box surrounding the target. However, the CAMShift tracking algorithm it not the only tracker employable to that aim, a full list is available on the MathWorks website [58].

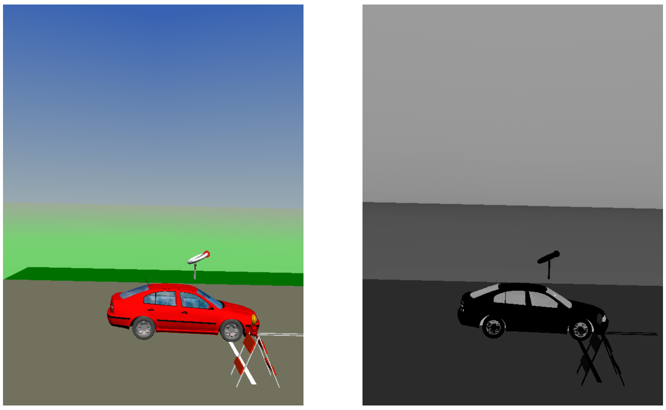


Fig. 13 A sample image extracted from the scenario. The RGB (left) and HSV (right) image representation are reported.

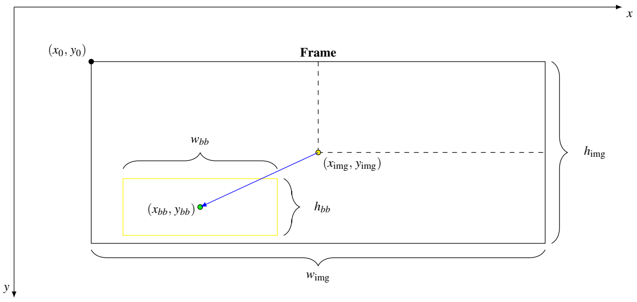


Fig. 14 The scheme illustrates the information extracted by the frames. In blue the distance vector and in yellow the bounding box are reported. The image and bounding box centroids are also represented.

4.3 Distance Vector

As explained in Secs. 4.1 and 4.2, recognized the target into the scenario, the scripts provide in output the bounding box surrounding the target. In particular, the horizontal x_{bb} and vertical y_{bb} location of the upper-left bounding box corner (x_0, y_0), and its width w_{bb} and height h_{bb}, are used to indicate the estimated car position into the frame. All variables are expressed in pixel as well as the size constants of the image, i.e., width $w_{\text{img}} = 800px$ and height $h_{\text{img}} = 600px$.

Starting from that, the image $(x_{\text{img}}, y_{\text{img}})$ and the bounding box (x_{bb}, y_{bb}) centroids are computed. Such coordinates are employed to compute the distance vector among centroids, used as reference for the drone trajectory control, as described in [28] and illustrated in Fig. 14. The vector aims to provide how much the drone should move forward, backward, upstairs or downstairs, to follow the car along the path maintaining the distance from it (see, Sec. 6).

5 Model of a quadrotor drone

In our case study, we considered a drone with four rotors in plus configuration[6]. However, the software platform allows to simulate any drone and configuration. Indeed, thanks to the modular approach on which it has been developed, in easy way the drone parameter as well as its model and control can be replaced evaluating the behavior changes.

The development of a suitable attitude and position controller for the quad-copter required an accurate dynamical model to be derived. Classical Newtonian and Lagrange modeling methods [37] can be applied. In our case the Newtonian approach, the extensively employed choice for modeling traditional helicopters [59], has been used. We introduced two reference systems: the fixed-frame O_{FI} (where FI stand for fixed inertial), also called inertial frame, and the body-frame O_{ABC} (where ABC stands for Aircraft Body Center) that is fixed in the aircraft center of gravity and oriented according to the aircraft attitude, see Fig. 15.

Such model consists of six equations for the system dynamics (eqs. (3)) and four equations describing the inputs to the system (eqs. (5)), where $c_\bullet$ and $s_\bullet$ denote $\cos(\cdot)$ and $\sin(\cdot)$ functions, respectively.

$$
\begin{cases}
\ddot{\phi}_d = \dot{\theta}_d \dot{\psi}_d \left(\frac{I_{yy}-I_{zz}}{I_{xx}}\right) - \frac{J_r}{I_{xx}} \dot{\theta}_d \Omega + \frac{l}{I_{xx}} U_2 \\
\ddot{\theta}_d = \dot{\phi}_d \dot{\psi}_d \left(\frac{I_{zz}-I_{xx}}{I_{yy}}\right) + \frac{J_r}{I_{yy}} \dot{\phi}_d \Omega + \frac{l}{I_{yy}} U_3 \\
\ddot{\psi}_d = \dot{\phi}_d \dot{\theta}_d \left(\frac{I_{xx}-I_{yy}}{I_{zz}}\right) + \frac{1}{I_{zz}} U_4 \\
\ddot{z}_d = -g + \left(c_{\phi_d} c_{\theta_d}\right) \frac{l}{m} U_1 \\
\ddot{x}_d = \left(c_{\phi_d} s_{\theta_d} c_{\psi_d} + s_{\phi_d} s_{\psi_d}\right) \frac{1}{m} U_1 \\
\ddot{y}_d = \left(c_{\phi_d} s_{\phi_d} s_{\psi_d} - s_{\phi_d} c_{\psi_d}\right) \frac{1}{m} U_1
\end{cases}
\tag{3}
$$

The first three equations describe the angular accelerations of the aircraft while the remaining three equations describe the UAV linear acceleration in the direction of x, y and z, respectively. The parameter J_r is the inertia for each rotor while Ω (eq. (4)) is the overall propeller speed. I_{xx}, I_{yy}, I_{zz} are the inertia components of the body about the x-axis, y-axis and z-axis, respectively. The total mass of the quad-copter is m and

$$
\Omega = -\Omega_1 + \Omega_2 - \Omega_3 + \Omega_4. \tag{4}
$$

The system inputs are reported in eqs. (5), where U_1 represents the throttle command (that acts on $\dot{z}_c$), U_2 represents the roll angle command (that acts on $\dot{\phi}_c$), U_3 is the pitch angle command (that acts on $\dot{\theta}_c$), while U_4 represents the yaw rate of change command (that acts on $\dot{\psi}_c$).

Finally, l is the distance to the center of gravity while b and d are the thrust and drag factor in hovering, respectively. Increasing or decreasing the speed of the propellers together

[6] Two are the possible configurations for the motors and the propellers in a quad-copter. One configuration looks like a plus sign ("+") while the other looks like a cross ("×"). The "+" configuration is distinguished by front and back propellers that make it easier to fly. Conversely, in the "×" configuration the propellers are less likely to get in the way of a front-facing camera since they are on the sides.

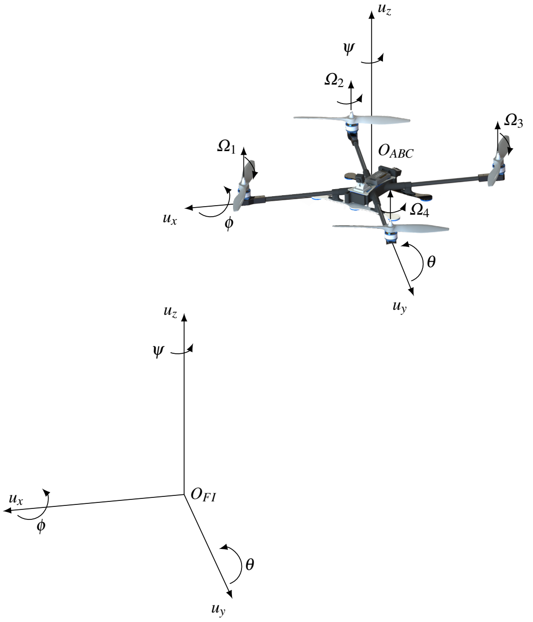


Fig. 15 Drone in the body-frame (O_{ABC}) and the fixed-frame (O_{FI}) reference system. The forces, the spin direction and the propellers velocity, Ω_i, from each rotor are also reported.

will determine the altitude change in position and velocity, while varying the speed of two propellers (Ω_1 and Ω_3) will cause the aircraft to tilt about the y-axis which is denoted as pitch angle (θ). Similarly varying the speed of the two propellers (Ω_2 and Ω_4) will cause the aircraft to tilt about the x-axis which is denoted as roll angle (ϕ). Finally, the vector sum of the reaction moment produced by the rotation of the pair Ω_1 and Ω_3 and the reaction moment produced by the ration of Ω_2 and Ω_4 will cause the quad-rotor to spin about its axis (z-axis) which is denoted as yaw angle (ψ). These are the six systems DOFs consisting of the position x_d, y_d and z_d and the orientation ψ_d, ϕ_d and θ_d. Further details are reported in [37], while the employed parameters values of the drone are listed in Tab. 1.

$$U_1 = b\left(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2\right) \tag{5a}$$

$$U_2 = b\left(-\Omega_2^2 + \Omega_4^2\right) \tag{5b}$$

$$U_3 = b\left(\Omega_1^2 - \Omega_3^2\right) \tag{5c}$$

$$U_4 = d\left(-\Omega_1^2 + \Omega_2^2 - \Omega_3^2 + \Omega_4^2\right) \tag{5d}$$

6 Flight Control System

The flight control system follows the same idea discussed in [37], with a reference generator that uses the information extracted from the images to generate the path to follow, and the

	Sym.	Value	Unit
Mass	m	0.65	[kg]
Distance to center of gravity	l	0.23	[m]
Motors inertia	J_r	$6.5 \cdot 10^{-5}$	[kg m^2]
Thrust factor	d	$7.5 \cdot 10^{-7}$	[1]
Drag factor	b	$3.13 \cdot 10^{-5}$	[1]
Inertia component along x-axis	I_{xx}	$7.5 \cdot 10^{-3}$	[kg m^2]
Inertia component along y-axis	I_{yy}	$7.5 \cdot 10^{-3}$	[kg m^2]
Inertia component along z-axis	I_{zz}	$1.3 \cdot 10^{-3}$	[kg m^2]

Table 1 System design variables. The units are reported in the SI expressed in terms of base units.

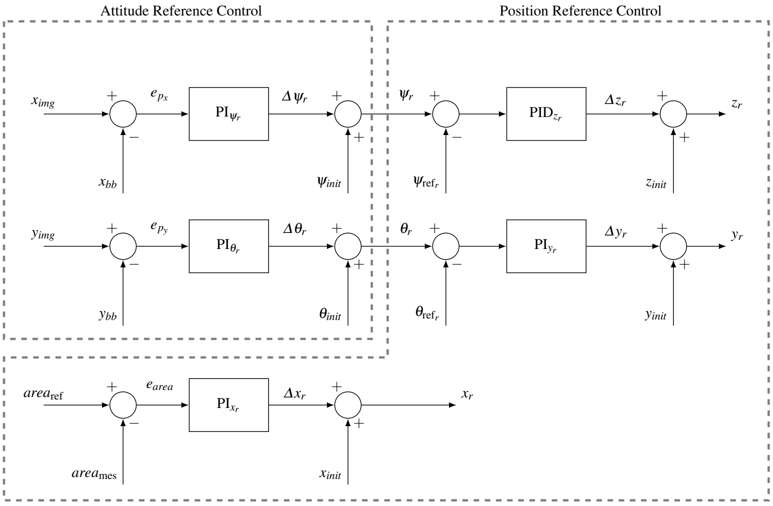


Fig. 16 The reference generator scheme, referred to virtual fixed reference system (O_{FVR}). The obtained heuristic PID gains are: $K_{P_{\psi_r}} = 1 \cdot 10^{-5}$, $K_{I_{\psi_r}} = 1 \cdot 10^{-3}$, $K_{P_{\theta_r}} = 1 \cdot 10^{-5}$, $K_{I_{\theta_r}} = 1 \cdot 10^{-3}$, $K_{P_{x_r}} = 1 \cdot 10^{-6}$, $K_{I_{x_r}} = 6 \cdot 10^{-6}$, $K_{P_{y_r}} = 1 \cdot 10^{-2}$, $K_{I_{y_r}} = 1 \cdot 10^{-2}$, $K_{P_{z_r}} = 15$, $K_{I_{z_r}} = 57.5$ and $K_{D_{z_r}} = 3.75$.

integral backstepping (IB) controller for the trajectory tracking. Figures 16 and 18 describes the whole control scheme, whereas in Figs. 3 and 2 represent as the blocks are connected to each other. In the event that the car is not detected, the drone keeps the reference computed in the previous step until the next revelation.

6.1 Reference Generator

Also the reference generator is decomposed in two parts: the attitude and the position controller, both illustrated in Fig. 16. The attitude controller tunes the yaw (ψ_r) and pitch (θ_r) angles, while roll (ϕ_r) remains equal to zero, trying to overlap the image (x_{img}, y_{img}) and the bounding box (x_{bb}, y_{bb}) centroids (see, Fig. 14). The angles are later used to tune the reference position x_r, y_r, and z_r with respect to the virtual world reference (see, Fig. 5).

The references generator takes also into account the camera (the drone) initial position (x_{init}, y_{init} and z_{init}) and orientation (θ_{init}, ψ_{init} and ϕ_{init}), in addition to its movements in the virtual environment. In our cases study, we assumed that the drone is flying 4 meters from

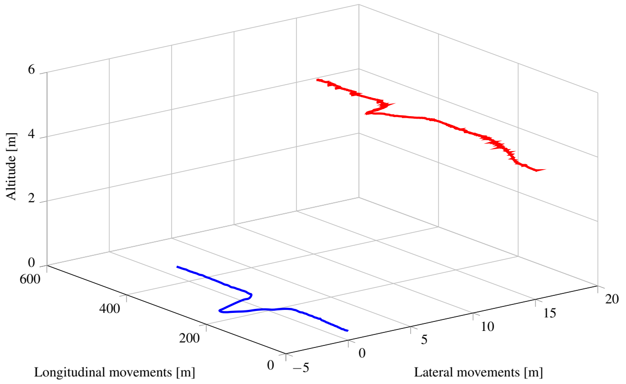


Fig. 17 The car (in blue) and the reference (in red) path described during the target following by neglecting UAV dynamics.

the ground, with a distance of 15 with respect the car. That hypothesis is not a limitation but simplifies the testbeds.

The PI_{ψ_r} and PI_{θ_r} outputs are the variations Δ_{ψ_r} and Δ_{θ_r} of ψ_r and θ_r that are used by the IB controller as path reference (see, Fig. 18) and to compute y_r (see, Fig. 16), respectively. Similarly it happens for the variable x_r and z_r, that need the aircraft initial positions x_{init} and z_{init}. The values ψ_{ref_r} and θ_{ref_r} (see, Fig. 16), both equal to zero, are the attitude references that the aircraft assumes during the target tracking. Finally, the error signal e_{area} is given by difference between the bounding box area ($w_{bb} \cdot h_{bb}$, aka $area_{\text{mes}}$ in Fig. 16) and the reference area ($area_{\text{ref}}$, equal to $30\,000\,\text{p}x^2$). Such reference, obtained as the sample mean of the ROIs collected during the classifier learning phase, allows to tune the distance x_r.

Figure 17 reports the trajectories followed by the car and the drone while a further video [60] has been made available for showing the results of the proposed approach. In such video, the quad-copter dynamic has been neglected to show how the reference generator works.

6.2 Integral Backstepping controller

The integral backstepping of [37] has been used as the controller for the trajectory of path tracking. It combines the robustness against disturbances offered by backstepping with the sturdiness against model uncertainties guaranteed by the integral action. The scheme in Fig. 18 summaries as the system controller works.

The trajectory control strategy works for making the aircraft attitude ($\theta_{\text{ref}_{IB}}$ and $\phi_{\text{ref}_{IB}}$) to follow the reference generators outputs (x_r and z_r). The reference signals $\phi_{\text{ref}_{IB}}$ and $\theta_{\text{ref}_{IB}}$ are computed (see, Fig. 18) as:

$$\theta_{\text{ref}_{IB}} = \frac{m}{U_1}\left[\left(1-c_1^2+\lambda_1\right)e_x + \left(c_1+c_2\right)e_{x_{IB}} - c_1\lambda_1\int_0^t e_x(\tau)\mathrm{d}\tau\right] \tag{6a}$$

$$\phi_{\text{ref}_{IB}} = -\frac{m}{U_1}\left[\left(1-c_3^2+\lambda_2\right)e_z + \left(c_3+c_4\right)e_{z_{IB}} - c_3\lambda_2\int_0^t e_z(\tau)\mathrm{d}\tau\right], \tag{6b}$$

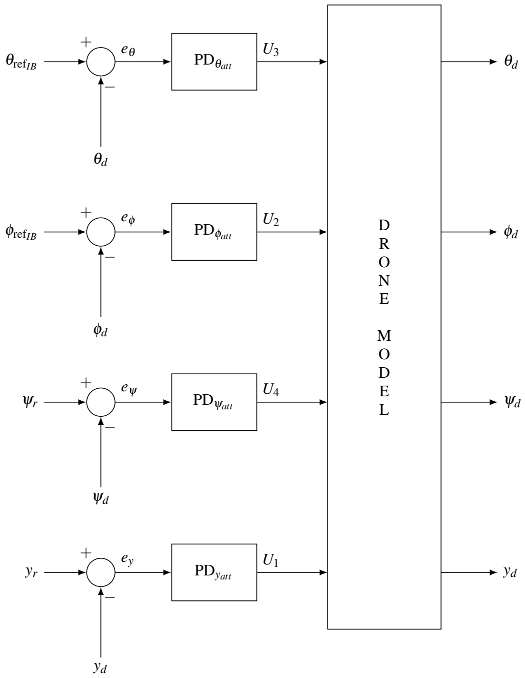


Fig. 18 Attitude and position control scheme using PID technique. U_1, U_2, U_3 and U_4 are the physical drone inputs: throttle ($\dot{z}_c$), roll (ϕ_c), pitch (θ_c), and yaw rate ($\dot{\psi}_c$), respectively. For the simulation we used the gains $K_{P_{y_{att}}} = 1000$, $K_{D_{y_{att}}} = 200$, $K_{P_{\phi_{att}}} = 8$, $K_{D_{\phi_{att}}} = 4$, $K_{P_{\theta_{att}}} = 12$, $K_{D_{\theta_{att}}} = 4$, $K_{P_{\psi_{att}}} = 10$ e $K_{D_{\psi_{att}}} = 4$.

with

$$e_{x_{IB}}(t) = \lambda_1 \int_0^t e_x(\tau)\mathrm{d}\tau + c_1 e_x(t) + \dot{e}_x(t) \tag{7a}$$

$$e_{z_{IB}}(t) = \lambda_2 \int_0^t e_z(\tau)\mathrm{d}\tau + c_3 e_z(t) + \dot{e}_z(t), \tag{7b}$$

and

$$e_x = x_r - x_d \tag{8a}$$

$$e_z = z_r - z_d, \tag{8b}$$

where (c_1, c_2, c_3, c_4, λ_1 and λ_2) are positive constants. For ours 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 $c_4 = 0.5$.

6.3 Numerical results

The overall system has been simulated in Matlab and the results illustrate in a direct way (the video is available, [61]) how the system performs. In the video both the image and the bounding box centroids as well as the distance vector are shown underling as the detection and tracking algorithms work. Also, the drone dynamics together with the fly control system are included in the loop: the quad-copter tilt around the x-axis into the virtual world reference system to follow the car along the path.

Starting from it, different scenarios have been considered pointing out the easiness and effectiveness of the proposed software platform. In [62, **?**], two cars were simulated: the yellow one switches the ESP control unit off, while the red one remains it on. The example allows to prove the robustness of the vision based target detection system. Indeed, the tracking algorithm resize the search window in according to the colored area avoiding the target loss. Thanks to the VRML97 EXTERNPROTO format, it is also possible to add a third vehicle into the scenario creating increasingly complex structures.

Conversely, in [63] the target (the red car, in whom the ESP is disabled) is placed behind the yellow car partially covering it. Nevertheless, the tracking vision system is able to follow the car for entire path until the simulation stops.

Finally, in [64] the world has been changed (the *vr_octavia* example was used, [65]), keeping invariant the other blocks. The example proves how the detection and tracking algorithms are able to work in scenarios different from those in which they were trained proving the good results of the learning phase.

Those results demonstrated as the system works and how it is possible to modify the initial configuration simulating any custom scenario, controller or vision based target system, too. The software platform allowed to test complex systems composed by computer vision and control algorithms interacting among them and with moving objects dynamics.

The experiments as well as the software platform were developed with the 2015b release of Matlab, equipped with CVS and VR toolboxes, but it is compatible with any Matlab release successive to 2015a. 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 world, 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 a easy way. Nevertheless, in our opinion the work could constitute the first step towards the development of a more structured platform aimed for the simulation-in-the-loop of such kind of applications. The idea is to rely on more flexible and dedicated solution like V-REP [21] or Gazebo [20, 66].

Furthermore, this paper illustrated how to expand the functionalities of the software platform for modeling and integrating any vehicle in the simulation framework. The overall approach aimed at developing the system in a modular way by facilitating the reuse of software modules already available and the reuse of the same modules for other projects. In particular the software framework allows to combine different vision-based and control algorithms, evaluating the performances and how they interact with the surrounding scenario for educational purposes making the tool potentially endless.

The simulator designed for fast simulation, even for educational purposes (for the well-know reasons), of autonomous multi-aerial systems has been released as open source framework via a Version Control System (VCS), such as GitHub, essential for testing and sharing code.

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. M. Adams, M. L. Levitan, and F. J. Carol, "High Resolution Imagery Collection for Post-Disaster Studies Utilizing Unmanned Aircraft Systems (UAS)," *Photogrammetric Engineering and Remote Sensing*, vol. 80, no. 12, pp. 1161–1168, 2014.
3. D. Erdos, A. Erdos, and S. E. Watkins, "An experimental UAV system for search and rescue challenge," *IEEE Aerospace and Electronic Systems Magazine*, vol. 28, no. 5, pp. 32–37, 2013.
4. 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.
5. C. Eschmann, C. Kuo, C. Kuo, and C. Boller, "Unmanned aircraft systems for remote building inspection and monitoring," in *Proceedings of the 6th European Workshop - Structural Health Monitoring*, vol. 2, 2012, pp. 1179–1186.
6. 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.
7. B. Kamel, M. Santana, and T. De Almeida, "Position estimation of autonomous aerial navigation based on hough transform and harris corners detection," in *International conference on Circuits, Systems, Electronics, Control and Signal Processing*, 2010, pp. 148–153.
8. K. Kanistras, G. Martins, M. Rutherford, and K. P. Valavanis, "A survey of unmanned aerial vehicles (UAVs) for traffic monitoring," in *International Conference on Unmanned Aircraft Systems*, 2013, pp. 221–234.
9. J. Xu, G. Solmaz, R. Rahmatizadeh, D. Turgut, and L. Boloni, "Animal monitoring with unmanned aerial vehicle-aided wireless sensor networks," in *IEEE 40th conference on local computer networks*, 2015, pp. 125–132.
10. 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.
11. C. Bills, J. Chen, and A. Saxena, "Autonomous MAV flight in indoor environments using single image perspective cues," in *IEEE International conference on robotics and automation*, 2011, pp. 5776–5783.
12. 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.
13. 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.
14. 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.
15. T. Hinzmann, J. L. Schönberger, M. Pollefeys, and R. Siegwart, "Mapping on the Fly: Real-Time 3D Dense Reconstruction, Digital Surface Map and Incremental Orthomosaic Generation for Unmanned Aerial Vehicles," in *Field and Service Robotics*, M. Hutter and R. Siegwart, Eds. Springer International Publishing, 2018, pp. 383–396.
16. I. A. Sucan and S. Chitta, "Moveit!" 2013. [Online]. Available: http://moveit.ros.org/
17. 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.
18. R. Diankov and J. Kuffner, "Openrave: A planning architecture for autonomous robotics," Robotics Institute, Pittsburgh, PA, Tech. Rep. 79, 2008.
19. A. Elkady and T. Sobh, "Robotics middleware: A comprehensive literature survey and attribute-based bibliography," *Journal of Robotics*, 2012.
20. 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.
21. 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.
22. O. Michel, "Cyberbotics ltd. Webots professional mobile robot simulation," *International Journal of Advanced Robotics Systems*, vol. 1, pp. 39–42, 2004.

23. S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles," in *Field and Service Robotics*, 2017. [Online]. Available: https://arxiv.org/abs/1705.05065
24. 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.
25. 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. PP, no. 99, pp. 1–8, 2017.
26. G. Silano, "Mat-Fly GitHub Repository," 2018. [Online]. Available: https://github.com/gsilano/MAT-Fly
27. S. Hutchinson, G. D. Hager, and P. I. Corke, "A tutorial on visual servo control," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 5, pp. 651–670, 1996.
28. 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.
29. 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.
30. 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.
31. S. B., S. L., V. L., and O. G., *Robotics: Modelling, Planning and Control*. Springer-Verlag, 2009, ISBN, 978-1846286414.
32. 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.
33. G. R. Bradski, "Computer vision face tracking for use in a perceptual user interface," *Intel Technology Journal*, 2nd Quarter 1998.
34. E. Rosten and T. Drummond, *Machine Learning for High-Speed Corner Detection*. Springer Berlin Heidelberg, 2006, pp. 430–443.
35. MathWorks, "Virtual Reality World and Dynamic System Examples," Mathworks Online documentation resources. [Online]. Available: https://goo.gl/rtEx3S
36. R. Carey and G. Bell, *Annotated VRML 97 Reference Manual*. Addison Wesley, 1997, ISBN, 0-201-41974-2.
37. S. Bouabdallah, P. Murrieri, and R. Siegwart, "Towards Autonomous Indoor Micro VTOL," *Autonomous Robots*, vol. 18, no. 2, pp. 171–183, 2005.
38. R. M. Murray, Z. Li, and S. S. Sastry, *A Mathematical Introduction to Robotic Manipulation*. Boca Raton, FL: CRC Press, 1994.
39. R. Lienhart, A. Kuranov, and V. Pisarevsky, *Empirical Analysis of Detection Cascades of Boosted Classifiers for Rapid Object Detection*. Springer Berlin Heidelberg, 2003, pp. 297–304.
40. G. Silano, "Frames acquisition video," YouTube. [Online]. Available: https://youtu.be/A70zed84zv0
41. ——, "Positive and negative images acquired from the virtual scenario." Mega. [Online]. Available: https://github.com/gsilano/MAT-Fly#basic-usage
42. ImageAnalyst, "Image Segmentation Tutorial," MathWorks File Exchange. [Online]. Available: https://goo.gl/mn8dhJ
43. 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.
44. L. He, X. Ren, Q. Gao, X. Zhao, B. Yao, and Y. Chao, "The connected-component labeling problem: A review of state-of-the-art algorithms," *Pattern Recognition*, vol. 70, pp. 25–43, 2017.
45. E. R. Davies, *Computer and Machine Vision: Theory, Algorithms, Practicalities*. Academic Press, 2012, vol. 4.
46. 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.
47. 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.
48. ——, "A hybrid vehicle detection method based on viola-jones and hog + svm from uav images," *Sensors*, vol. 16, no. 8, 2016.
49. K. Pulli, A. Baksheev, K. Kornyakov, and V. Eruhimov, "Realtime Computer Vision with OpenCV," *Communications of the ACM*, vol. 55, no. 8, pp. 61–69, 2012.
50. MathWorks, "Computer System Toolbox." [Online]. Available: https://goo.gl/gBHQuJ

51. P. Viola and J. J. Michael, "Robust real-time face detection," *International Journal of Computer Vision*, vol. 57, no. 2, pp. 137–154, May 2004.
52. O. T., M. Pietikainen, and T. Maenpaa, "Multiresolution Gray-scale and Rotation Invariant Texture Classification With Local Binary Patterns," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 971–971, 2002.
53. D. N. and B. Triggs, "Histograms of Oriented Gradients for Human Detection," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1, pp. 886–893, 2005.
54. D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, vol. 2, 1999, pp. 1150–1157.
55. ——, "Distinctive Image Features from Scale-Invariant Keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.
56. T. Suwan, T. Naddao, and N. Matthew, *Dailey. Rapid detection of many object instances*. Springer Berlin Heidelberg, 2009, vol. 5807, pp. 434–444.
57. A. R. Smith, "Color Gamut Transform Pairs," in *SIGGRAPH 78 Conference Proceedings*, 1978, pp. 12–19.
58. MathWorks, "Computer Vision System Toolbox Examples." [Online]. Available: https://goo.gl/2MdAwL
59. K. P. Valavanis, *Advances in Unmanned Aerial Vehicles*. Springer, 2007.
60. G. Silano, "Drone follows a car: the drone dynamics and control is not in the loop," YouTube. [Online]. Available: https://youtu.be/qAtndBIwdas
61. ——, "Drone follows the car along a non trivial path," YouTube. [Online]. Available: https://youtu.be/b8mTHRkRDmA
62. ——, "Drone follows the red car superimposed on the yellow one," YouTube. [Online]. Available: https://youtu.be/xedC3ezuCz4
63. ——, "Drone follows a car partially covered," YouTube. [Online]. Available: https://youtu.be/RjXBtPqZZBc
64. ——, "Drone follows a car along a non trivial path (vr_octavia scenario)," YouTube. [Online]. Available: https://youtu.be/m43Zadq-6XM
65. MathWorks, "Vehicle dynamics visualization." [Online]. Available: https://goo.gl/Sd8rHf
66. 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.