Autonomous Flight of Micro Air Vehicles 2015 - Group 6

M. Bevernaegie,[*] C. Cheung,[†] Y.I. Jenie,[‡] S.H. Lee,[§] P. Lu[¶]
Delft University of Technology, Delft, The Netherlands

Abstract

This should be done at the end - by Cherry

1 Introduction

by Cherry

1. Overall introduction of the project
2. Equipment - Opti-track, AR.Drone 2.0
3. Software - Paparazzi

2 Vision algorithm

2.1 Vision-based Navigation Literature

Here we talk about the vision algorithm used in the literature

2.1.1 Optic Flow

2.1.2 Computer Vision

2.1.3 Mapping

SLAM

2.2 Own Vision Algorithm

In Section 2.1, a number of the vision-based navigation methods used in literatures were briefly introduced and discussed. Although these methods have shown considerably good results in certain work domains, our team decided to come up with more simple method for the obstacle detection and avoidance. The rationale behind this is as follows:

- Lucas-Kanade optic flow generation from Harris corner detector or FAST feature detector will not work well unless the obstacles have many "features" on their surface. In the competition, monotonic orange poles and a black wall will be used, and hence we anticipate that there will not be many features available on the texture of the obstacles.

- For the same reason, it will be very difficult to compute the Time-to-Contact because it primarily uses the optical flow vectors.

- Object recognition or image classification method can be difficult for tuning and other practical reasons. For example, in order to avoid the orange poles, one cannot simply judge from the overall color of the image as to whether the obstacle is close enough or not. This is because there are cases where multiple poles are visible in the image, although none of them are close enough to the drone. Also, the poles appearing from the "blind spot" of the camera's view are more likely to be hit than the poles located straight ahead (These are verified during the tests). In comparison to other simpler methods, the object recognition will not allow for a flexible adaption to account for all these various situations.

- As the drone has only one front camera, the mapping techniques, such as SLAM, will take considerably more computational time and memory. This technique does not go along well with the objective of the competition which is to enable the drone travel as much as possible while minimizing the collisions.

For these reasons, it was necessary to come up with much simpler vision algorithm which can be implemented and tested easily. The proposed vision algorithm (Figure 1) uses a simple color thresholding method for the detection of specified obstacles, which is done in the following steps:

1. Take only the bottommost row of the pixels in the image as input. Note that the floor will be visible until the distance between the drone and an obstacle standing on the floor becomes lower than a certain threshold distance, assuming that the drone moves slowly in a hovering flight at a fixed altitude. For the visualization, refer to Section 4.1.

2. First of all, the drone should avoid the black wall. By setting a intensity (i.e. Y color channel) threshold, convert the bottommost pixel row into a binary row which has the elements of value 1 and 0 (i.e. 1 if the pixel intensity value is lower than the threshold, and 0 other wise). Therefore, if the image of the black wall is captured at the bottommost row, it will be indicated by the continuous sequence of 1's.

3. Count the number of continuous 1's, and if the sequence is longer than a certain threshold, namely the "black wall width-threshold", then mark the position as the position of a black wall. For each mark, give the weight based on the detected width. Determine either

[*] M.Bevernaegie@student.tudelft.nl
[†] C.Cheung@student.tudelft.nl
[‡] Y.I.Jenie-1@tudelft.nl
[§] S.H.Lee-2@student.tudelft.nl
[¶] P.Lu-1@tudelft.nl

to turn left or right, depending on the weighted position of the black wall.

4. If there is no black wall detected (i.e. no continuous sequence of 1's exceeding the "black wall width threshold"), evaluate the Cr chroma value to detect the orange poles. Generate a binary row in the similar manner as for the black wall detection by setting an appropriate Cr threshold.

5. In order to react more quickly and sensitively towards the poles appearing from the sideways, evaluate the left- and right-end of the Cr binary row to see if there is any shorter continuous sequence of 1's appearing from the sides. This necessitates a different type of width-threshold on the sides, namely the "orange pole sideway threshold". If there is no detection of an orange pole on the sides, continue to the next step.

6. Using the analogous procedure as for the black wall detection, determine either to turn left or right, depending on the weighted position of the orange pole which can be computed using another width-threshold, namely the "orange pole width-threshold".

7. If there is still no detection of the continuous sequence of 1 which exceeds any of the above-mentioned width-thresholds, then yield a command to fly forward.

The proposed vision algorithm has the following advantages: first of all, there are many tuning parameters (e.g. color-thresholds and width-thresholds). This means that it is possible to adjust the system by trial and error in a flexible manner. For example, the color-thresholds can be easily varied and/or combined to detect specific colors of the obstacle, or the width-thresholds can be changed to adjust the stopping distance from the detected obstacle. Also, it is possible to test the vision algorithm with static images. This allows the tuning easier as the drone does not necessarily have to move around to detect the obstacles. Moreover, the fact that only the bottommost pixel rows are evaluated makes the computation extremely fast; only 640 pixels are accessed. Last but not least, the algorithm has a practical strength as the code is relatively short and simple, which allows for an easy implementation, variation, and debugging.

The proposed vision algorithm has the following limitations: first, the collision performance of the drone is largely dependent on the tuning of the parameters due to the high sensitivity. The tuning procedure is thus usually very time-consuming. Also, the drone cannot fly too fast because (1) it should be able to stop well ahead of the obstacle, thereby ensuring a sufficient stopping distance, and (2) it should maintain the bank angle small in order to keep the front camera pointing horizontally forward for the detection of the obstacles ahead. Furthermore, since the proposed vision algorithm yields the control command entirely based on the individual image captured at each frame, it is subject to erroneous obstacle detection caused by the motion blur. Last of all, the biggest limitation of the algorithm would be that it can only work in the Cyberzoo environment with orange poles and black walls as obstacles, and the monochromatic floor and background which have different colors than the obstacles. The algorithm can easily fail if, for example, the floor was painted in a orange or completely black color.

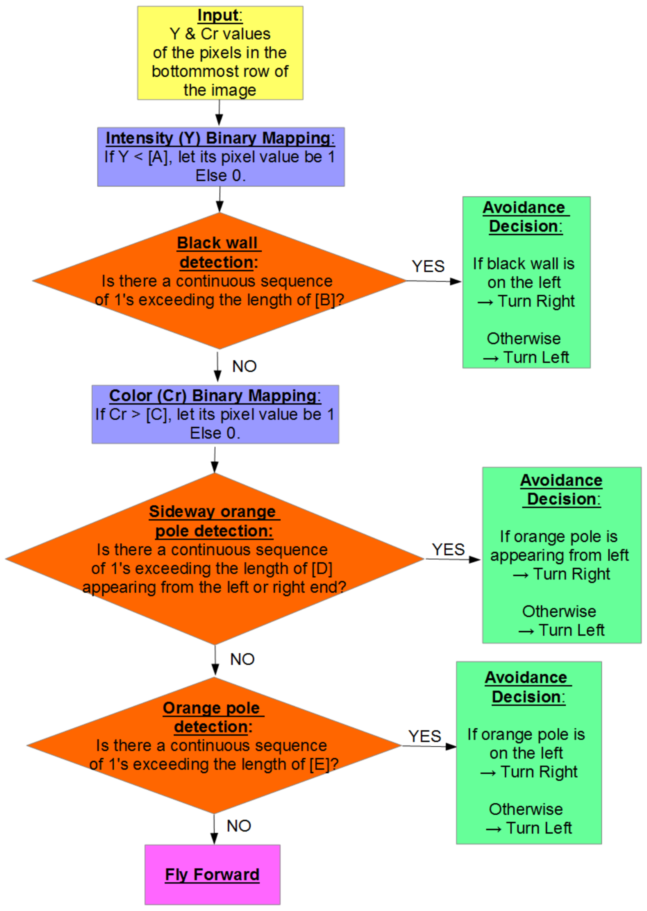

Input:
Y & Cr values of the pixels in the bottommost row of the image
Intensity (Y) Binary Mapping:
If Y < [A], let its pixel value be 1 Else 0.
Black wall detection:
Is there a continuous sequence of 1's exceeding the length of [B]?
YES
Avoidance Decision:
If black wall is on the left → Turn Right
Otherwise → Turn Left
NO
Color (Cr) Binary Mapping:
If Cr > [C], let its pixel value be 1 Else 0.
Sideway orange pole detection:
Is there a continuous sequence of 1's exceeding the length of [D] appearing from the left or right end?
YES
Avoidance Decision:
If orange pole is appearing from left → Turn Right
Otherwise → Turn Left
NO
Orange pole detection:
Is there a continuous sequence of 1's exceeding the length of [E]?
YES
Avoidance Decision:
If orange pole is on the left → Turn Right
Otherwise → Turn Left
NO
Fly Forward

Figure 1: Own Vision algorithm for obstacle detection and avoidance. A to E represent various threshold values

3 Flight Plan

In order to remove the drift of the drone, it is necessary to implement an outer feedback loop that controls and corrects the position and velocity. This can be done by using the Optitrack system in Cyberzoo. In this section, the flight plan using the Optitrack system is proposed and discussed. There are two flight plans proposed - one that actively moves the waypoints, and the other which uses a preset fixed waypoints.

3.1 *Flight Plan I - Moving Waypoints*

This flight plan simply does the two main things. First, if the drone detects an obstacle nearby, it either turns to the left or right. It performs this maneuver in two steps of 45^o to ensure the stable turning of the heading. Secondly, if the drone goes outside the predefined flight area, it turns around and sets the waypoint to the opposite side of the flight area. The turning maneuver is again carried out intermittently in four steps of 45^o turns. If the drone neither detects an obstacle nearby or goes out-of-bounds, it simply flies forward with a constant speed. This flight plan logic is illustrated in Figure 2. Note that this flight plan is called "moving waypoints" because the maneuvers like stopping and turning are carried out by actively and continuously relocating the waypoints for the desired flight trajectory.

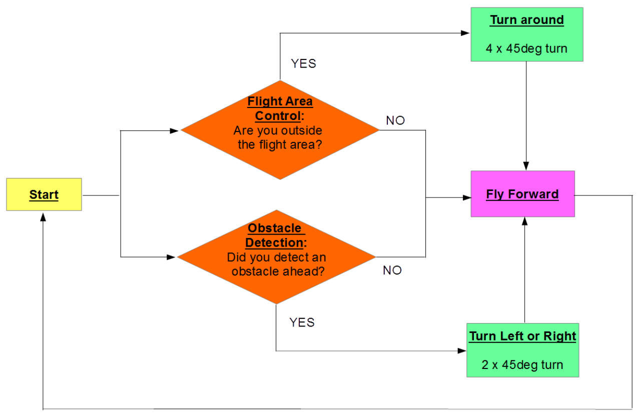

Turn around
4 x 45deg turn
YES
Flight Area Control: Are you outside the flight area?
NO
Start
Fly Forward
Obstacle Detection: Did you detect an obstacle ahead?
NO
YES
Turn Left or Right
2 x 45deg turn

Figure 2: Logic used for the Flight Plan I - Moving Waypoints.

3.2 *Flight Plan II - Fixed Waypoints*

Peng's flight plan

3.3 *Comparison and Choice of Flight plan*

Between the two proposed flight plans, the first flight plan was chosen for the following reasons: (1) The first flight plan moves the waypoints intermittently during the turning maneuver, allowing for a more stable turning of the heading with minimum overshoot. (2) In case of the second flight plan with fixed waypoints, it is plausible that the turning of the heading may not be sufficient to avoid the obstacle. This is a weakness compared to the first flight plan where the turning of 90^o is guaranteed when confronting the obstacle. (3) Finally, the second flight plan could not be fully tested or verified due to the limited time. Nevertheless, the flight plan using the fixed waypoints has a strong advantage over the first flight plan as it can allow the drone to achieve a high-speed flight. This can be elaborated in the future work.

Table 1: Range of pole parameters randomization

Parameter	Range	Unit
Pole number	4 - 10	-
Diameter	40 - 60	cm
Positions (x,y)	2.1 - 7.9 *	m

4 SIMULATION

4.1 *Linear Simulation*

Yazdi's method and results

In order to grasp the concept of the computer vision, a linear simulation program is developed in MATLAB. The simulation program is also used to preliminary test the chosen strategy before the code implementation into the paparazzi system. This simulation also gives our team possibility to easily test the strategy in various setups of the field at the tournament.

4.1.1 *Model and Setup*

The tournament field is modeled based on our observation of the cyberzoo and the rules stated in the first announcement of the tournament[REF]. It is a 10 x 10 meter square field, with a smaller 7 x 7 meter square as the obstacle field , symmetrically placed inside, as shown in three example in Figure3.

Poles, our initial assumption of the possible obstacle, are placed inside the obstacle field. Since there are no setup information, we choose to randomize the pole initial parameters, i.e., the pole number, the pole diameter and the x-y position, as shown in the table1 below. The ranges of position (x,y) are set to make the whole body of a pole to always be inside the obstacle area, considering the diameter ranges. Note that the origin of the field model is on the bottom-left. A limitation is added in the randomization process, so that each pole are always separated by at least a certain distance, which is assumed to be 2 times the diameter of the drone. This last assumption actually limit the number of pole that can be put inside the obstacle field: we found out that randomly positioning more than 10 poles is very difficult. The height of the pole is assumed to be constant, i.e., 2 meters high.

The Drone is modeled from the observation of the AR-drone quad-rotor. Four circles with a heading indicator is used to visualized the drone with diameter of 0.4 m, in a top-down view as shown in Figure 3. The drone motion is modeled using only linear kinematic equations with velocity changes, treating the drone as a point mass. Heading orientation is added to support the camera-view model, explained in the next paragraph. The initial idea is to make a complete non-linear mathematical model of the drone using simulink, and hence have a nonlinear simulation system. However, this is proven to be very difficult and time consuming, especially in determining the PID gain for its stability, and therefore was dropped. The focus of this simulation program then is only to

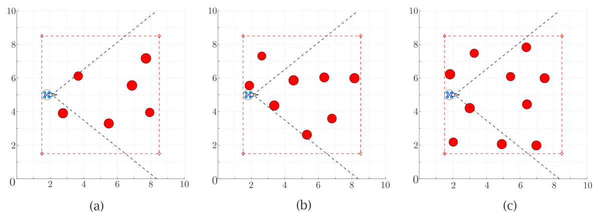


Figure 3: TopViewSamples 3

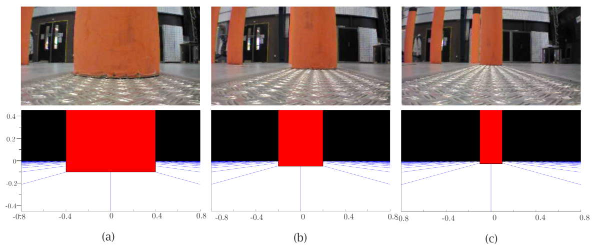


Figure 4: Calibrations

test the strategy in various obstacle setups. The dynamic non-linear simulation will be conducted using paparazi simulator instead.

Camera-view is modeled based on the pin-hole camera model, coupled with the specification of the AR-drone front camera. The result can be observed, from a random position of obstacle, in Figure 5. Further calibration is conducted to match the result with the sample pictures (Figure 4) taken from zero altitude with a pole positioned 0.5, 1 and 2 meters infront of the camera. The calibrations results in a field of view (rounded) of 44 degree vertically, and 77 degree horizontally, as also shown by the dashed line coming from the front of the drone, in the top-down view Figure 3. The poles are modeled as straight rectangles with specific width dictated by the randomization of poles, with a constant 2 meters of height. The color of the pole is not considered in the simulation, assuming a perfect detection of pole width. The floor is illustrated using straight lines, from the camera bottom field until the horizon at the bottom edge of the tournament field. The lines are always in the direction of the drone and only served as an illustration.

4.1.2 Strategy Implementation Result

The implemented strategy is as described in the previous section, summarized in the flow chart in Figure **??** and Figure **??**. The result is shown in series of frames (time-captured) as can be observed in Figure **??**. The first strategy is tested in random obstacle configuration in the tournament field. Table **??** summarized the result of number of collisions, outbounds, as well as covered distance. It should be noted that the result shown in this report is the best result for the strategy proposed, after various calibrations of speeds and threshold.

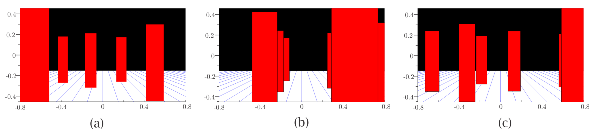


Figure 5: CameraViewSamples 3

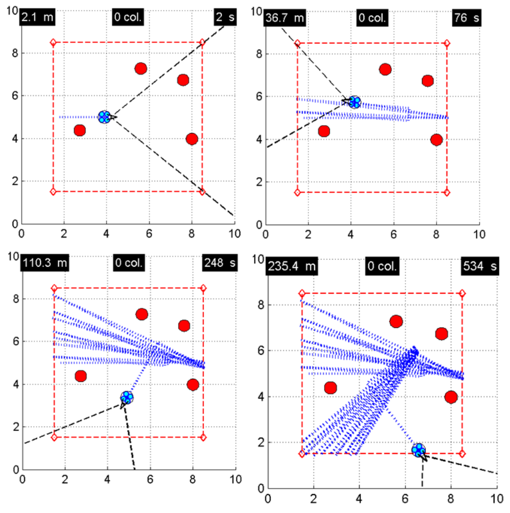


Figure 6: Simulation 4Poles

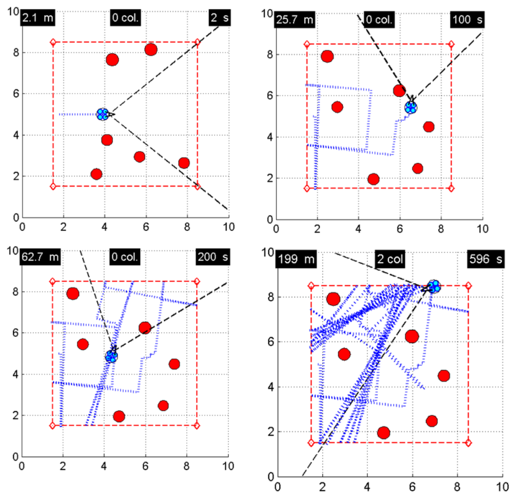


Figure 7: Simulation 6Poles

4.2 MATLAB & Paparazzi Simulation

Another way to carry out a simulation is to use MATLAB for the simulation of the vision algorithm (see Figure 10), and Paparazzi NPS simulator for the simulation of the flight plan (see Figure 11). From the MATLAB simulation results, the appropriate color and width-thresholds for the initial practical trial can be derived. Note that these thresholds can only serve as the initial values because in the end they have to be

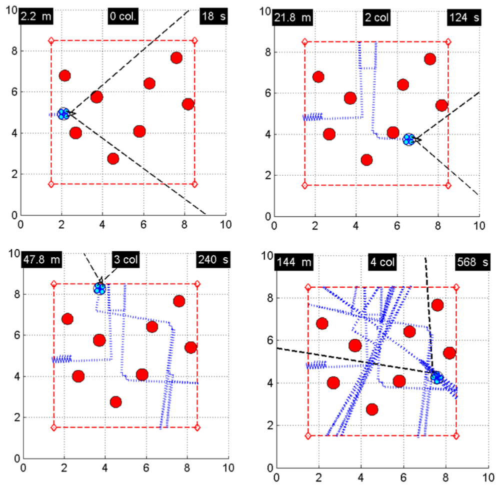


Figure 8: Simulation 8Poles

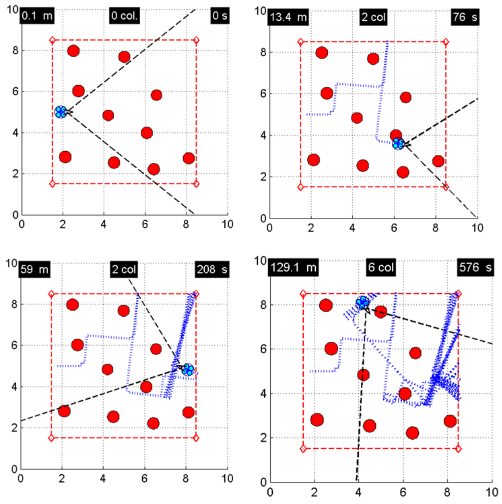


Figure 9: Simulation 10Poles

fine-tuned through real-life tests. As for the flight plan, the Paparazzi NPS simulator can be used to verify the practicability and performance of the proposed flight plan. Although this simulator can yield realistic results, it has a drawback that the simulation cannot incorporate the vision algorithm. Thus, a "fake vision result" is supplied by an NPS file which we designed as a random number generator; it gives 0, 1, and -1 as output where 0 means 'no obstacle detected', 1 means 'obstacle on the left', and -1 means 'obstacle on the right'. The frequency of the occasions when it gives 1 or -1 is adjusted in this NPS file, so that by increasing this frequency, the simulation represents the situation where there are many obstacles present.

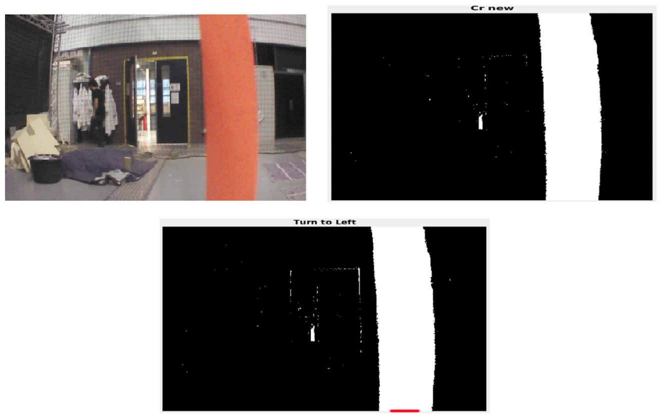


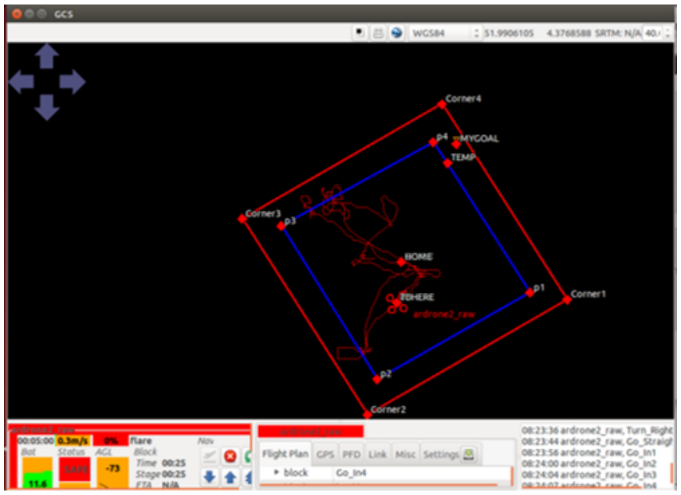
Figure 10: MATLAB simulation of vision algorithm using an image from the AR.Drone 2.0. Top left: original image captured. Top right: binary map computed using a threshold in Cr channel. Bottom: Detection of the orange pole. The red marks on the bottommost row indicate the location of the orange pole nearby.

5 Competition Result and Discussion

After tuning the control and vision threshold parameters through trial-and-error in real tests, it was confirmed that both vision algorithm and flight plan works very well with adequate sensitivity towards the distance from the orange poles and black wall. Also, it was observed that the drone could maintain a stable flight during the maneuvers, such as stopping and turning of the heading. The slow speed of the drone, together with the intermittent turning strategy, allowed the drone to stop and turn with only a marginal overshoot whenever an obstacle is detected or it goes outside the flight area. Regrettably, however, the tuning was done in the environment where the obstacles were rather spaced out widely, and in particular, the black wall was positioned near and parallel to one of the boundaries of the flight area.

During the actual competition, the black wall was positioned adjacently near one of the corners, and there was another orange pole next to it. What happened in the first half of the competition was: the drone moved and made a few turns, and eventually went to near the corner. After it realized that it was out of bounds, it made 180^o turn and attempted to come back inside. However it saw the adjacent black wall, and turned to the other side. Then it saw the orange wall and turned back to the previous heading. And it saw the black wall again, and so on. After drifting slowly towards the black wall, it touched the black wall and had to be restarted due to the low battery.

In the second half of the competition, the drone was started after being repositioned in the home position, so it did not go into the corner and did not get trapped again. As a



```
08:23:36 ardrone2_raw, Turn_Right
08:23:44 ardrone2_raw, Go_Straigh
08:23:56 ardrone2_raw, Go_In1
08:24:00 ardrone2_raw, Go_In2
08:24:04 ardrone2_raw, Go_In3
08:24:07 ardrone2_raw, Go_In4
```

Figure 11: Paparazzi simulation of the flight plan. The blue lines indicate the boundaries of the flight area.

result, the drone covered the traveled distance of well more than 100 m. Although this is not quite a high result compared to the other teams, our drone had the fewest collisions with the obstacle, and our team won the 6th place out of 13 teams in total.

In the following list, the main reasons are addressed as to why it could not be foreseen that the drone may get trapped near the boundaries/corners:

- In the Paparazzi simulation, the vision algorithm was "mimicked" by a random number generator. Hence, there was never a possibility of a drone getting trapped in a place.
- During the real tests for tuning, the black wall was located parallel to one of the boundaries and further way from the corner. So, the drone could not be possibly trapped by the "blocking" of the wall.

To solve this problem in the future, the following strategies may be used to improve the flight plan:

- "Breakthrough Maneuver": Incorporate a maneuver that when a drone makes more than two consecutive turns back and forth, turn the heading only 45^o and fly forward for a certain period of time while suppressing command from the vision result.
- "Temporary Blinding": In case the displacement of the drone is too low during a fixed time period, flexibly adjust the vision threshold parameters (e.g. the width threshold for the pole detection) such that the drone becomes less sensitive towards the obstacles nearby.
- "Circular Flight Area": Define the flight area as a circle, instead of a square. This will prevent the drone from going too close into a corner.

6 Conclusion

References

Appendix A: Data

Appendix B: More data