MAV 2023

H. Jekel
TU Delft - Aerospace Engineering

1 Introduction

Our project aimed to create an autonomous obstacle avoidance system for a drone, with the ultimate goal of covering the largest distance in a 10-minute flying contest. To accomplish this, we utilized two different approaches: optical flow and color filtering. The optical flow approach was primarily focused on identifying obstacles and computing the distance between the drone's camera and the obstacle, as detailed in section 2. During circular flight, we observed that obstacles in the drone's flight path exhibited a constant optical flow magnitude, which allowed us to detect them as explained in section 3. Furthermore, this approach enabled the drone to evade obstacles by merely adjusting its flight radius, obviating the need for deceleration and directional changes. Nevertheless, we chose the color filtering approach for the competition, as it allowed the camera to identify orange, green, white, and black objects, as described in section 4.

2 Birds-eye view map construction and obstacle avoidance

In this section, we present the first optical flow approach we examined, which involves constructing a 2D horizontal obstacle map using depth estimation from optical flow. The resulting map can be used for path planning. We provide details on the visual concept, control concept, and implementation of this approach below.

2.1 Vision concept

Using the horizontal optical flow equation and rearranging it for the depth z, the following can be found:

$$z = \frac{V_x - u \cdot V_z - V_u V_z \Delta t}{V_u} \tag{1}$$

Here, V_x and V_z are the horizontal velocity components of the drone in its body frame (left to right and up to down respectably), u is the horizontal image projection of the marker w.r.t. the center of the image, V_u is the horizontal optical flow and Δt is the time difference between the frames.
Using a vertical edge detector on the image one can find markers to be tracked by the optical flow to calculate the depth of the associated point. After accounting for the curvature of the lens and calculating their depth, the exact position of the point in 3d can be calculated in the camera frame. Only the flow in the x-direction would be considered, allowing these measurements to be projected on a 2D obstacle map. This approach, as any optical flow approach, had the pitfall of not being able to detect featureless surfaces, and we came up with an additional remedy for that. If there was a big area without flow detections, the color in that area would be checked. If it was black, like the featureless background, it would be left alone. However, if it was e.g. the whiteboard, we would interpolated the closest depth measurements to its left and right over that whole area.

2.2 Control concept

With the 2d obstacle map constructed, navigation can be carried out. Three approaches where considered: One was checking, if the transversal obstacle clearance for continuing straight is sufficient, otherwise turning in the direction of best clearance, and the second was calculating the direction with the best transversal obstacle clearance to be followed. The third approach was a bit more complex, but would likely be the best one given more time. Here, measurements would be assigned to radially dispersed grid points, which were linked together to allow the drone to fly in circles by following the connections between these points. Then, the heuristic, which also represents the permissible speed, is updated for these grid points based on the obstacle detections and finally, an A*-like algorithm could find the best path forward.

2.3 Implementation

Many parts of this approach were constructed in python. The image undistortion, derotation and cropping was implemented in python as well as C++. Finding flow in the image was implemented using a ClaHE equalization followed by a vertical Sobel filter. The pixel values after the Sobel filter were then used to select and describe features to track. Work on the third control concept also started in python, with the grid created and connected in a meaningful way. However, as the competition was approaching, the team realised there was just too much to finish with this approach and we had to move on to something simpler. That simpler approach was the color avoider described in section 4 but as we were encouraged by the professors to give optical flow one more chance, we devised one more simpler optical flow approach as well, presented in section 3.

3 Direct flow classification in circular flight

We brainstormed ideas for something that could be implemented quickly and we came up with an approach that used circular flight to be able to classify obstacles from a single flow detection.

3.1 Vision concept

Obstacle detection is possible during straight flight, however, the obstacle directly in front of the drone, on a colli-

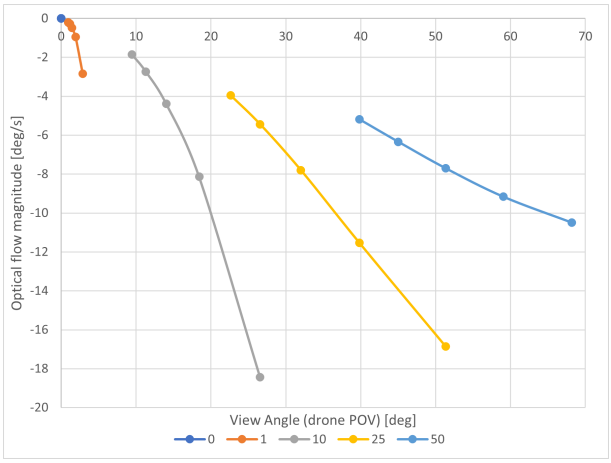

Figure 1: Optical flows of obstacles in straight flight

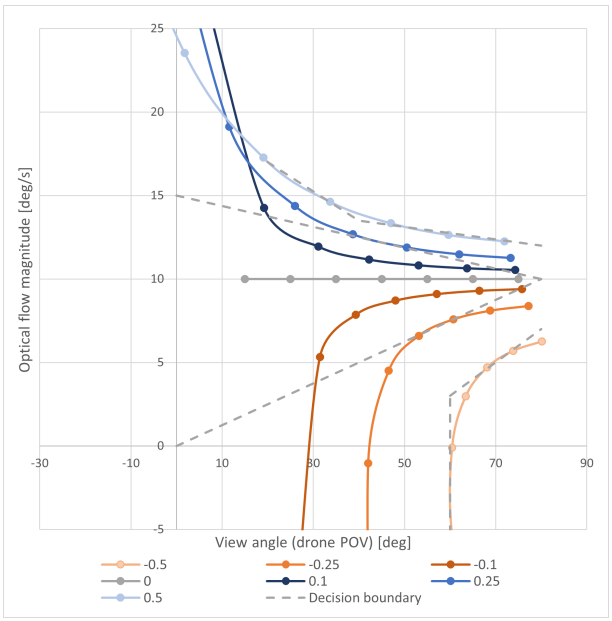

Figure 2: Optical flows of obstacles in circular flight

sion course, has no flow. Figure 1 plots the flow magnitudes for obstacles at increasing static distances from the centerline during a steady forward flight, illustrating this point. It can also be seen that there exists a 1-to-1 transform from the viewing angle and flow magnitude to distances in the x and y direction in straight flight, as long as the speed and shutter speed are known and obstacles are stationary.

In circular flight, obstacles on a collision trajectory have some constant optical flow on their way through the camera view, making it possible to detect the most dangerous obstacles on a direct collision course. Additionally, circular motion induces more optical flow, which means that noise has a lower impact on the accuracy of these readings. We plotted a graph of the movement of obstacles in the visual field of the drone during steady circular flight, with obstacles on one line perpendicular to the circle at various higher and lower radii. Figure 2 displays the result of this experiment, which shows that although the transform from this space to the position of the obstacles relative to the drone is no longer 1-to-1, it is still almost 1-to-1, with just a small region where the lines cross over, allowing us to classify most obstacles from a single flow vector detection.

3.2 Control concept

Compared to straight flight, there are several advantages to circular motion. Firstly, moving on a circular path means that the drone will never hit the boundaries of the arena, with constraints on the maximum radius. Secondly, the only thing to control is the radius and speed based on the obstacle detection. The exact logic will be explained in the implementation section but basically it follow: ”if there is an obstacle on your collision path, check the next upper and lower radius, adjust to a free radius” in perpetuity. Finally, moving through the arena on a circular path allows for a significantly higher average speed than moving along straight lines in an ideal scenario. Additionally, a controller that allows the drone to fly in circular flight in guided mode had already been developed, which also contributed to the attractiveness of this option.

3.3 Implementation

To implement this approach, several files have been adjusted and pieced together, all of which can be seen in the *Optical_flow_circle* branch. Starting from the camera, the existing *opticflow_calculator* and *opticflow_module* files were used. Obstacle detections were checked with linear boundaries shown in dashed grey in Figure 2. The pixel values corresponding to the angle values were calibrated. The number of detections was also capped and would evaporate with new frames to deal with noise. Detections were made on the collision path and one lower and higher radius each. The optical flow ABI message was modified to send the numbers of these detections, received by the modified *orange_avoider_guided* module, which controlled the radius and speed of the circular flight. All of this is put together in the BUU configuration.

3.4 Shortcomings

Almost all of the parts of the system were there and connected, however, it was not enough to fly successfully in time. There is only one full part that was missing and that was image undistortion. This would skew the optical flow magnitudes, mainly in the edges of the view, but it was assumed that we could still work around this since the readings around the center of the image should still be accurate. Other than that, the drone missed testing, debugging, tuning, and verification, to better set the detection classifiers, flight speeds and other parameters. Of course, there would still be shortcomings to this detector even if it was finished, mainly that it would not be able to avoid areas with no optic flow, it would not be able

to detect obstacles while changing radii, and it would not be able to avoid an obstacle if it spanned through all of its available radii. All of these shortcomings could be mitigated by additional classifiers or control rules.

4 COLOUR AVOIDER

As previously mentioned, our team chose to employ a color recognition approach for obstacle detection in anticipation of possible delays in completing the optical flow method before the competition. The color recognition algorithm involved two main steps: determining the number of pixels in the camera footage that have a color that falls within the color filter range and comparing it to a predefined threshold to determine if the drone was too close to an obstacle.

4.1 Colour filtering and finding thresholds

Although YUV images are not easiest to interpret, we opted to directly analyze colors in the YUV format for greater efficiency. To identify the proper lower and upper bounds for each color in YUV format, we utilized the "Learning color classifier" notebook provided on Brightspace, which employs a machine learning-based decision tree approach to identify the conditions in which pixels of the desired color are found. The first thing we had to do for this algorithm was to correctly label out test data. Of which we had approximately 2000 images gathered from test footage in the CyberZoo, featuring all possible obstacles. We wrote a Python script which would iterate through photos and create masks for each of them using HSV color filtering. (Figure 3). Then, once the labeling was done, we only had to feed the test data to the algorithm which would train itself to recognize the chosen color and then give the final values in the form of a logical decision tree. And this decision tree helps define 2 arrays which correspond to lower and upper bounds to determine a color in YUV format.

In the subsequent step, we developed a Python code to determine an initial approximation of the pixel count threshold by iterating through ten hand-picked images. We then used this threshold to analyze all test footage and adjust the values of the arrays and the threshold to minimize false positives while ensuring accurate detection of true positives. We also conducted a test flight in the CyberZoo a few days before the competition, but our drone did not take off due to technical issues with Paparazzi. The obtained color filters are presented in table 1.

Table 1: YUV Color Filters and Thresholds

Color	Lower Bound	Upper Bound	Threshold
Green	(100, 0, 0)	(114, 126, 155)	3000
Black	(40, 0, 0)	(114, 126, 165)	20000
White	(213, 0, 0)	(255, 255, 255)	45000

4.2 Paparazzi implementation of Color Filtering

After discussing the color filtering approach in theory, the next step was to implement it in Paparazzi. The relevant code

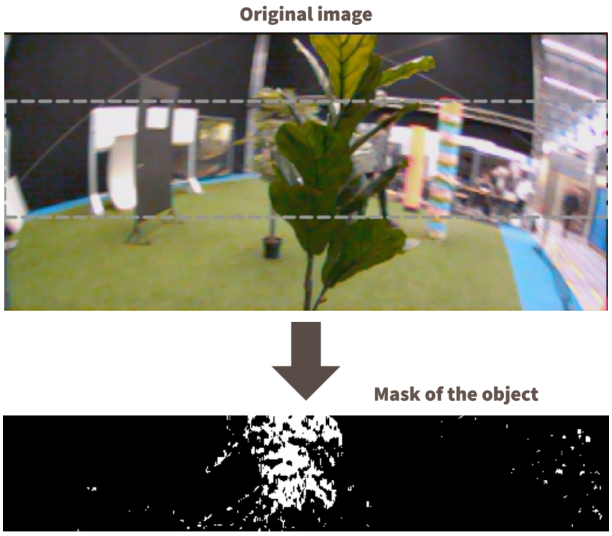

Figure 3: Example of a mask obtained from an image in the dataset

Figure 4: Example of settings to change the thresholds and colour values on the spot.

can be found in the orange branch. The color filtering approach was built on top of the original orange avoider, so the two have a very similar structure. During the addition of the extra functionality, several files were modified as described below.

4.2.1 bebop_course_orangeavoid.xml

In this file, the extra color filters were initialized. The actual values were then optimized using the Python script described earlier to obtain two arrays corresponding to the lower and upper bounds for the desired color in YUV format.

4.2.2 cv_detect_color_object.xml

To avoid hardcoding all the color and threshold values in the code, the settings were expanded to include sliders for both the color values in YUV and the threshold. This allowed us to walk around the cyberzoo and fine-tune the values during testing. Figure 4 illustrates how these sliders and inputs function.

4.2.3 cv_detect_color_object.c

The color detection code underwent modifications to accommodate the additional filters, both in the header and C code. Of particular note is the new implementation, which involves checking each filter individually for a pixel count threshold, rather than simply summing all pixels that fall within the filter's range. This modification allows the AbI to send a message to the orange avoider code for evasive action only when a threshold has been exceeded for a specific color filter, indicating that an obstacle of that particular color is in close proximity. As such, the drone will not take action if multiple obstacles of different colors are detected in the distance, such as orange cones or the audience, thereby minimizing unnecessary evasive maneuvers.
An additional modification made in this part is that only the central section of the camera feed is considered, which is situated approximately at the drone's altitude and is crucial for obstacle avoidance. This approach also ensures that the ground plane does not interfere with the detection of obstacles, such as trees, as the ground is generally green. The camera feed in VLC is presented in Figure 5, where the original top and bottom sections of the image are accompanied by the middle section with the color white overlaid on the original image, indicating pixels whose color falls within the range of the detected colors, as previously described. The implementation of the color avoider method resulted in a flight distance of 67 meter in 10 minutes.

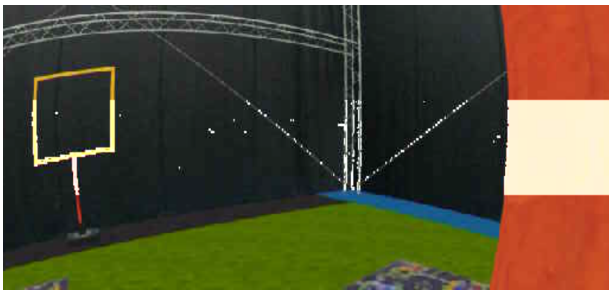

Figure 5: Example of only the middle horizontal being part of the colour avoider algorithm.

4.3 Reflection on colour avoider method

The use of the color avoider method, as previously described, is subject to limitations that must be taken into account. Specifically, the absence of depth information presents a challenge in avoiding obstacles, and the color values of objects can be affected by lighting and orientation, resulting in a wide range of colors that must be monitored, potentially interfering with other objects. Additionally, as the number of distinct objects increases, the challenges associated with monitoring numerous color values grow more significant.
Moreover, the lack of a "memory" to track how quickly the pixel count changes for a given filter is regrettable. This capability would enable the program to initiate the turn command only if a significant number of pixels of the same filter are detected for several frames in a row, indicating that the observation is not an anomaly. Implementing this feature would reduce instances in which the drone is unable to navigate out of a difficult situation.
In summary, although the color avoider method is useful, alternative approaches should be investigated to develop a more dependable object avoidance system.

5 Conclusion

In conclusion, the objective of the project was to create an obstacle avoidance system for a drone, with the ultimate goal of competing in a 10-minute flying competition. Two distinct approaches were implemented, optical flow and color filtering. Optical flow was used to detect obstacles and compute the distance between the drone's camera and the obstacle, while color filtering enabled the camera to detect specific colors of objects. The team found that the optical flow approach was more effective in avoiding obstacles by adjusting the flight radius, while color filtering was preferred for the competition. Towards the end of the project, a new approach using circular flight was developed to classify obstacles from a single flow vector detection. The circular flight approach had several advantages over straight flight, such as never hitting the boundaries of the arena. Overall, the project was successful in creating an obstacle avoidance system for the drone, which could be used in future competitions, as we obtained a distance of 67 meters in 10 minutes.