VOL. 18, NO. 3, FEBRUARY 2023 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences

www.arpnjournals.com

OBSTACLE DETECTION AND AVOIDANCE SYSTEM PERFORMED WITH DRONE USING ROS AND PYTHON

Karen Cerón, David Reina and Faiber Robayo Betancourt
Departamento de Ingeniería Electrónica, Facultad de Ingeniería, Universidad Surcolombiana, Neiva, Huila, Colombia
E-Mail: faiber.robayo@usco.edu.co

ABSTRACT

The AR Drone 2.0 obstacle detection and avoidance system using ROS (Robot Operating System) as a development environment, OpenCV as a library for image processing, and Python as a programming language are presented. The algorithm is based on the object's color feature to determine whether it is an obstacle to allowing autonomous flights to the drone if the drone's battery charge level and its distance from the ground station are considered. The obstacle detection scenario is simulated using ROS-Gazebo, and then the algorithm is tested in a real controlled environment. It shows the proper functioning of the system, that is, it manages to detect a red object correctly and avoid it by the control actions implemented in the algorithm.

Keywords: algorithm, ROS, Hue Saturation Value, controlled environment, drone.

1. INTRODUCTION

The UAV (Unmanned Aerial Vehicle) was developed some time ago. However, nowadays, its use has increased exponentially due to its easy access. Drones are the most popular ones as they are tools that can carry out multiple applications that simplify human work. For example, they can use them in journalism to capture different perspectives (Gynnild and Uskali, 2018), in crop spraying (Pharne *et al.*, 2018), for goods transport and shipment (Jones, 2017), in search and rescue, in agriculture, among others. Drones are characterized by several motors (4, 6, or 8) with direct drive on their respective blades. Independent control of each rotor's rotation speed allows the vehicle to advance, change direction, or remain floating in a fixed position (Barreiro and Valero, 2015).

These UAVs have a communication system (usually radio control and/or WiFi) for an operator to control such an aerial robot. Regardless of the task being performed; the operator is located in a strategic location, where he makes eye contact with the drone while performing proper maneuvers to avoid accidents. Operator distraction or low visibility can lead to drone crashes against objects. The robot operator can make mistakes when performing an activity with the drone, either out of fatigue, in a hurry, by the person's reaction time, or by any other factor involving human failures.

Drones can be controlled using alternative tools such as development environments, image processing, and neural networks, among others. Works such as the Polytechnic University of Cartagena propose the visual control of the AR quadrotor Drone using OpenCV, ROS, and C++, but in a simulated way using the Gazebo environment containing the graphical representation of the model of the drone in question. The results obtained were good except for a small problem with the HSV (Hue Saturation Value) transformation and in the adjustment of the color of the object, because even human skin is detected so the person moving the object must as far as possible be covered with garments (Banach, 2016).

A height control system of the AR quadrotor Drone 2.0 applying fuzzy logic in a controlled environment is proposed (Robayo, 2019). Three controllers are developed using fuzzy logic whose parameters are obtained from the sensors in such a way that it allows control of the height and orientation angles such as Pitch, Roll, and Yaw. Also, they design and implement an interface in MatLab that provides communication between the user and all system functions. The implemented interface allows the choice of the execution mode, follows the reference parameters autonomously, and stores data for further analysis.

ROS is a collection of frameworks for robot software development. ROS was initially developed in 2007 under the name of a switchyard by Stanford's Artificial Intelligence Laboratory to support the Robot with Artificial Intelligence project (ROS, 2018). Since 2008, the development has continued mainly at Willow Garage, a robotic research institute with more than twenty institutions collaborating on a federated development model (Ortega, 2017).

OpenCV is a free machine vision library developed by Intel, which since 1999 has been used in all kinds of applications that require incorporating object recognition (Cheblender, 2018). In recent years, the wide use of ROS and OpenCV has constituted them as essential and fundamental tools for robotic applications where it is necessary to work with image processing. So far, various techniques such as PTAM, and Dense Depth Mapping, among others, have been used to avoid obstacles.

Work proposes to design and implement an obstacle detection and avoidance algorithm based on the object color characteristic. The algorithm is tested in a controlled environment using the AR Drone 2.0 quadrotor with programming in OpenCV, ROS, and Python. Instrumentation is essential when executing the control, so sensors such as the camera and height sensor are used. The use of the ROS development environment makes it possible to adjust both linear and angular drone speeds. The OpenCV library's operation allows the processing of the image obtained in real-time thanks to the cv bridge

package, which allows converting the image obtained from ROS for further processing.

2. MATERIALS AND METHODS

Figure-1 represents the block diagram of the proposed system. At first, the ground station is connected to the drone via WiFi, the algorithm is initialized, and the drone camera data is obtained. Image processing is then performed to perform obstacle detection and take the necessary avoidance control actions.

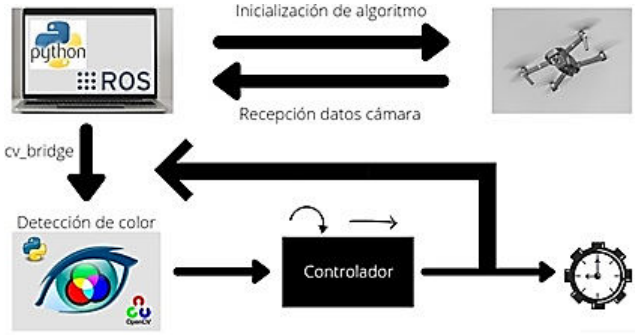

Figure-1. Diagram of the implemented system block.

2.1 OpenCV and cv_bridge

To convert ROS format images to an image type that can be manipulated in OpenCV, the cv_bridge package (Navarro and Angulo, 2015) is used. This package contains CvBridge, a ROS library that provides an interface between ROS and OpenCV. The use of this tool enables the acquisition of data from the robot camera in real time. It is necessary to import the image obtained from the drone, subsequently subscribe to the "topic" of the drone camera, and carry out the format conversion of the image for this case from ROS to BGR. Figure-2 shows the process for converting image format using CvBridge.

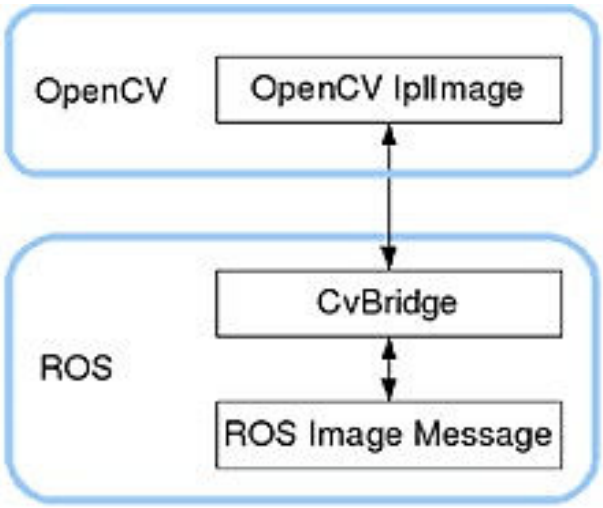

Figure-2. Image conversion process using CvBridge.

2.2 Color Obstacle Detection

For the obstacle detection stage, it is necessary to set the range covered by the color to be detected using the HSV format. The NumPy library is used to create arrangements with the minimum and maximum values of that range. Red detection is performed for this work, considering that the red color appears in two different HSV format spaces. Figure-3 shows the values for HSV format components (Solano, 2019).

For the detection of an object, a mask must be created that gathers the colors covered by the set ranges, and an image format change (BGR to HSV) is appropriate. The Range() function of the OpenCV, which the library receives as parameters for the image on which it is possible to perform color detection, the lower bound of that color, and the upper bound are used as parameters. The image that returns this function is of binary type, where the appearance of white color represents the detection of the color specified in the ranges set above. In this particular case, it is necessary to have the OpenCV libraries add () function to put together the two masks created in one.

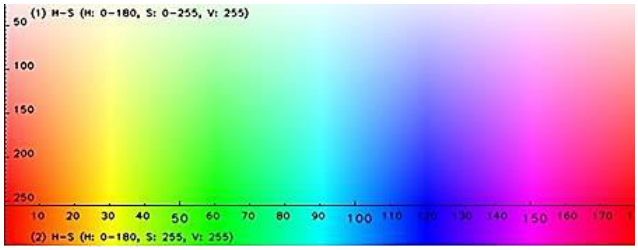

Figure-3. View of HSV components.

To display the detected color outline, use the find Contours () function of the OpenCV library. The center of the largest outline detected in the image is located and enclosed in the smallest possible circle encompassing the entire detected object. It is possible also to find the "x" coordinate and the "y" coordinate using the OpenCV moments () function and print them on the screen using the rospy loginfo () function.

Concerning the creation of the circle enclosing the detected object, the size of the radius obtained by the min Enclosing Circle() function is taken into account; if that size exceeds 10 pixels, then the enclosed object is displayed in the circle created with the help of the circle() function of the OpenCV library.

When detecting an obstacle, the coordinates found are always positive, so it is optional to create variables that contain negative values (-1 and -2) as a mechanism to perform control actions in case no obstacle is detected. Following this, the final image is displayed with the help of the OpenCV imshow () function. The BGR format image is converted to ROS format and published using the ROS publish () function. It is also defined as the main () function of the algorithm in which the image_converter class object is created. The node is executed until it is interrupted by the keyboard. Finally, the node is created, and a value is defined that specifies the message rate or cup using the Rate () function and executes the main ().

2.3 Avoidance of Obstacles

For obstacle avoidance, a class called Detector () is created. Its due constructor is defined, and two variables with a value of None are set, which are intended to store the coordinate values published by the object detection algorithm. After that, the function responsible for performing control actions is implemented depending on whether the object detection algorithm's coordinates are positive or negative. Two publishers are created regarding the topics /cmd_vel and ardrone/land because they allow manipulating the drone's speeds and landing by sending messages and data, respectively. This function receives a parameter known as data, the "x" coordinate published by the obstacle detection algorithm, to assign it to one of the two variables set at the start of the avoidance algorithm. It is decided to work with one of the two coordinates because either represents the detection of an obstacle.

Once the "x" coordinate, published by the obstacle detection algorithm, is obtained, an if/else cycle is implemented that is responsible for printing the presenting situation (presence or not of an obstacle) and taking the appropriate control actions. If the "x" coordinate is negative, adjust the angular velocity relative to the z-axis to a value of zero, and the linear velocity relative to the x-axis is changed to a value of 0.1 meters/second. A variable is also created that automatically counters to set a flight limit value and land later (forward timer). Otherwise, when the "x" coordinate is positive, the linear velocity relative to the x-axis is set to a value of zero, and the angular velocity relative to the z-axis is set to a value of 0.3 radians/second. Similarly, a variable is set that serves as a timer for drone landing (turn timer). Figure-4 shows the drone's axes with their respective directions (Maravall et al., 2017).

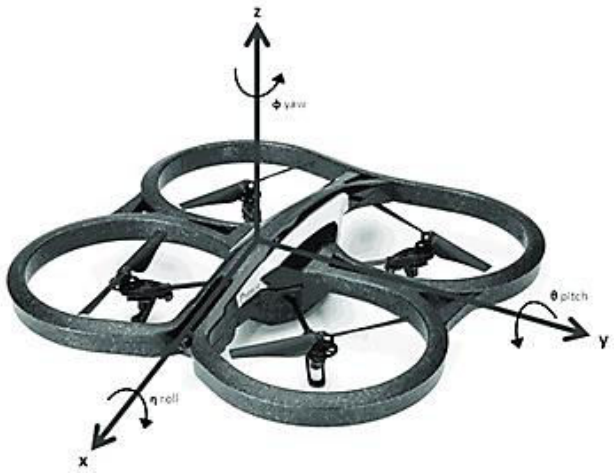

Figure-4. AR. Drone 2.0 with their respective axes.

The proposed algorithm allows the drone to detect and avoid red obstacles, always turning in a positive direction to the z-axis (anti-clockwise direction). Figure-5 illustrates the direction of rotation to the z-axis.

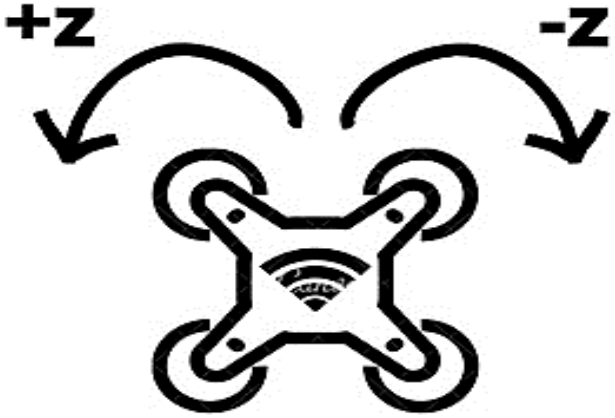

Figure-5. The direction of rotation of the drone relative to the z-axis (Yaw).

Linear and angular speeds are then published depending on the situation the drone is witnessing, and the display of both turn and forward timers and two control cycles are set for these timers. The robot is then allowed to land after a set time by publishing the topic /ardrone/land.

Finally, the node is created for this algorithm, the creation of the class object, the setting of timer variables as global, and their respective initialization to zero, as well as the creation of subscribers, to receive the data published by the object detection algorithm.

3. RESULTS AND DISCUSSIONS

3.1 Simulation of the Detection Scenario

Figure-6 shows the simulation of the obstacle detection scenario with AR Drone 2.0 using ROS-Gazebo. The Gazebo is a tool that allows simulating environments in three dimensions to estimate a robot's behavior (drone) in a virtual environment. The left side of the figure, it can see the drone and the obstacle in front of it. On the right side of the figure is evidenced by the detection of the obstacle. At the bottom, the coordinates "x", "y" of its center.

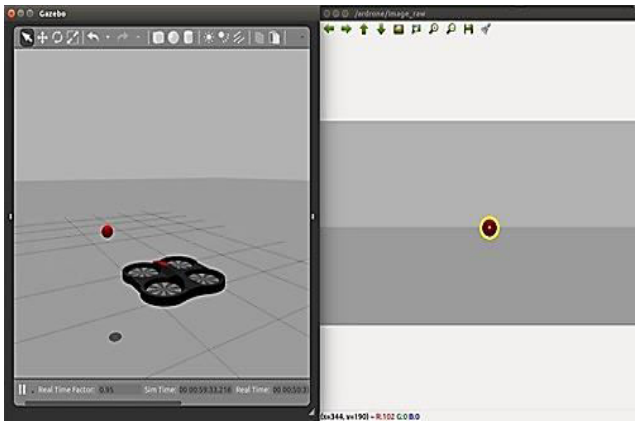

Figure-6. Simulation of obstacle detection with AR. Drone 2.0 using Gazebo.

3.2 Detection and Avoidance in a Real Controlled Environment

Figure-7 shows real-time obstacle detection using AR Drone 2.0. This figure shows two scenarios, the first of which is the drone with the obstacle located in front of it, and the second is the ground station obtaining the image of the drone's front camera with its respective detection.

Figure-7. Obstacle detection with AR. Drone 2.0 in real-time.

The algorithm is tested, and the proper functioning of the system is evident. That is, it manages to detect a red object correctly. According to tests performed with the largest obstacle, it can recognize it up to an average distance of 2.5 meters. It then avoids the obstacle using the control actions implemented in the algorithm. It is possible by changing the angular velocity until the obstacle is outside the drone's visual range to modify the linear velocity and follow a straight path until it detects another red obstacle in its path. As a safety measure implemented for the system, limit times are set, which makes it possible for a safe landing of the drone to be generated when exceeded. By turning the drone continuously for 17.5 seconds, it lands satisfactorily, according to the tests performed. Also, by continuing in a straight line during this period, the drone performs its landing correctly. Figure-8 shows the trajectory made by the drone during testing in the controlled environment.

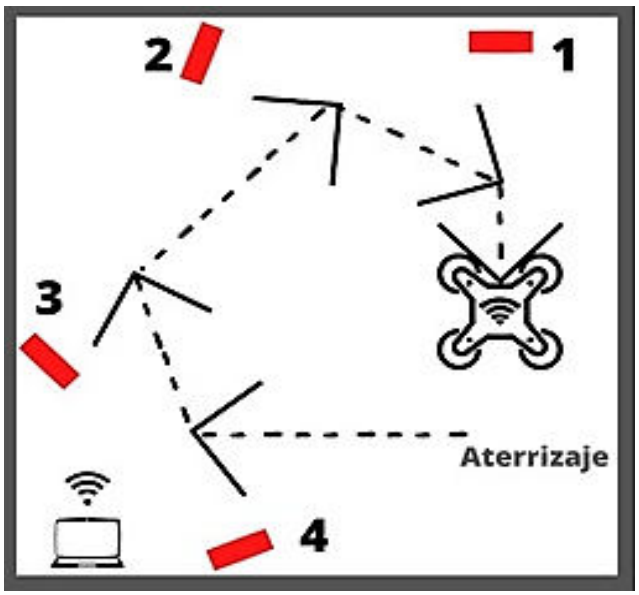

Figure-8. Trajectory made by the drone.

3.3 Record of Heights in Tests Carried Out

The algorithm's effectiveness can be affected by the state of the drone, as due to the wear of its components, it can cause there to be variations in its performance and obtain unwanted results. The results obtained for height vs. time in two of the tests performed are presented below.

Figure-9 shows the curve representing the height data relative to the time for the first chosen test. The analysis that is performed covers the time range for which the height is different from zero so the average height for this test is 710.22 mm. The initial percentage of the battery charge for this test is 64%, and the final percentage is 41%.

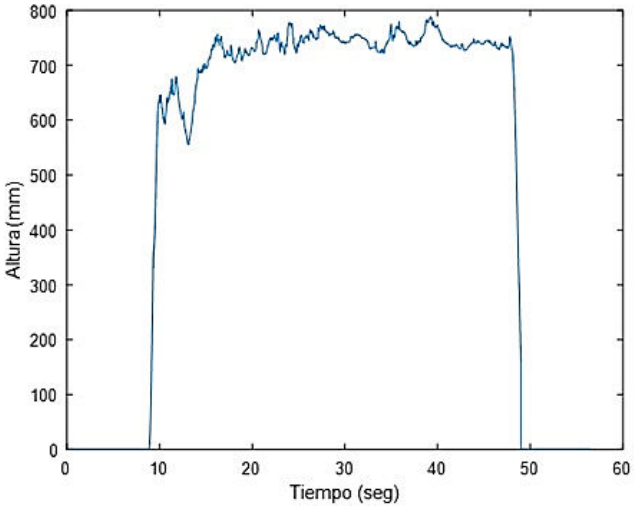

Figure-9. Height vs. time plot for the first test.

Figure-10 shows the curve that represents the height versus time data for the second chosen test. This test's initial drone battery charge is 49%, and the final charge is 11%.

The time interval to be analyzed is from the second 11.78 to the second 107.78. During this period, the algorithm presented is executed, and the average height for this test is 635.60 mm. According to Figure-10, the average height for this test is very low due to the battery reduction charge percentage of the drone during the test.

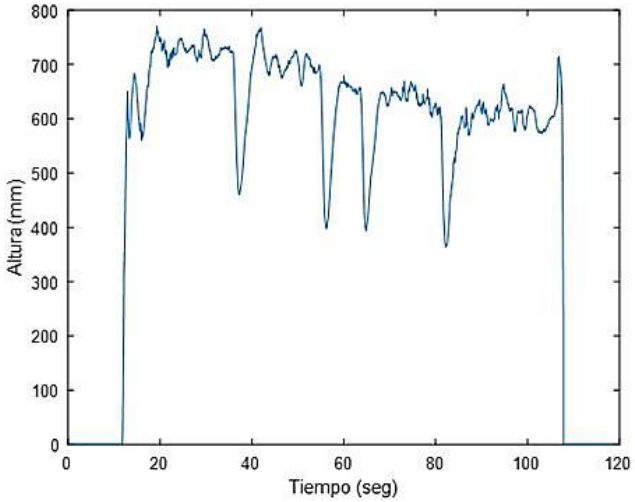

Figure-10. Height vs. time plot for the second test.

Figure-11 shows the orientation data on the Z-axis (yaw) for the time for the first chosen test. This figure shows that the drone has an initial orientation of approximately -145 degrees (regardless of the first nine seconds time interval approximately when the drone is at rest). This initial orientation is the drone's value in degrees relative to the z-axis when executing the takeoff command. In the designed system, the drone performs counter clockwise avoidance, i.e., when detecting an obstacle, it executes the positive turn of the z-axis (yaw). It is concluded that the upward sections represent the rotation of the drone, looking at the curve obtained in Figure-11 because its orientation relative to the z-axis is changing. However, the first two sections are related to the avoidance of the first obstacle, and the third upward section corresponds to the avoidance of the second obstacle. Finally, the drone is landed manually because it is impossible to wait 17.5 seconds as a safety measure for landing due to lack of space in the enclosure.

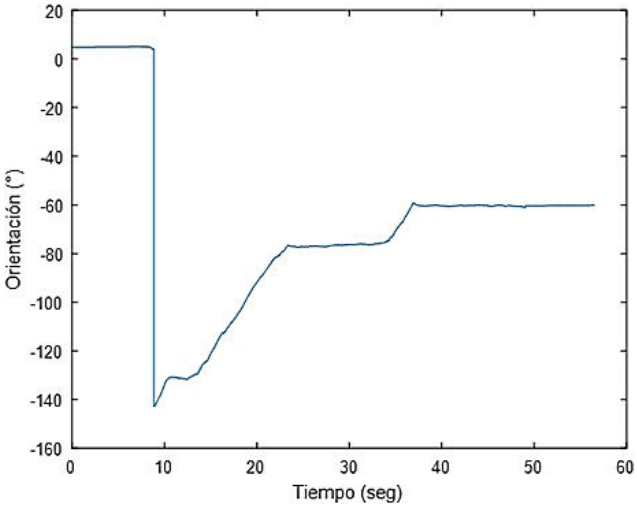

Figure-11. Z-axis orientation plot vs. time for the first test.

3.4 Comments on the Results

This work's contribution is academic in type because technologies that are also interesting are used and that in our region have been little explored.

ROS is composed; its architecture provides utilities such as obtaining data (images and sensory) for further processing and modifying variable values (engine speeds) to perform desired control actions. The results obtained when using this development environment were as expected.

It highlights the cv_bridge of the package to obtain images in real-time on other tools such as the FFmpeg software and the OpenCV Video Capture function. During the entire process of constructing the algorithm, these three tools were tested, finding that obtaining the images using FFmpeg and Video Capture had a significant delay (approximately 15 seconds). This delay affects the entire system control mechanism making it inefficient, while when using the cv_bridge, the robot image is obtained immediately.

OpenCV software for image processing provides many facilities that make this a complete library, ideal for the various cases where image processing is required. However, the effectiveness of obstacle detection in the algorithm presented depends on the correct establishment of the color ranges to be detected. Adjusting an extensive range results in the detection of unwanted colors and, on the contrary, adjusting a minimal range is equivalent to not detecting all possible ranges of the chosen color.

The distance between the drone and the ground station is important for proper operation and, therefore, for the system's performance. Getting too far away from the drone means that the image obtained from it to the ground base freezes. Consequently, it is impossible to process the image in real-time; then, obstacle detection is not achieved, and control actions are taken to perform correct avoidance.

Finally, through the tests carried out in a controlled environment, it is possible to appreciate the system's correct functioning. It manages to perceive a red

object, and can recognize it as an obstacle, then avoid it through the control actions implemented in the algorithm. That is possible by the variation of the angular velocity until the obstacle is outside the drone's drone's visual range to modify the linear speed and follow a straight trajectory until it detects another obstacle in its path.

4. CONCLUSIONS

A red obstacle detection and avoidance algorithm was developed using the Python programming language, the ROS development environment, and the OpenCV image processing library. The project is affordable to anyone, as the tools mentioned above are open-source. On the other hand, some factors negatively influence the algorithm's performance, such as the state of the drone (blades, motors, and batteries) which makes it difficult to see the expected performance of the algorithm performed.

It is important to consider the WiFi network's coverage (scope) created by the drone when energized since exceeding this distance loses communication between the drone and the ground station (computer). This situation results in the loss of video transmission packets, and therefore no control actions are performed for the drone.

The ROS structure offers a lot of advantages among which it is worth highlighting the real-time data acquisition of the robot under study, such as obtaining camera images, and sensor data, as well as its large existing community that provides useful information on the web.

Highlights include real-time imaging performance by subscription to the drone camera topic versus using the Video Capture () function. This feature provides images with considerable delay time while delivered images are obtained for ROS in real-time. However, for obstacle detection and avoidance to be performed correctly, it is necessary to define the color ranges to be detected, and to do this factors such as the luminosity of the environment can turn the color of the object to a different one must be taken into account.

This work provides important foundations on a very little-known topic in the region, which is very useful, as is ROS's use for the implementation of algorithms in robots. The algorithm presented provides valuable information and functions as a guide for future projects or research, and the logic of the algorithm works with significant similarity in other robots, not only aerial but also terrestrial and aquatic.

REFERENCES

Banach A. 2016. Visual control of the Parrot drone with OpenCV, Ros and Gazebo Simulator. Repositorio Digital Universidad Politécnica de Cartagena. [On-Line]. Available in: https://repositorio.upct.es/handle/10317/5442.

Barreiro P. and Valero C. 2015. Drones en la agricultura. Feria del Sector Agropecuario. Universidad Politécnica de Madrid, España.

Cheblender. 2018. Qué es OpenCV. [Online]. Available in: https://www.cheblender.org/que-es-opencv/.

Gynnild A. and Uskali T. 2018. Responsible Drone Journalism. London, Taylor & Francis Group.

Jones T. 2017. International Commercial Drone Regulation and Drone Delivery Services. RAND Corporation. [On-Line]. Available in: https://doi.org/10.7249/rr1718.3.

Maravall D., de Lope J., Fuentes J. 2017. Navigation and Self-Semantic Location of Drones in Indoor Environments by Combining the Visual Bug Algorithm and Entropy-Based Vision. Universidad Politécnica de Madrid, España. https://doi.org/10.3389/fnbot.2017.00046.

Navarro J. and Angulo C. 2015. People exact-tracking using a Parrot AR. Drone 2.0. Universidad Politécnica de Cataluña, Barcelona, España.

Ortega D. 2017. Qué es ROS (Robot Operating System). [On-Line]. Available in: https://openwebinars.net/blog/que-es-ros/.

Pharne Mr. I. D., Kanase S., Patwegar S., Patil P., Pore A., Kadam Y. 2018. Agriculture Drone Sprayer. International Journal of Recent Trends in Engineering & Research (IJRTER). pp. 181-185.

Robayo F. 2019. Control of UAV based on fuzzy logic in a controlled environment. ARPN Journal of Engineering and Applied Sciences. 14(24): 4221-4227.

ROS. 2018. Robotic Operating System. [On-Line]. Available in: https://www.ros.org.

Solano G. 2019. Detección de colores en OpenCV [en 4 Pasos] - Parte1 [Archivo de video]. [Online]. Available in: https://www.youtube.com/watch?v=giwtDYcIXKA.