VOL. 18, NO. 4, FEBRUARY 2023 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences

www.arpnjournals.com

COLOR OBSTACLE DETECTION ALGORITHM USING AR DRONE 2.0 IN A CONTROLLED ENVIRONMENT

Faiber Robayo Betancourt[1] and Daniel Suescún-Díaz[2]
[1]Departamento de Ingeniería Electrónica, Facultad de Ingeniería, Universidad Surcolombiana, Neiva, Huila, Colombia
[2]Departamento de Ciencias Naturales, Avenida Pastrana, Universidad Surcolombiana, Neiva, Huila, Colombia
E-Mail: faiber.robayo@usco.edu.co

ABSTRACT

This work presents an obstacle detection algorithm using quadrotor AR. Drone 2.0 in a controlled environment with programming in OpenCV, ROS (Robot Operating System), and Python. The algorithm is based on the color object feature to determine it as an obstacle. It is the first step for future work where the drone is allowed to perform autonomous flights and obstacle avoidance. The ROS development environment makes it possible to adjust both the drone's linear and angular speed. The OpenCV management library allows processing the image obtained in real-time thanks to the cv_bridge package. The obstacle detection scenario is simulated using ROS-Gazebo, and then the algorithm is implemented in a real controlled environment. The results obtained were good; an obstacle detection system was successfully implemented using open-source tools.

Keywords: algorithm, OpenCV, Python, ROS, drone.

1. INTRODUCTION

In recent years, the wide use of technologies such as robots, artificial intelligence, machine learning, among others, have allowed great results such as autonomous systems, biometric systems, virtual assistants, and so on. These results allow people to carry out activities they used to do, but with the advantage of carrying them out less time and with greater precision. Drones or UAVs (Unmanned Aerial Vehicles) result from these technologies that surprise humanity with their appearance due to their wide variety of applications such as sports activities, advertising, photography, locating missing persons, among others. Different types of drones, depending on their use, characteristics, control method, and how they move, are found in the literature (Adeva, 2020).

Drones are characterized by having several motors with direct drive on their respective blades. The independent rotation speed control of each rotor allows the vehicle to move forward, change direction, or remain floating in a fixed position (Barreiro and Valero, 2015). These UAVs have a communication system (usually radio control and/or Wi-Fi) that allows an operator to control said aerial robot. Regardless of the task being carried out, the operator must be located in a strategic place. He can make visual contact with the drone while performing the proper maneuvers to avoid accidents. A distraction from the operator or poor visibility can cause the drone to crash against objects. The robot operator is not exempt from making mistakes when carrying out an activity with the drone, whether due to fatigue, haste, person reaction time, or any other factor that involves human failure.

Developed by Intel, OpenCV is a free computer vision library that, since 1999, has been used in all types 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 the different robotic applications required to work with image processing.

This work proposes an algorithm for detecting obstacles by color in a controlled environment using quadrotor AR. Drone 2.0 with programming in OpenCV, ROS, and Python. This work is the first step for future work where the drone can perform autonomous flights and obstacle avoidance. The instrumentation is essential when executing the control, so it is necessary to use sensors such as the camera and the height sensor. The use of the ROS development environment makes it possible to adjust both the linear and angular speeds of the drone, as well as the OpenCV library management 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

2.1 AR Drone 2.0

A drone can be defined as an unmanned aerial vehicle, UAV. Several types and forms of UAVs can develop different kinds of tasks (Hernandez *et al*., 2015). The Parrot company manufactures the drone chosen for this project. It has an autonomy of 12 minutes, a range of 50 meters, a high-definition camera of 720 p at 30 fps. In addition to taking photos allows you to record videos, the piloting is done through its application for smartphones or tablets. It has an ARM processor Cortex A8 32-bit 1 GHz with 800 MHz DSP video TMS320DMC64x. It also has a Linux 2.6.32 operating system with on-board sensors such as an accelerometer, magnetometer, barometer, ultrasonic height sensor, and gyroscope (Parrot, 2019).

The AR. Drone 2.0's onboard technology provides extreme precision control and automatic stabilization features. Figure-1 shows the drone's axes with their respective directions (Maravall *et al*., 2017).

Figure-1. AR. Drone 2.0 with their respective axes.

2.2 The Red Obstacle Detection Algorithm

For the proposed algorithm implementation in this work, it is necessary to establish the color in the range of the HSV (Hue Saturation Value) object format that will be determined as an obstacle, obtaining the image in real-time drone, and getting the coordinates of the obstacle. The color red is chosen for this work.

The algorithm carried out consists of two codes (Publisher and Subscriber). The Publisher code is responsible for obtaining the drone's image in real-time and processing the image to determine if there is an obstacle present and thus know its coordinates. The Subscriber code is responsible for performing the drone's relevant control actions based on the value of the coordinates obtained by the Publisher code.

2.2.1 Creating the class and defining functions

In the Publisher code (called CvBridge.py), a class called "image_converter" is created in which two functions are defined. The first function is the class constructor "__init__", which allows the creation of publishers, both of coordinates in x and y, and the publisher for the drone landing. The subscription to the drone camera's topic is also performed to obtain it in real-time in this function. The object "CvBridge" is created to convert the image obtained in ROS in the kind of image used in OpenCV. This class's second function is "callback", detailed in detail in the next item.

2.2.2 Establishment of color ranges in HSV format for obstacle detection

It is necessary to use the "numpy" library to establish the color used as a reference to determine an obstacle. This library allows creating arrangements that contain the values of the ranges in HSV format, as shown in Figure-2 (Solano, 2019). This type of color format is composed of the following three features:

Hue: It is conceived as a circle of colors that varies from red (0º) to red color again (360º), passing through colors like green (120º), blue (240º), and the other “pure” colors like yellow, cyan, purple, among others.

Saturation: This feature indicates how much of the color chosen in the Hue value will appear in the mix. It has a range that varies between 0 and 100%.

Value: It varies from 0 to 100% and represents the amount of black present in color chosen in the Hue component (where 0% is entirely black and 100% is the color at its maximum brightness).

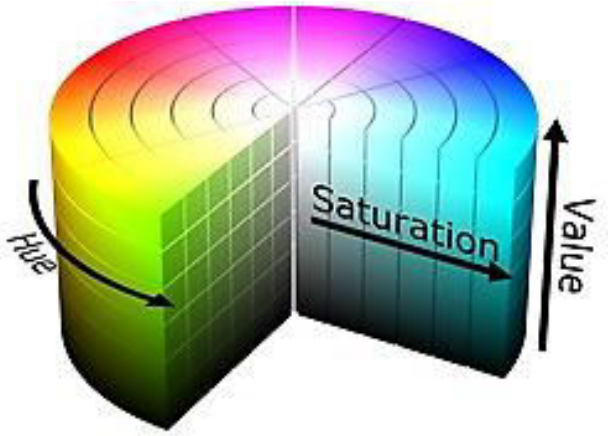

Figure-2. Representation of the HSV color format.

The "callback" function receives as an argument the "data" variable, which is the image obtained in real-time by the "__init__" function to convert it to the image type used in OpenCV. The processing of this image is also carried out, and the detection of obstacles present in the camera image is carried out. Using the "numpy" library, the color ranges in which the image object must be detected as an obstacle are defined.

2.2.3 Image type conversion and creation of masks

In the second section of the "callback" function, the conversion from ROS image type to OpenCV image type is performed. The "CvBrige" module imported from the "cv_bridge" class is used together with the "imgmsg_to_cv2" functionality. This conversion provides an OpenCV image in bgr8 format, which is later converted to the HSV format explained above, with the OpenCV library for Python known as "cv2". Next, the masks are created for each range with the help of the "inRange" function, which has parameters the image on which the color detection will be performed and the desired color lower and upper to be detected. Two masks are made because they are two different red color ranges for the HSV format.

2.2.4 Unification of masks and morphological operations

The mask unification is carried out to work with a single mask that contains the two ranges established above, using the "add" function. Subsequently, the morphological operators "erode" (erosion) and "dilate" (dilation) are used to complete the entire mask.

2.2.5 Contour creation

To visualize the detected obstacle's contour, use the "findContours" function from the "cv2" library. This feature receives as parameters the binary mask obtained previously, the contour recovery mode, and the contour approximation method.

2.2.6 Obtaining the obstacle coordinates

Initially, the variable "center" is created, but no value is assigned to it momentarily. Then an "if" loop is made in such a way that if the variable "cnts" (the contour) is greater than zero, then the largest contour present in the image is obtained using the Python "max" function. Besides, the center (x-coordinate, y-coordinate) and the smallest possible circle radius that encloses the detected obstacle are obtained. Following this, the moments of the image are calculated using the OpenCV function "moments". This function allows you to calculate specific properties of an image such as radius, area, centroid, among others. With this function, the circle centroid enclosing the obstacle is calculated. Two variables, "b" and "a" are created, containing the circle x and y coordinates that enclose the detected obstacle. The correct x and y coordinates of the obstacle are printed on the Linux terminal Using the "rospy" library. Finally, the "sleep" function of the "rospy" library is used, which allows executing the cycle at a specific speed or rate (in this case, specified in the "rate" variable).

2.2.7 Viewing the edge of the circle and the centroid

The radius size obtained by the "minEnclosingCircle" function stored in the variable "radius" is taken into account to show the circle's edge and centroid. Suppose this value is greater than 10 pixels (10px). In that case, a circle is created with the help of the "circle" function from the "cv2" library. In the input image (cv_image), the circle coordinates are passed as parameters, the circle radius, the desired color (in BGR format), and the circle border's thickness. The same input image, the centroid obtained by the "moments" function, the circle radius, the desired color, and the thickness are used as parameters to visualize the centroid.

2.2.8 Obtaining negative coordinates

The purpose of this code is to know when there are no obstacles. If the length of the variable "cnts" is less than zero, it indicates that there is no obstacle and, therefore, no contour or coordinates detected. The variables "b" and "a" (centroid coordinates) have negative values to be able to perform the control actions when there is no obstacle.

2.2.9 Image Conversion Error Detection

In converting a ROS image to an OpenCV image (BGR), the “except” statement is used. Then it is published in the terminal to detect possible errors.

2.2.10 Final Image publication

The final image is published using the “imshow” function of the “cv2” library, which receives the window name parameters that show the final image.

2.2.11 Final Image publication converted to ROS format and the x coordinate

Using the “publish” function as well as the “CvBrige” function, the conversion from OpenCV (BGR) format to ROS type format is performed, and both the publication of this image and the x coordinate of the obstacle are performed.

2.2.12 Main function definition

This function executes the "image_converter" class. Furthermore, using the "spin" function of the "rospy" library allows running the node until it is interrupted by keyboard interruption.

2.2.13 Main function initiator

In this conditional, the node called "image_converter" is created, the variable "rate" is also declared as global, and this variable is initialized using the "Rate" function, which provides the speed or rate of data publication. Finally, the "main" function is executed.

3. RESULTS AND DISCUSSIONS

3.1 Obstacle Detection in a Simulated Environment

Figure-3 shows the simulation of the obstacle detection scenario with AR. Drone 2.0, using ROS-Gazebo, which is a tool that allows simulating environments in three dimensions to estimate the behavior of a robot (drone) in a virtual environment. On the left side of the figure, it can see the drone and the obstacle in front of it, while on the right side of the figure, the obstacle's detection is evident. At the bottom, the coordinates x, y of its center.

Figure-3. Object detection in a simulated environment.

3.2 Real-Time Obstacle Detection

For the real implementation, space is sought where the algorithm's tests can be carried out. This place must be open, where the drone can fly without any difficulty and thus demonstrate the correct operation of the designed algorithm. You must be careful with the air

currents of the place because taking into account the drone's size and weight can cause unwanted results. Figure-4 shows the controlled environment in which the algorithm tests are performed.

Figure-4. Controlled environment for real tests.

Nine tests of the algorithm are carried out by varying the parameter of linear speed concerning the x axis (forward speed of the drone) and the shape of the obstacles and their size. Table-1 presents the convention used to distinguish the obstacles used in the tests. Table-2 shows the parameters used for each of the performed tests.

Table-1. Convention for the obstacles used.

Name	Area (cm^2)	Shape
Obstacle A	875	Rectangle
Obstacle B	187	Rectangle
Obstacle C	49.2	Rectangle
Obstacle D	320.47	Circle

Table-2. Parameters for performed tests.

Test	Obstacle	Linear Speed in x (m/s)	Landing Timer(s)
1	A	0.05	17.5
2	B	0.01	17.5
3	B	0.05	17.5
4	C	0.03	17.5
5	C	0.03	17.5
6	C	0.03	17.5
7	D	0.01	17.5
8	D	0.03	17.5
9	D	0.03	17.5

Figure-5 shows the detection of a red rectangular obstacle in real-time, using the AR. Drone 2.0. Two scenarios are observed; the first is the drone with the obstacle located in front of it, and the second one, the ground station obtaining the image from the front camera of the drone with its respective detection.

Figure-5. Detection of an obstacle in a real environment.

The results obtained were good; an obstacle detection system was successfully implemented using open-source tools, as shown below.

3.3 Detection Algorithm Implemented

3.3.1 Creating the class and defining functions

```
class image_converter:
def _init(self):
self.image_pub = rospy.Publisher("image_topic_2",Image,queue_size=10)
self.pub=rospy.Publisher('Coordenadas_Y',Float32,queue_size=1)
self.pub1=rospy.Publisher('Coordenadas_X',Float32,queue_size=1)
self.pub3=rospy.Publisher('Aterrizaje',Float32,queue_size=1)
self.bridge = CvBridge()
self.image_sub = rospy.Subscriber("/ardrone/front/image_raw",Image,self.callback)
```

3.3.2 Establishment of color ranges in HSV format for obstacle detection

```
def callback(self, data):
Lower = np.array([0, 100, 20],np.uint8)
Upper = np.array([5, 255, 255],np.uint8)
Lower1 = np.array([175, 100, 20],np.uint8)
Upper1 = np.array([179, 255, 255],np.uint8)
```

3.3.3 Image type conversion and creation of masks

```
try:
cv_image = self.bridge.imgmsg_to_cv2(data, “bgr8”)
hsv = cv2.cvtColor(cv_image, cv2.COLOR_BGR2HSV)
mask1 = cv2.inRange(hsv, Lower, Upper)
mask2 = cv2.inRange(hsv, Lower1, Upper1)
```

3.3.4 Unification of masks and morphological operations

```
mask = cv2.add(mask1, mask2)
mask = cv2.erode(mask, None, iterations=2)
```

```
mask = cv2.dilate(mask, None, iterations=2)
```

3.3.5 Contour creation

```
cnts = cv2.findContours(mask.copy(),
cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE
)[-2]
```

3.3.6 Obtaining the obstacle coordinates

```
center = None
if len(cnts) > 0:
c = max(cnts, key=cv2.contourArea)
((x, y), radius) = cv2.minEnclosingCircle(c)
M = cv2.moments(c)
center = (int(M["m10"] / M["m00"]), int(M["m01"] /
M["m00"]))
a=(int(M["m01"] / M["m00"]))
b=(int(M["m10"] / M["m00"]))
rospy.loginfo('%i %s %i', int(M["m10"] /
M["m00"]),",",int(M["m01"] / M["m00"]))
rate.sleep()
```

3.3.7 Viewing the circle edge and centroid

```
if radius > 10:
cv2.circle(cv_image, (int(x), int(y)), int(radius),(0, 255,
255), 2)
cv2.circle(cv_image, center, 5, (0, 0, 255), -1)
```

3.3.8 Obtaining negative coordinates

```
else:
a=(int(-1))
b=(int(-2))
```

3.3.9 Image conversion error detection

```
except CvBridgeError as e:
print(e)
```

3.3.10 Publication of final image

```
cv2.imshow("Image window", cv_image)
```

3.3.11 Publication of final image converted to ROS format and the x coordinate

```
try:
self.image_pub.publish(self.bridge.cv2_to_imgmsg(cv_im
age, "bgr8"))
self.pub1.publish(b)
except CvBridgeError as e:
print(e)
```

3.3.12 Definition of the main function

```
def main():
ic = image_converter()
try:
rospy.spin()
except KeyboardInterrupt:
print("Shutting down")
cv2.destroyAllWindows()
```

3.3.13 Main function initiator

```
if _name_ == '_main_':
rospy.init_node('image_converter', anonymous=True)
global rate
rate = rospy.Rate(20)
main()
```

3.4 Comments on the Results

The cv_bridge package's efficiency in terms of obtaining images in real-time is highlighted over other tools such as ffmpeg software and OpenCV's VideoCapture function. Throughout the algorithm construction process, tests were carried out with these three tolls where the obtaining of the images through ffmepg and VideoCapture had a significant delay (approximately 15 seconds). The delay affects the control mechanism of the system, making it inefficient; while using the cv_bridge package, the obtaining of the robot image is done immediately.

OpenCV software for image processing provides many facilities and functionalities, ideal for various cases where image processing is required. However, the obstacles detection efficiency in the presented algorithm depends on the correct establishment of the color ranges to be detected. Setting an extensive range results in the detection of unwanted colors and, by contrast, setting a very narrow range is equivalent to not detecting all possible ranges of the chosen color.

Regardless of the obstacle's shape, the designed algorithm finds the smallest circle covering most of the detected obstacle. This circumference is shown whenever the value of its radius is greater than 10 pixels; however, it is possible to see an obstacle and not be enclosed in the circle, mainly because detection is performed whenever the obstacle contour is more significant than zero. On the other hand, the marking of the obstacle depends on the value of the radius of the circumference.

4. CONCLUSIONS

It is important to consider the Wi-Fi network's coverage (range) created by the drone when it is powered up. When this distance is exceeded, communication between the drone and the station on the ground (computer) is lost, causing the video transmission packets to be lost, and therefore it is not possible to detect the obstacle.

ROS structure offers advantages such as data acquisition in real-time from the used robot, such as obtaining images from the camera, data from sensors, and its large existing community that provides handy information on the web. This feature builds confidence in the development of the project.

The algorithm's performance for obtaining images in real-time by subscribing to the "topic" of the drone camera is highlighted compared to the VideoCapture function. This last function provides the images with a considerable delay time, while the images obtained by ROS are delivered in real-time. However, for the detection to be carried out correctly, it is necessary to define as well as possible the ranges of colors to be detected. Factors such as the brightness of the environment must be considered, which can turn the color of the object to one different.

REFERENCES

Adeva R. 2020. Pensando en comprar un dron? Conoce los tipos de drones según el uso, diseño o control. [Online]. Available in: https://www.adslzone.net/reportajes/drones/tipos-drones/.

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/.

Hernández C., López H., Castellanos L., López E. 2015. Dron polinizador de cultivos. Tecnologías aplicadas para alternativas sustentables. Revista Mexicana de Ciencias Agrícolas. 1: 67-71.

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

Parrot. 2019. Parrot AR. Drone 2.0 elite edition. [Online]. Available in: https://www.parrot.com/es/drones/parrot-ardrone-20-elite-edition.

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.