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

In pursuit of achieving a vision-based autonomous flgiht of AR Drone 2.0 in the Cyberzoo environment with OptiTrack system, a simple obstacle detection algorithm which involves the color- and size-thresholding of the obstacles, and a flight plan which actively and continuously moves the outer-loop waypoints is proposed, simulated and implemented for the real-life test. The actual flight in the competition showed satisfying results with hardly no collisions and stable avoidance maneuvers, but recommendations are given to optimize the flight plan for the larger coverage.

1 INTRODUCTION

This report is structured as follows. After explaining the project requirements, three main equipments for the project is introduced. Section 2 follows with the explanation of the vision algorithm used to achieve the goal of the project. Our Drone flight-plans are elaborated in section 3 by comparing two possible plans our team came up with. Two types of simulation were conducted before the tournament, elaborated in Section 4 along with the result analysis. The competition result, as well as the discussion, follows in section 5. Section 6 finally wrap the report with few concluding remarks.

1.1 Project assignment

1.2 AR Drone: A Quad-rotor MAV

A Quad-rotor named AR Drone 2.0 is the chosen MAV in this project. The first version of the vehicle was build by the Parrot company (France) in 2004, and released for public in 2010, aiming for the market of video games and home entertainment[1]. The platform of the vehicle is a Quad-rotor, a four fixed propeller that are rotated by four brushless motors, each controlled by an ATMEGA8L 8bit microcontroller. The AR.Drone is equipped with 3 cells battery that supplies 11.1 V and 1000 mAh providing power for 10 to 15 minutes of flight. The platform is very popular in MAV research field[1][2][3], since it has the capability of both hovering and fast forward cruising. In order to make the vehicle acceptable for public, a very effective control and stabilization system is mandatory for a easy piloting platform. The general view of the AR Drone 2.0, the one used for this project, is shown in Figure 1 below.

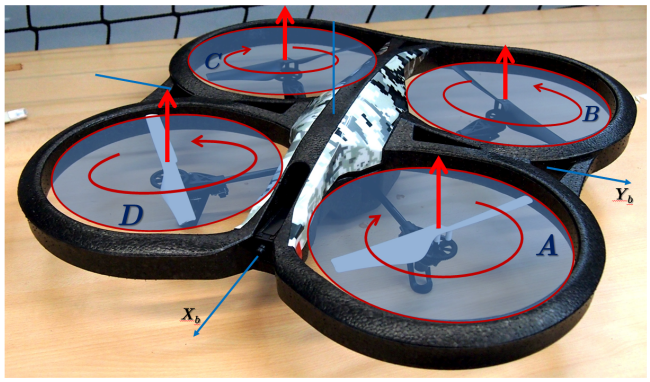

Figure 1: The AR Drone 2.0 from the Parrot Inc (France) - with schematic of the four propellers direction of rotation

Quad-rotor is controlled by adjusting the thrust and rotation combination of its propellers. AR Drone propeller setup is as shown in Figure 1, where two of the propeller (A and C) turn clockwise, while the other two (B and D) turn counter clockwise. On stationary hover condition, every propeller produce the same thrust from the same RPM. Torque from each motor, therefore, are canceled by the combination of propeller rotation. To have a positive pitching motion, more power is given to propeller B and C, while reducing power in A and D to keep the balance with the weight and the desired forward/backward speed. Rolling motion is controlled in similar manner, positive rolling is achieved by giving more power to A and B, while reducing power in C and D. Finally, positive yawing is controlled by giving more power to propeller A and C, while reducing power in B and D. Controlling task for stability is challenging, because the rotational motion, i.e, roll, pitch and yaw, and the translational motions, i.e., heave, sway, and surge, are coupled. The work of [4] elaborated this challenge comprehensively.

The Guidance, Navigation, and Control system of AR.Drone is supported by an a set of sensory system consisting a 3-axis accelerometer, a 2-axis gyroscope, a 1-axis gyroscope, and two ultrasonic sensors. The ultrasonic sensor is used for altitude and vertical displacement estimation, while the other is part of an Inertial Measurement Unit (IMU) that is essential for control and stability of the vehicle.All the sensors is chosen to be as low-cost as possible, which implies that the embedded control system have to deal with a lot of

[*]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

bias, misalignment angles, and other errors[1]. Furthermore, two cameras are equipped in AR.Drone, which are not mainly intended to be used as sensor for the flight. Coupled with the IMU rotation measurement and vision algorithms (explained in the next section), however, the camera can be used in vertical speed estimation (camera pointed down).

1.3 CyberZoo and OptiTrack

At march the 5th, 2014, TU Delft's so called **Cyber Zoo** is opened as a new research and test laboratory for robots and unmanned vehicles. Initiated by the TU Delft Robotic Institute, this facility is a 10 x 10 meters area covered by 7 meter high net. The facility is located in the hangar of the Faculty of Aerospace Engineering, and it is the designated field in which the project MAV will be flown.

The Cyberzoo is equipped with a three dimensional optical motion tracking system based on called OptiTrack by NaturalPoint Inc[5][6]. The system consist of 24 camera, observed in Figure 3, to triangulate the position of retro-reflective markers, placed on the surface of a motion platform in a specific distance between each other, such as shown in Figure 4 below. By defining a set of marker positions as a rigid body, the OptiTrack system can recognized the particular pattern and track the motion continuously in a screen, also shown in Figure 3. The OptiTrack then provide, through a UDP connection, a local positioning data that is almost similar with the data provided by a GPS. with the This OptiTrack system is known to be a low-cost system that have accuracy comparable with its more high end competition, the well established Vicon motion capture system[5].

Figure 2: The cyber zoo inside the hangar of the Faculty of Aerospace Engineering, Delft University of technology. Currently covered by a thick canvas that serves as a background for computer vision for the robots inside

1.4 Paparazzi: Open Source Autopilot

The AR.Drone in this project will be equipped with the open source autopilot system Paparazzi, which is mainly developed and maintained by the Ecole Nationale de l'Aviation Civile (ENAC) in Toulousse, France. The system run in a

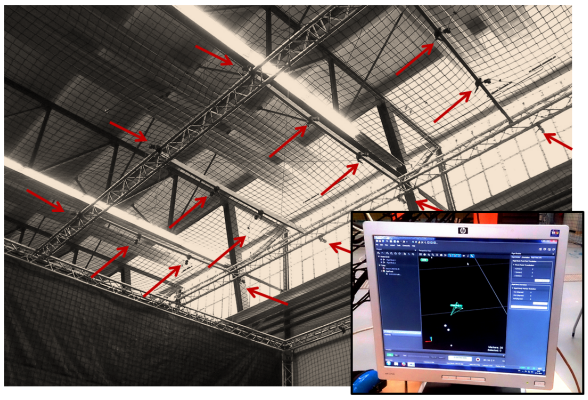

Figure 3: Some camera positions on the roof of the cyber zoo. (inset) the OptiTrack GUI that combines data from the 24 camera to triangulate marker positions inside the cyberzoo

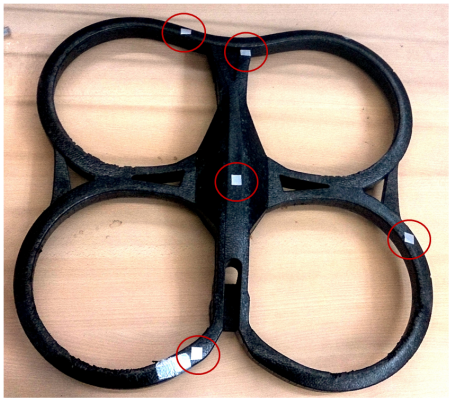

Figure 4: Retro-reflective marker is installed on the surface of AR.Drone hull. Each Hull have a specific position combination that can be recorded and tracked by the OptiTrack system

Linux platform, and has been used by hundreds of research from all around the world[7][8]. The opening window of this software is shown in the Figure below.

Paparazzi's Ground Control Station (GCS) is used to determined the flight plan of a vehicle to fly autonomously. The flight plan is mainly set using combination of several blocks, or modules, embedded in the GCS, e.g., the take-off, set direction to a waypoint, go to a waypoint, or go make a circle. A wifi connection is established between Paparazzi and the vehicle for a two-way communications, enabling the AR Drone to sent its states back to the GCS and operator. The flight plans can be interrupted by the operator anytime during the flight.

Inside the cyber zoo, however, the states of position is not given directly by the drone, since it is flying in a GPS-denied environment. Instead, Paparazzi must be modified to receive data from the OptiTrack local positioning. This is done by

using the NavNet module in Paparazzi, coupled with a view changes in the airframe and flight plan code. The local position data from each rigid body is sent through a UDP connection, and therefore the computer platform, running Paparazzi in the CyberZoo, is required to have two connection port.

2 Vision algorithm

2.1 Vision-based Navigation Literature

This section is about the vision algorithm used in the literature.

2.1.1 Optic Flow

When two frames are used one can see the movement of the objects relative to each other. Therefore it is necessary to recognize textures points in both images, so the movement between a texture point in both images can be determined. This optical flow can be used for calculating the distance to and between objects, so the 3D world can be reconstructed and the drone knows what it should avoid [**?**]. Also, from the rate of expansion the time-to-contact can be calculated. Because two frames are required and textures need to be recognized for optical flow, this method requires more computation time, but the resulting information is much more useful.

Determining optical flow from two images:

- Feature detection: One can use the methods of Harris and FAST to detect corners in images. The speed of this calculation can be increased by downsampling the image. This will make it easier to find similar corners because the average of a few pixels compared to the average of a few pixels will have more similarities than comparing one pixel to one pixel.

- Feature tracking: The movement between two corners on two images can be expressed as a vector. A taylor expansion illuminance change and a brightness constancy constraint can solve the changes in lighting between the two images. The Lucas-Kanade method is a differential method used for estimating the optical flow by solving the flow equations for the pixels in het neighborhood of the tracked pixel. By doing this it has less noise influences and ambiguities.

2.1.2 Segmentation and Object recognition

Segmentation [9] is a method that is used for image processing, digital images are partitioned into multiple segments to extract the essential characteristics. It is still challenging since the wide range of image styles influences the difficulty of the process. Therefore two properties need to be defined for segmentation methods: 1. capture the important regions, because this often reflects the global aspects of the image. 2. be efficient, running in time nearly linear in the number of image pixels.

The object recognition uses a similarity measure between the two feature vectors. One typical similarity measure is Euclidean distance between two vectors. For the computation of the feature vector representing the image, its color moments (i.e. mean, standard deviation, and skewness) can be used for HSV color system. These feature vectors can be compared with the database, and the most similar object class with the minimum weighted Euclidean distance can be therefore chosen.

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 5) 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 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

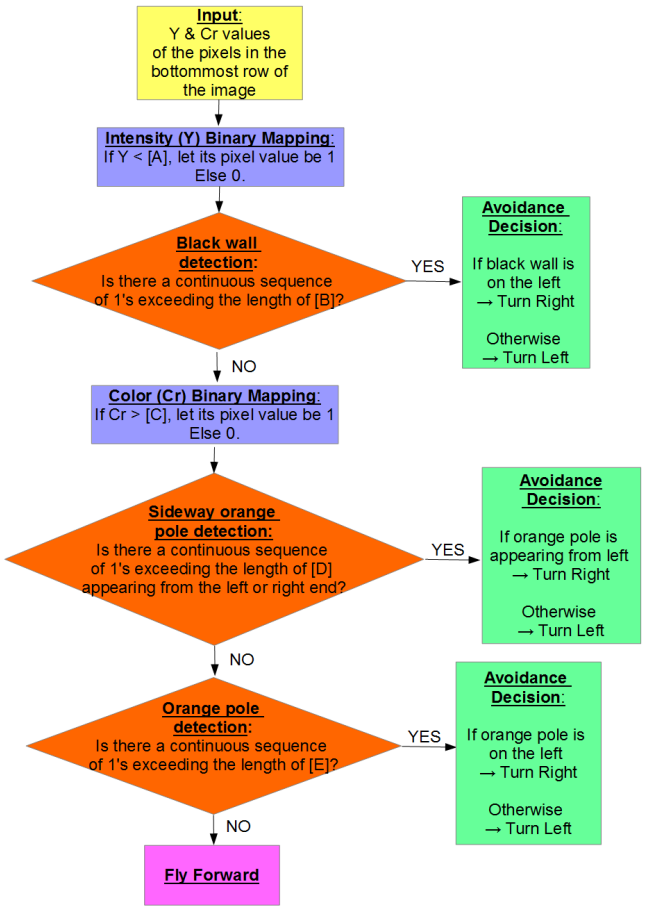

Figure 5: Own Vision algorithm for obstacle detection and avoidance. A to E represent various threshold values

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 obsta-

cles 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.

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.

The proposed flight plan mainly uses the principle of "continuous relocation of waypoints", and 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 6. 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.

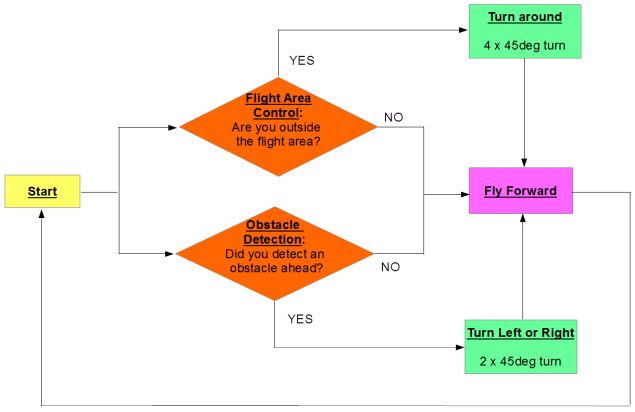

Figure 6: Logic used for the Flight Plan I - Moving Waypoints.

Table 1: Range of pole parameters randomization in the linear simulation

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

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. 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 Figure7.

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 7. 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 model also consist of a random addition for the direction in each time step to simulate Drone drifting in the real life.

It should be noted that the initial idea is to make a complete non-linear mathematical model of the drone using simulink, and hence to 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 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 8. Further calibration is conducted to match the result with the sample pictures (Figure 9) 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 7. 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.

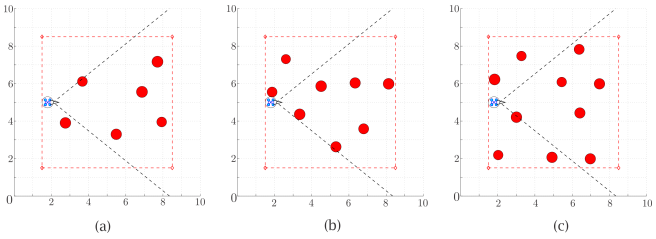

Figure 7: Three sample of obstacle random positioning in the competition field (cyberzoo), (a) with six poles, (b) with eight poles, and (c) with ten poles

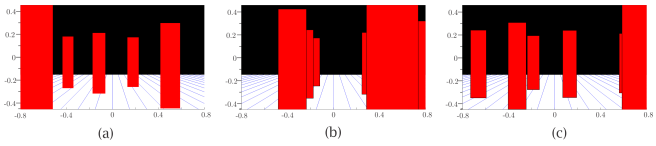

Figure 8: Poles position as viewed by the AR.Drone front camera for (a) six poles, (b) eight poles, and (c) ten poles, corresponded to the sample (a), (b) and(c) in Figure 7. Notice that not all poles are covered by the camera field-of-view (FOV)

4.1.2 Strategy Implementation Result

The implemented strategy is as described in the previous section, summarized in the flow chart in Figure 6. The result is shown in series of frames (time-captured) as can be

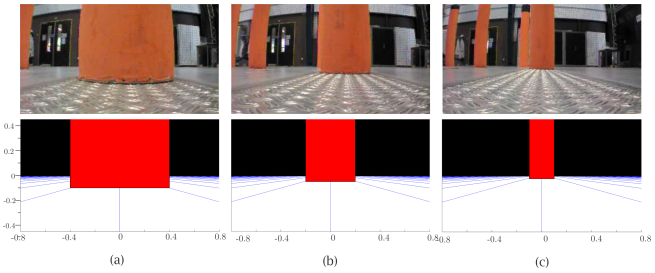

Figure 9: Calibration of the camera view model, conducted by comparing real snapshots from the AR.Drone front camera (top) with the result in the simulation (bottom). The drone on ground and a pole is positioned at (a) 0.5 meter, (b) 1 meter, and (c) two meter from the front camera.

observed in Figure 10, for the situation where eight poles is used. The first strategy is tested in random obstacle configuration in the tournament field. The result of number of collisions, as well as covered distance, can be seen in the fourth frame. It should be noted that the result shown in this report is not the best result for the strategy proposed, since it result in four collisions. Tuning of threshold, as explained in the previous sections is required, which unfortunately have not been conducted thoroughly in this project due the limitation of time.

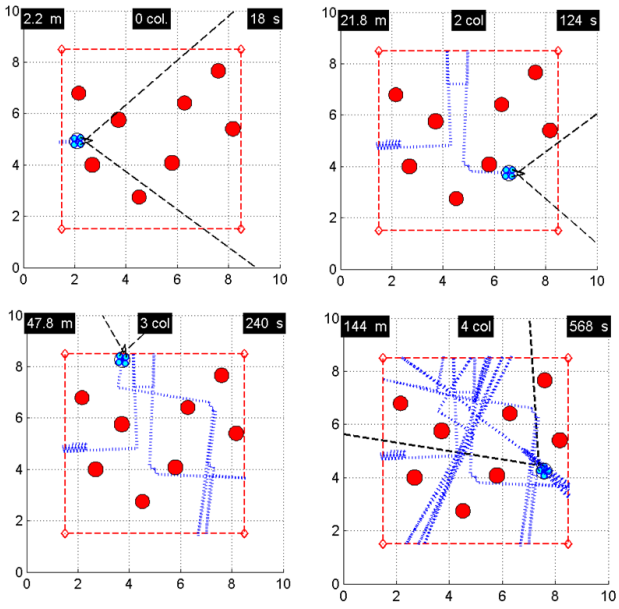

Figure 10: Simulation runs for 10 minutes, for cases with eight poles positioned in random.

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 11), and

Paparazzi NPS simulator for the simulation of the flight plan (see Figure 12). 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 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.

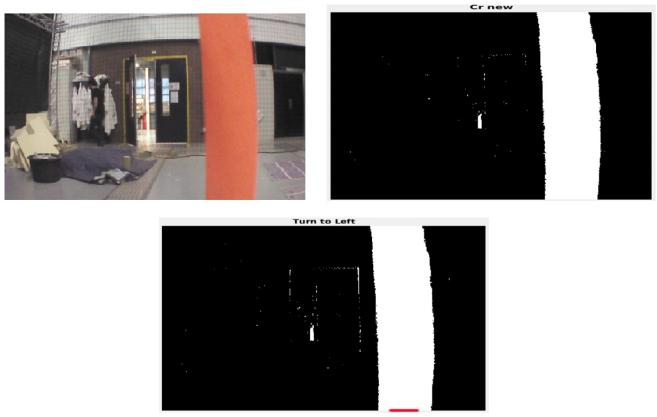

Figure 11: MATLAB simulation of vision algorithm using an image from the AR.Drone 2.0. Top left: original image captured. Top right: binary map computed using a threshold in Cr channel. Bottom: Detection of the orange pole. The red marks on the bottommost row indicate the location of the orange pole 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

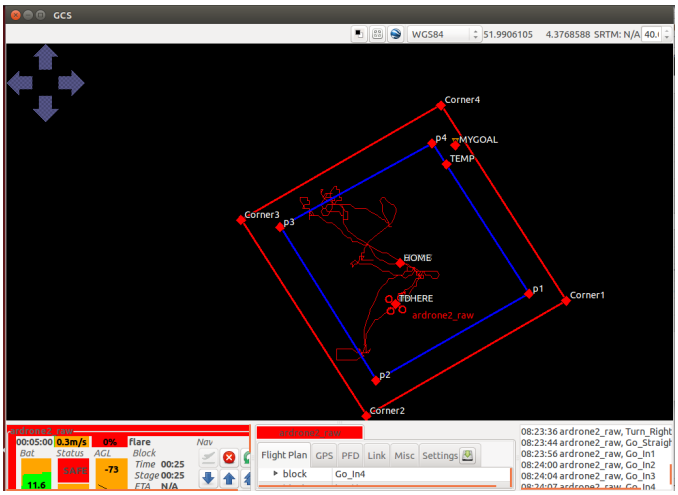

Figure 12: Paparazzi simulation of the flight plan. The blue lines indicate the boundaries of the flight area.

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 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 lo-

cated 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

The drone performed well in terms of not crashing at all. The vision algorithm worked as required as all obstacles were detected and the drone turned away from them. On the other hand, it was quite slow in moving, resulting in not much distance flown. This speed had to be low to react before colliding with an obstacle. Another limitation was the dependency on the tuning. The drone had a good performace on orange and black obstacles as it has been tuned on them, but had some troubles with other colors or patterns.

The flight plan was a succes. Both the fixed waypoint and moving waypoint method could keep the drone inside the competition area and moved the drone from one waypoint to another. To prevent the drone from being trapped a breakthrough maneuver, temporary blinding and a circular flight area can give solution.

References

[1] David Vissière Nicolas Petit Pierre-Jean Bristeau, François Callou. The navigation and control technology inside the ar.drone micro uav. In *18th IFAC World Congress*, pages 1477–1484, Milano, Italy, 2011.

[2] J. Pestana, I. Mellado-Bataller, Changhong Fu, J.L. Sanchez-Lopez, I.F. Mondragon, and P. Campoy. A general purpose configurable navigation controller for micro aerial multirotor vehicles. In *Unmanned Aircraft Systems (ICUAS), 2013 International Conference on*, pages 557–564, May 2013.

[3] Jacobo Jiménez Lugo and Andreas Zell. Framework for autonomous on-board navigation with the ar.drone. *Journal of Intelligent and Robotic Systems*, 73(1-4):401–412, 2014.

[4] S. Bouabdallah and R. Siegwart. Full control of a quadrotor. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 153–158, Oct 2007.

[5] Clint Hansen, David Gibas, Jean-Louis Honeine, Nasser Rezzoug, Philippe Gorce, and Brice Isableu. An inexpensive solution for motion analysis. *Proceedings of the Institution of Mechanical Engineers, Part P: Journal of Sports Engineering and Technology*, 228(3):165–170, 2014.

[6] J.Rogelio Guadarrama-Olvera, JoséJ. Corona-Sánchez, and H. Rodríguez-Cortés. Hard real-time implementation of a nonlinear controller for the quadrotor helicopter. *Journal of Intelligent and Robotic Systems*, 73(1-4):81–97, 2014.

[7] Joachim Reuder, MariusO. Jonassen, and Haraldur Ólafsson. The small unmanned meteorological observer sumo: Recent developments and applications of a micro-uas for atmospheric boundary layer research. *Acta Geophysica*, 60(5):1454–1473, 2012.

[8] Pablo Royo, Enric Pastor, Cristina Barrado, Eduard Santamaria, Juan Lopez, Xavier Prats, and Juan Manuel Lema. Autopilot abstraction and standardization for seamless integration of unmanned aircraft system applications. *Journal of Aerospace Computing, Information, and Communication*, 8(7):197–223, 2011.

[9] Pedro F. Felzenszwalb. Efficient graph-based image segmentation. *International Journal of Computer Vision, Vol. 59, No. 2, September 2004*, 2004.