Collision Avoidance Algorithms For Unmanned Aerial Vehicles Using Computer Vision

Technical Report # CSSE17-02

Hani Sedaghat-Pisheh
University of California, Berkeley
spisheh@berkeley.edu

Amaury Rodríguez Rivera
University of Puerto Rico, Arecibo
amaury.rodriguez2@upr.edu

Dr. Saad Biaz
Auburn University
biazsaa@auburn.edu

Dr. Richard Chapman
Auburn University
chapmro@auburn.edu

Abstract—This papers present our approach to use computer vision to enable Unmanned Aerial Vehicles (UAVs) to avoid collisions. In order to detect a UAV, a machine learning algorithm, called Cascade Classifier, is used. Then, to track it, the Camshift algorithm is implemented. These algorithms determine the coordinates of center of an object moving towards the UAV. Finally, in order to determine the distance of the object to the UAV, a stereo camera is used. Once an object is successfully tracked, the UAV will execute avoidance maneuvers in case the path of the object conflicts with the UAV's.

Keywords—Unmanned Aerial Vehicles; UAVs; Collision Avoidance; Computer Vision; Machine Learning; Meanshift; Camshift; Cascade Classifier; Haar Cascade; Sense and Avoid; Stereopsis; Depth Perception; Stereo Camera.

I. Introduction

There is a growing trend of Unmanned Aerial Vehicles (UAVs) replacing human pilots. These types of aircrafts can decrease the number of casualties. For instance, in a military application, UAVs can travel into hostile areas without risking the pilot's life. For civilian applications, UAVs can accomplish search and rescue tasks without exposing a human pilot. Moreover, they can be less expensive than manned aircraft, making them a cost effective choice [5]. However, a UAV without collision avoidance capabilities can cause property damage, injuries, or even death. But even though unmanned aircrafts are excellent solutions for military and civilian tasks, they require specialized and expensive equipment to "sense" their surroundings and detect conflicting objects. This paper proposes the use of computer vision to sense the surroundings because cameras cost less than specialized equipment.

II. Problem Description

Unmanned Aerial Vehicles require an infrastructure to avoid collisions by other UAVs. That infrastructure, such as ADS-B, requires that each aircraft determines its current location and broadcast its around. In order to avoid such an infrastructure, this works

proposes to implement computer vision algorithms to provide UAVs with human-like "*sense and avoid"* capabilities. For humans, identifying and analyzing objects visually seems easy, but for computers it is a complex task. Therefore, providing UAVs the capability to see and avoid obstacles can be arduous. This work uses the open source *Open Computer Vision* library (**OpenCV**).

III. Literature Review

To provide detection capabilities for a UAV, object tracking algorithms must be implemented. There are already numerous algorithms that have been tested and used for object tracking. All these algorithms are included in the open source library *OpenCV*. For this work, these tracking algorithms were considered and evaluated: *Shi-Tomasi, Lucas-Kanade, Blob Detection, Meanshift, and CamShift*.

A. Shi-Tomasi

The *Shi-Tomasi* method uses feature detection to find objects. It finds the strongest corners in an image, but it requires the number of corners, the quality of the corners, and the minimum distance between them. [1]. *Shi-Tomasi* can detect the corners of multiple objects with different shapes (see Figure 1). In the *OpenCV* library, the Shi-Tomasi method can be invoked through the `goodFeaturesToTrack()` function. This method was eliminated because it is processing intensive and it requires the number of.

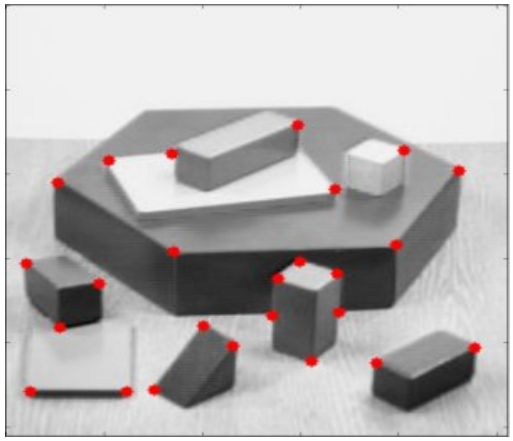

Figure 1 *Shi-Tomasi detecting corners [10]*

B. Lucas-Kanade

Lucas-Kanade uses optical flow for tracking objects by analyzing an object's feature found on the current frame and in consecutive frames[1]. It can be invoked in the *OpenCV* library through the `calcOpticalFlowPyrLK()` function. Figure 2 shows how *Lucas-Kanade* tracks a ball moving through five consecutive frames. This method, like Shi-Tomasi, also proved to be processing intensive, thus this method was eliminated.

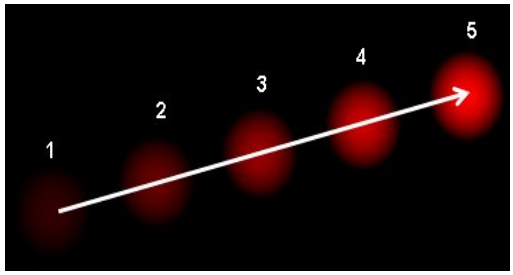

Figure 2 Lucas-Kanade ball tracking

C. Blob Detection

A *blob* is a group of connected pixels in an image that share some common property[8]. *Blob detection* senses these connected pixels. *OpenCV* provides methods to detect and filter blobs based on different characteristics (see Figure 3) [8]. The blob detection method is not accurate for finding specific objects and it works best with 2 dimensional solid color geometric shapes, therefore this method was also eliminated.

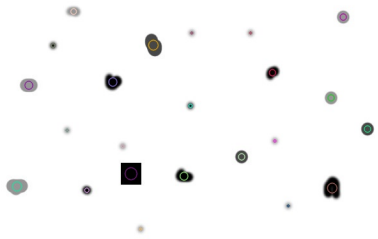

Figure 3 Blob detection

D. Meanshift

Meanshift is a color-based object tracking algorithm. It uses a color histogram and gradient density to find the maximum density of pixels in an image. Figure 4 shows how Meanshift moves the search window (Circle C1) to where it found the maximum density of pixels (Circle C2). Although excellent, *Meanshift* still has problems when an object approaching the camera. For this reason, *Meanshift* was also eliminated for tracking UAVs. Thus, an enhanced version of *Meanshift* -- dubbed *Camshift*, that is described below.

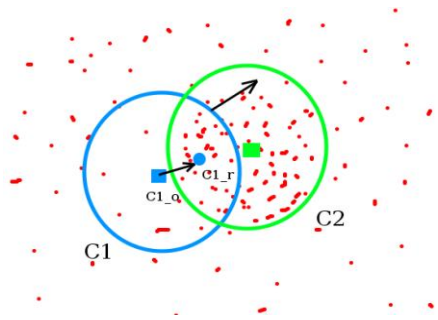

Figure 4 Meanshift (maximal density)

E. Camshift

Camshift stands for *Continuously Adaptive Meanshift*. The *Camshift* algorithm works exactly as *Meanshift*, but before each *Meanshift* iteration it resizes the search window [7] . This makes *Camshift* much more efficient and practical than *Meanshift*. For this reason, we implemented *Camshift* to track UAVs. This algorithm was used because of its efficiency on tracking approaching objects, which is critical for tracking approaching UAVs and detect collision threats. Figure 5 shows how Camshift makes the tracking rectangle larger as the car approaches the camera.

Figure 5 Camshift tracking an approaching car [7]

Since the distance between UAVs is needed, we overhaul a method of finding the depth and distance of an object by providing the ability of stereopsis using two cameras. *Stereopsis* is the depth perception that humans and animals possess by having binocular vision. This method is called **stereoscopic vision**. *Stereoscopic vision* creates an illusion of depth using two pictures taken at slightly different positions. Figure 6 shows how a picture of a tree is taken with two cameras, this represents a human-like stereopsis. The principles of detecting distance between stereo cameras and an object is finding the angles that object makes with the normal line to the center of each camera's lense (φ_1 and φ_2 in Fig 6). In order to find φ_1 and φ_2 a geometric similarity and proportionality is used (see Figure 7) [3].

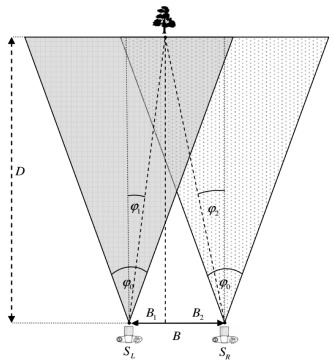

Figure 6 Stereoscopic vision [3]

x_1 and x_2 are the horizontal distances (in pixels unit) between the object and center of image of each cameras. Also, φ_1 and φ_2 are respectively the angles that object makes with the first and second camera. Moreover, φ_0 is the viewing angle of the camera and x_0 is the total number of horizontal pixels of the image. Finally, having x_1, x_2, and B, which is the distance between two cameras, it is possible to calculate the distance *D* between stereo camera and object by using the formula $D = B/(\tan\varphi_1 + \tan\varphi_2)$ [3].

$$\frac{x_1}{\frac{x_0}{2}} = \frac{\tan\varphi_1}{\tan\left(\frac{\varphi_0}{2}\right)},$$

$$\frac{-x_2}{\frac{x_0}{2}} = \frac{\tan\varphi_2}{\tan\left(\frac{\varphi_0}{2}\right)}.$$

Figure 7 Finding x1 and x2

IV. Approach

Before executing any avoidance maneuver, collision threats must be detected. This work uses the *Cascade classifier* to detect other UAVs, the *Camshift* algorithm to track them, and *stereoscopic vision* to find the distance separating from the obstacle. Two AR.drone 2.0 are used to test this approach: one as a self-flying drone with stereo camera and a Raspberry Pi 2 Model B attached to it and the second drone as a threat. Several tests were conducted with these two drones. All tests were successful and the self-flying drone took off and landed as expected every time. Below are the different approaches on detecting and tracking UAVs with computer vision algorithms using the *OpenCV* library.

A. Color Detection

The *color detection* method finds and tracks the objects based on their color. For this method, the object's color must be known in advance. In addition, the minimum and the maximum values of Hue, Saturation, and Value (HSV) must also be known. In other words, the program just focuses on detecting a specific color. In this case, the thresholded image shows the intended color as white and every other color as black (see Figure 8). The *color detection* method is faster than all methods described so far. However, this method cannot be used when the background's color is the same as the UAV's.

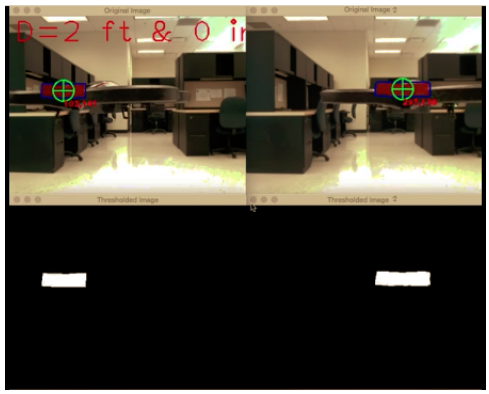

Figure 8 Tracking with color detection

B. Motion Detection

Another approach in this work was a motion detection method in order to sense moving objects. There are several motion detection algorithms in the *OpenCV* library. The method implemented in this project was by comparing each frame with the previous one and detect if any change has occurred in them using the `absdiff()` function. An advantage of motion detection, compared with color detection, is that it does not require to know in advance the color of the object and the background's color does not affect the results. The problem of the motion detection algorithm in this project was having a moving camera on a UAV. Since the camera is moving, it identifies every object as a moving object, so it cannot detect moving objects accurately. This algorithm can effectively detect a UAV in movement with a stationary camera (see Figure 9).

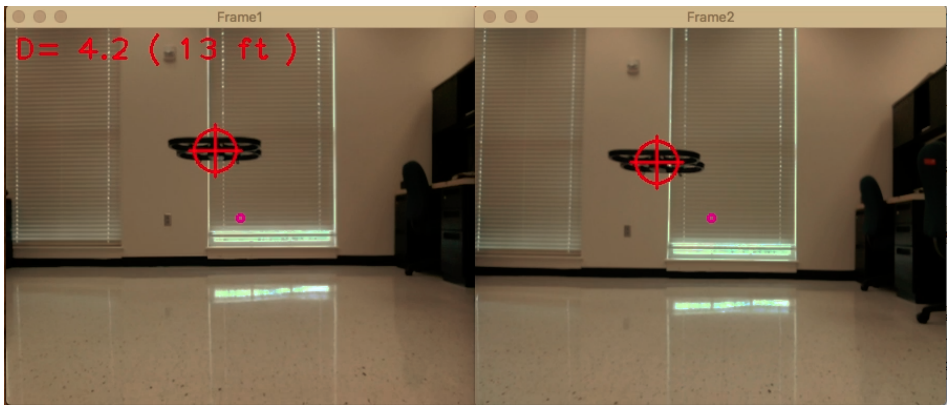

Figure 9 AR Drone detection with motion detection

C. Cascade Classifier

Haar cascade is a machine learning algorithm for object detection. A cascade is a trained function using a large amount of positive and negative pictures [6]. Positive pictures are those containing the desired object to track. On the contrary, negative pictures are those without the desired object to track. We implemented a *Haar* cascade training using 1600 positive pictures containing an AR Drone 2.0 and 800 negative pictures without it. Once trained, an XML file is used for finally detecting the AR Drone 2.0 (see Figure 10).

Figure 10 Haar cascade

D. Camshift for Tracking

Since detecting and tracking objects using *Cascade Classifier* requires many resources, we only used *Cascade Classifier* for detecting the UAV. Then the *Camshift* algorithm was implemented to track the UAV after it has been detected. For the *Camshift* algorithm, we need to have the initial search window which, has been already provided by the *Cascade Classifier*. Figure 11 shows how an AR Drone 2.0 is detected with the *Cascade Classifier* and tracked with the *Camshift* algorithm simultaneously.

Figure 11 Detected by Haar and tracked by Camshift

E. Stereo Camera

In this project, the UAVs were tracked using stereo camera, which provides very useful information about the distance of those UAVs. This information was used to define a threat zone. The threat zone is defined when the UAV detects another UAV within less than 5 feets. As a UAV enters the threat zone, the proper reactions are executed. In order to accomplish this, we used an AR Drone 2.0 equipped with a Raspberry Pi 2 Model B, two Logitech c615 cameras and a 3D printed frame for placing the two cameras (see Figure 12). Apart from the *OpenCV* library, we implemented the maneuvers of the AR Drone 2.0 with a Python library called *python-ardrone* [9].

Figure 12 Equipped with Raspberry Pi, two camears, and 3D printed frame

V. Conclusion

After testing different approaches, the best way of detecting a UAV in order to avoid a collision using *OpenCV* is to use *a Cascade Classifier*, since this method is accurate and reliable. It does not require to sense a moving object like the motion detection does. Furthermore, a suitable background color like (as in the color detection algorithm) is not necessary. The *Camshift* algorithm successfully tracks already detected UAVs when they approach the stereo cameras. These cameras can accurately calculate the distance of a near detected UAV, sending a collision thread when it's 5 feets from it. Finally, the avoidance maneuvers are successfully executed using the *python-ardrone* library. Thus, computer vision is an effective method for UAV collision avoidance. A future work should enhance the avoidance maneuvers and test the algorithm on fixed-wing UAVs.

References

[1] Richards B., Gan M., Dayton J., Enriquez M., Liu J., Quintanna J., and Bhandari S. Obstacle Avoidance System for UAVs using Computer Vision.
http://arc.aiaa.org/doi/pdf/10.2514/6.2015-0986

[2] Carnie R. J., Walker R. A., Corke P. I. Computer-Vision Based Collision Avoidance for UAVS.
http://eprints.qut.edu.au/4627/1/4627.pdf

[3] Mrovlje J., and Vrančić D. Distance measuring based on stereoscopic pictures.
http://dsc.ijs.si/files/papers/S101%20Mrovlje.pdf

[4] Parekh K. B. A Computer Vision Application to Accurately Estimate Object Distance.
http://digitalcommons.macalester.edu/cgi/viewcontent.cgi?article=1018&context=mathcs_honors

[5] Boyle A. The US and its UAVs: A Cost Benefit Analysis, American Security Project. 2012. Retrieved June 12, 2016.
http://www.americansecurityproject.org/the-us-and-its-uavs-a-cost-benefit-analysis/

[6] OpenCV Open Source Computer Vision. OpenCV: Face Detection using Haar Cascades. 2016. Retrieved June 15, 2016.
http://docs.opencv.org/master/d7/d8b/tutorial_py_face_detection.html#gsc.tab=0

[7] OpenCV Open Source Computer Vision. OpenCV: Meanshift and Camshift. 2016. Retrieved June 10, 2016.
http://docs.opencv.org/3.1.0/db/df8/tutorial_py_meanshift.html

[8] Mallick S. Blob Detection Using OpenCV (Python, C++). 2015. Retrieved June 10, 2016.
https://www.learnopencv.com/blob-detection-using-opencv-python-c/

[9] Venthur B. python-ardrone library. 2015. Retrieved July 6, 2016.
https://github.com/venthur/python-ardrone

[10] OpenCV Dev Team.. Shi-Tomasi Corner Detector & Good Features to Track. 2014. Retrieved June 5, 2016.
http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_shi_tomasi/py_shi_tomasi.html

[11] OpenCV Open Source Computer Vision. OpenCV: Optical Flow. 2016. Retrieved June 5, 2016.
http://docs.opencv.org/master/d7/d8b/tutorial_py_lucas_kanade.html#gsc.tab=0

[12] Bradski G., and Kaehler A. Learning OpenCV Computer Vision with the OpenCV Library. O'Reilly Media, Inc, Sebastopol, 2008.