A Computer Vision Based Algorithm for Obstacle Avoidance (Outdoor Flight)

Wander Mendes Martins, Felix Mora-Camino, Alexandre Carlos Brandao-Ramos

A Computer Vision Based Algorithm for Obstacle Avoidance (Outdoor Flight)

Wander Mendes Martins
Federal University of Itajubá
Computer Science and Technology
Itajubá, Minas Gerais, Brazil
wandermendes@unifei.edu.br

Alexandre Carlos Brandão Ramos
Federal University of Itajubá
Mathematics and Computation's Institute
Itajubá, Minas Gerais, Brazil
ramos@unifei.edu.br

Felix Mora-Camino
Ecole Nationale de l'Aviation Civile-ENAC
Toulouse, France
felix.mora@enac.fr

***Abstract*—This paper presents the implementation of an algorithm based on elementary computer vision techniques that allow an UAV (Unmanned Aerial Vehicle) to identify obstacles (including another UAV) and to avoid them, using only a trivial camera and applying six mathematical treatments on image. We applied this algorithm in a drone in real flight.**

***Keywords*—UAV, Computer Vision, Camera, Embedded Systems, Obstacles Avoidance.**

I. Introduction

In the last 50 years, machine vision has evolved into a mature field embracing a wide range of applications including surveillance, automated inspection, robot assembly, vehicle guidance, traffic monitoring and control, signature verification, biometric measurement, and analysis of remotely sensed images[15]. Computational vision is the study of the extraction of information from an image. More specifically, it is the construction of explicit and clear descriptions of objects in an image[13]. Computational-based navigation of autonomous mobile robots has been the subject of more than three decades of research and it has been intensively studied[1][2]. Processing images in real time is critical for decision making during the flight[3].

Another technology that is gaining great popularity are the Unmanned Aerial Vehicles (UAVs), commonly known as drones. Since the technology to build them today is inexpensive and well understood, they are already being used in many researches and in many applications. Among the proposed applications in the civil area are fire monitoring and extinguishing, inspection of bridges and buildings, crop dusting and even search an rescue of survivors after a disaster. In the military area the applications are surveillance, defense and even air strikes.

There are many ways to make a drone perceive obstacles in their path and deviate from them, such as using monocular or stereo camera and sensors[4] like as sonars, GPS (global position system), previously known routes, etc. In this work we use only computer vision through the treatment of images collected in real time from a trivial embedded camera. The advantage of this solution is to be simple and to have a low cost, proving to be efficient in the tests performed of outdoor flight.

Low-cost solutions to drones are very interesting for use their in large-scale global projects[5] and to popularize their application[6].

Fig. 1. Captured Photo - A drone flying

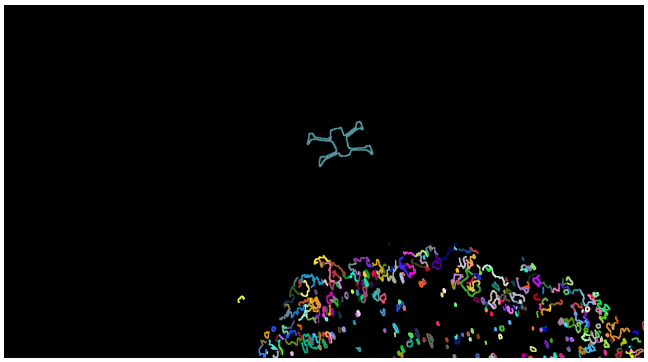

Fig. 2. Edge Detecting

In this work the images are captured in real time during the drone flight, providing a ready-to-run solution.

The platform considered is a quadrotor micro aerial vehicle (MAV) which is similar to a helicopter, but has four rotors[11]. The quadrotor platform is appealing because of its mechanical simplicity, agility, and well understood dynamics[12]. The rest of the paper is structured as follows. In Section II, we present the resources, hardware, software and we detail the technique used. The section III presents some considerations for use of this algorithm in outdoor flights. Section IV shows and discusses the experimental results obtained. Section V presents the conclusion of the work and an evaluation as to its applicability.

Fig. 3. Obstacle Detecting

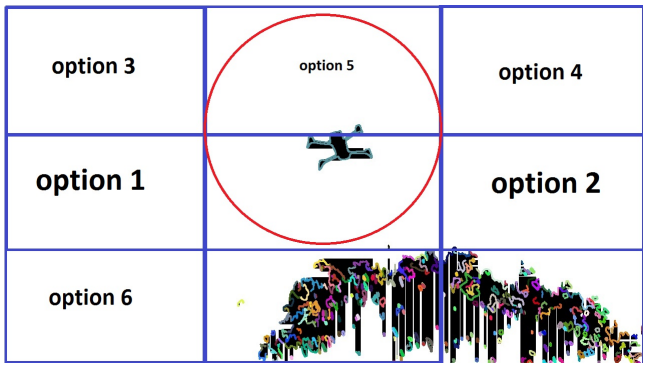

Fig. 4. Free Areas

II. Materials and Methods

We combine hardware and software to acquire, treat and interpret images in real time during flight, and embedded the developed algorithm into a drone.

A. Software

For development we used Linux operating system (OS) Ubuntu Desktop 14.04 LTS[7]. The code was written in C++ language using OpenCV 3.2 open source graphic library[8]. We used ROS Indigo (Robot Operating System) firmware embedded into a Raspiberry PI board on the drone.

B. UAV Platform

The proposed algorithm was intended to be run on a 250mm quadrotor using a Pixhawk[18] as the main controller board. The Pixhawk is an open-source flight controller board responsible for the low-level control of the quadrotor. It is equipped with gyroscope, accelerometers, magnetometer and barometer, and can also be connected to an external GPS module, and has a powerful embedded software that implements the basic control functions. An useful feature of the Pixhawk is the ability to communicate with other devices through a protocol called MAVLink[20], which was developed specifically for UAV applications. This can be used to achieve autonomous control of the UAV, by running the control algorithms in a small portable computer, such as a Raspberry Pi[17], which is carried by the UAV and sends commands to the Pixhawk by MAVLink messages. The UAV is also equipped with a web camera, mounted on its front, to capture the images of the path it is moving into.

C. Technique

The technique consists of (1) image capturing; (2) apply gray scale to the captured image; (3) blur the image; (4) detect edges; (5) find contours; (6) draw contours; (7) identify obstacles and free areas; (8) command the UAV moves to the freer area; and (9) to repeat the process. The figures 1, 2, 3 and 4 show the result of this process. The images shown in these pictures were taken by the embedded camera.

(1) Image Capturing: Capturing frame by frame the image of the front camera (fig 1).

(2) Apply Grey Scale to the captured image: The first mathematical treatment that the primitive image receives is to have reduced its number of colors to a narrower band, limited to tones or degrees or scales of gray, using the toCvShare cvbridge library function, using the ”mono8” parameter.

(3) Blur the image: The ”blur” openCV library function is used to blur the image. This function blurs an image using both the normalized box filter and the following kernel K (fig 5)[27].

$$K = \frac{1}{\texttt{ksize.width*ksize.height}} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 & 1 \\ 1 & 1 & 1 & \cdots & 1 & 1 \\ & & \cdots\cdots\cdots & & & \\ 1 & 1 & 1 & \cdots & 1 & 1 \end{bmatrix}$$

Fig. 5. kernel used by the Blur openCV library function

(4) Detect Edges: The ”Canny” openCV library function is used to apply the Canny Algorithm to detect edges into image. The Canny Edge detector was developed by John F. Canny in 1986. Also known to many as the optimal detector, Canny algorithm aims to satisfy three main criteria: (a)Low error rate: Meaning a good detection of only existent edges. (b)Good localization: The distance between edge pixels detected and real edge pixels have to be minimized. (c) Minimal response: Only one detector response per edge[23].

(5) Find Contours: The ”findContours” openCV library function is used to find object limits into image. The function retrieves contours from the binary image using the algorithm [Suzuki85]. The contours are a useful tool for shape analysis and object detection and recognition[24].

(6) Draw Contours: The ”Scalar” and ”drawContours” opencv functions are used to draw the object’s contours into the image (fig 2). ”Scalar” represents a 4-element vector. The type Scalar is widely used in OpenCV for passing pixel values. We define a BGR color such as: Blue = a, Green = b and Red = c, Scalar(a, b, c)[25]. The ”drawContours” function draws contours outlines or filled contours[26].

(7) Identify obstacles and free areas: In this point, a image has two kinds of basic pixels (dots): the black pixels (RGB 0,0,0) and the non-black pixels. Our algorithm converts the black pixels into white pixels (RGB 255,255,255) and it converts all non-black pixels into black-pixels. After this, the white pixels represents free spaces and the black pixels represents obstacles (fig 3). Dividing the area into nine (or more) subareas, we interpret as the most appropriate way for

the UAV to advance: the area with more white dots. The figure 4 shows option 1 and option2 areas witch more free areas witch need less movements to avoid obstacles.

i)Scanning the image The process of identifying free areas and obstacles consists of scanning the image from the outermost pixels to the inner ones (from the outside to the inside) (algorithm 1). As this is an external flight, we consider the entire free area until the edge contour points of the figure are found (non-white pixels). From there, what is inside the contour is considered an obstacle and what is out of the contour is considered free area where there is some chance of the drone passing through the obstacles.

```
for Each row of the image. (from 0 to maxrow) do
    for Each column of the row. (from 0 to maxcolumn)
     do
        while not found a pixel non-black do
            Turn white the color pixel;
        end
    end
end
for Each row of the image. (from 0 to maxrow) do
    for Each column of the row. (from maxcolumn
     downto 0 ) do
        while not found a pixel non-black do
            Turn white the color pixel;
        end
    end
end
for Each column of the image. (from 0 to maxColumn)
 do
    for Each row of the column. (from 0 to maxrow ) do
        while not found a pixel non-black do
            Turn white the color pixel;
        end
    end
end
for Each column of the image. (from 0 to maxColumn)
 do
    for Each row of the column. (from maxrow downto
     0 ) do
        while not found a pixel non-black do
            Turn white the color pixel;
        end
    end
end
for Each pixel of the image. (row x column) do
    if color pixel is non-white
    Turn black the color pixel;
end
```

Algorithm 1: Scan the contour image

ii) Count the amount of white pixels in each segment: Defined the segments, we will count the number of points "on" in each segment, represented by Sn, where n is the segment and $0 <= n <= 2$.

The pseudo-code for this count is:

```
for each segment n do
    Count the number of pixels on (Sn);
end
```

Algorithm 2: Count pixels on

iii) Suggest that the UAV moves to the segment with the more amount of white pixels (totally 255): We interpret as the most appropriate way for the UAV to advance, the segment S with smaller amount of black dots. Here we interpret black pixel as obstacle and white pixel as free area. In the first step the algorithm divides the image into 3 areas: Left(0), center(1) and right(2).

The decision table is:

IF	AND	Action
S(0)> S(1)	S(0)>S(2)	turn left
S(1)> S(0)	S(1)>S(2)	go center
S(2)> S(0)	S(2)>S(1)	turn right

If two or more segments have the same amount of pixels RGB(255,255,255), we apply the following decision table:

IF	Action
S(n) = S(1)	go center
S(0) = S(1)	turn right

After this step, the area with more white pixels is divided into 3 others vertical areas ,up(0), front(1), down(2), and the process (iii) is repeated.

(8) Command the UAV moves to the freer area: Once it is determined that the UAV has a chance to advance from the some free space, and that there is sufficient amount of free space for this advance, it should be assessed whether the free space is arranged in such a way as to allow the UAV to pass through the segment chosen. Lighted points being scattered may indicate that their sum is greater than the dimensions of the UAV but that their arrangement in space prevents their advance. Commands witch control the UAV movement, as ROLL, PITCH and YAW (fig 6). The move decision is translated neither flying command and transmitted to the UAV that will execute it.

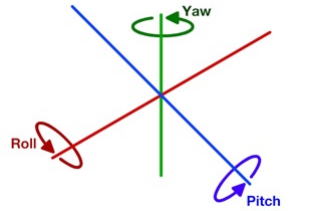

Fig. 6. Quadcopter's orientation

At this point other Artificial Intelligence features come in and decisions that fall outside the scope of our work.

The idea is to offer a quick preliminary decision on the direction the UAV should follow.

9) To repeat the process: Discard of the processed image and its replacement by a new frame, now with the new positioning of the UAV. Once the flight instruction is passed to the UAV, another image is captured at a (t+1) time, and all processing is repeated.

Fig. 7. Diagram generated by the program rqt_graph showing the relevant nodes and topics in the application

D. Implementation

Our algorithm was implemented in C++ and runs on the ROS platform. ROS (Robot Operational System) is an open source technology created to aid researchers in developing robotic applications. ROS provides us with many tools and facilities that were very useful in our work.

A ROS application is a network of nodes that communicate with each other. Each node is an independent program in the system and a Master node also exists to manage the network. Nodes that generate data publish this information in topics in the form of messages, while nodes that need that data subscribe to the corresponding topics and receive the published messages.

In our application we created a node called image_subscriber, which subscribes to the topic where the image messages from the camera are being published and is responsible for processing the images and decide where the UAV should move. In a real implementation the image messages would come from a node connected directly to the camera hardware, but in our implementation Gazebo publishes the data from the simulated camera. Another node, called Mavros, is responsible for communicating with the Pixhawk. When the image_subscriber node decides where to move the UAV, it creates a message of type mavros_msgs/OverrideRCIn, which represents a command from a radio controller, fills it with the corresponding values to cause the desired movement and publishes it to a topic where Mavros is subscribed. Mavros creates a MAVLink message containing that information and sends to the Pixhawk via a serial connection. Figure 7 shows a diagram generated by a ROS tool called rqt_graph showing the nodes and the relevant topics.

III. The Experiment

The experiments were carried out on the campus of the Federal University of Itajub.

IV. Results and Discussion

The tests show that the algorithm is efficient and diverted the drone from all the obstacles it encountered in its path in a outdoor flight. However this work limited itself to diverting the drone and not executing a flight plan. In this context, we can say that the goal of the algorithm was successfully achieved. Getting the drone back on its original route after it veers off an obstacle is not the scope of this algorithm and there are other codes to do this. Also we do not include the calculation of the distance between the obstacle sighted and the drone in flight. Again the reason is the existence of other algorithms to perform this task. The initial focus is for outdoor flight, and we do not implement deviations over or under obstacles. Deviations to the sides, to the center, up and down were contemplated. Although the algorithm was efficient in detecting obstacles and free spaces in indoor images, no tests were performed in this sense.

V. Conclusion

The algorithm was proved efficient in driving the drones through obstacles in outdoor flight without causing collisions. There is a limitation related to the lighting in the environment. It is known that computer vision solutions are very sensitive to changes in lighting, so it would be interesting in future works to test our algorithm in different conditions.

ACKNOWLEDGEMENT

This work was funded by Capes - Coordination of Improvement of Higher Level Personnel[21].

References

[1] A.M.Waxman, J.J LeMoigne and B.Scinvasan, A visual navigation system for autonomous land vehicles, IEEE J.Robotics Auto., vol.RA-3, No.2, pp124-141 (1987)

[2] C. Thrope. M.H. Hebert. T. Kanade and S.A. Shafer, Vision and navigation for the Carnegie-Mellon Navilab, IEEE Trans. Pattem Anal. Mach. Intell., vol.PAM1- 10, No.3. pp.362-373 (1988)

[3] A. Davison, Real-Time Simultaneous Localization and Mapping with a Single Camera, IEEE International Conference on Computer Vision, pp. 1403-1410, 2003.

[4] Sergio Garca; M. Elena Lpez; Rafael Barea; Luis M. Bergasa; Alejandro Gmez; Eduardo J. Molinos,Indoor SLAM for Micro Aerial Vehicles Control Using Monocular Camera and Sensor Fusion, International Conference on Autonomous Robot Systems and Competitions (ICARSC 2016)

[5] John-Thones Amenyo; Daniel Phelps; Olajide Oladipo; Folly Sewovoe-Ekuoe; Sangeeta Jadoonanan; Sandeep Jadoonanan; Tahseen Tabassum; Salim Gnabode; Tanging D Sherpa; Michael Falzone; Abrar Hossain; Aerren Kublal, MedizDroids Project: Ultra-low cost, low-altitude, affordable and sustainable UAV multicopter drones for mosquito vector control in malaria disease management, IEEE Global Humanitarian Technology Conference (GHTC 2014)

[6] M. F. A. Rahman; A. I. C. Ani; S. Z. Yahaya; Z. Hussain; R. Boudville; A. Ahmad,Implementation of quadcopter as a teaching tool to enhance engineering courses, IEEE 8th International Conference on Engineering Education (ICEED 2016)

[7] Ubuntu Desktop 14.04 LT https://www.ubuntu.com, accessed on 13 March 2017

[8] OpenCV 3.2, http://opencv.org, accessed on 13 March 2017

[9] ROS Indigo (Robot Operation System), http://www.ros.org, accessed on 13 March 2017

[10] Bueno, Andr, *Fundamentos da Computao Grfica*(in portuguese), Pontifcia Universidade Catlica, Brasil: Rio de Janeiro, 2011

[11] Ascending Technologies, GmbH, http://www.asctec.de, accessed on 13 March 2017

[12] Koushil Sreenath, Taeyoung Lee, and Vijay Kumar. Geometric Control and Differential Flatness of a Quadrotor UAV with a Cable-Suspended Load. In IEEE Conference on Decision and Control (CDC), pages 22692274, Florence, Italy, December 2013.

[13] Ballard, Dana H. and Brown, Christopher M. Computer Vision. Prentice Hall. ISBN 0131653164, 1982

[14] Bradski, Gary and Kaehler, Adrian. Learning OpenCV: Computer Vision with the OpenCV Library. O'Reilly.

[15] E.Roy Davies, "Machine Vision: Theory, Algorithms, Practicalities", 2005

[16] R. C. Gonzales and E. R. Woods, Processamento de Imagens Digitais (in portuguese). So Paulo, Brazil, 2000.

[17] Raspberry pi, https://www.raspberrypi.org, accessed on 13 March 2017

[18] Pixhawk, https://pixhawk.org/, accessed on 13 March 2017

[19] L.F.Lima, https://lazarolima.wordpress.com/2010/08/19/processando-imagens-em-grayscale-e-negativo-em-c/(in portuguese),2010, accessed on 13 March 2017

[20] MavLink, http://www.mavlink.org/, accessed on 13 March 2017

[21] CAPES, http://www.capes.gov.br, accessed on 13 March 2017

[22] The Bored Engineers (https://theboredengineers.com/), "The quadcopter : control the orientation", https://theboredengineers.com/2012/05/30/the-quadcopter-basics/, accessed on 11 Abril 2017

[23] OpenCV 2.4.13.2 documentation (imgproc module). Image Processing, "Canny Edge Detector", http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html?highlight=canny, accessed on 27 April 2017

[24] OpenCV 2.4.13.2 documentation, "Structural Analysis and Shape Descriptors", http://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=findcontourscv2.findContours, accessed on 27 April 2017

[25] OpenCV 2.4.13.2 documentation, "Basic Drawing", http://docs.opencv.org/2.4/doc/tutorials/core/basic_geometric_drawing/basic_geometric_drawing.html?highlight=scalar, accessed on 27 April 2017

[26] OpenCV 2.4.13.2 documentation, "drawContours", http://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=drawcontoursdrawcontours, accessed on 27 April 2017

[27] OpenCV 2.4.13.2 documentation, "Blur", http://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=blurcv2.blur, accessed on 27 April 2017