Autonomous Flight With the AR Drone©

Maarten Inja & Maarten de Waard
5872464 & 5894883

February 1, 2011

Contents

1 Introduction

1.1 Our Goal

Our absolute goal, is winning in some subtasks of the Summer-IMAV 2011 Indoor competition [1]. This is a competition for flying radio controlled vehicles to autonomously solve some tasks. For this competition, we tried to solve the following sub-tasks:

- Pick-up Object
- Exit Building
- Release Object

We used the Parrot AR Drone© to solve these problems.

1.2 The AR Drone©

The AR Drone© is an over WiFi remote controlled quadrocopter that has several onboard sensors:

- One vertical camera, pointing downward
  This is also used for stabilisation and calculating the speed. It has a 63° angle.

[1] This link refers to a pdf file explaining the competition.

Figure 1: The AR Drone©

- One horizontal camera, pointing forward
  This is a 91° angle camera.
- Ultrasound altimeter, to measure the altitude
- 3 axis accelerometer (measures propellor acceleration)
- 2 axis gyrometer
- 1 yaw precision gyrometer

Furthermore, it has an onboard computer system running Linux.

2 Controlling the AR Drone©

The AR Drone© has, like any other flying vehicle, the usual 3 critical flight dynamic parameters for the angles of rotation; pitch, yaw and roll (See figure 2). In addition to these three parameters the AR Drone© has the *gaz* parameter to control the upward or downward change. This allows a pilot to increase or decrease the altitude directly.

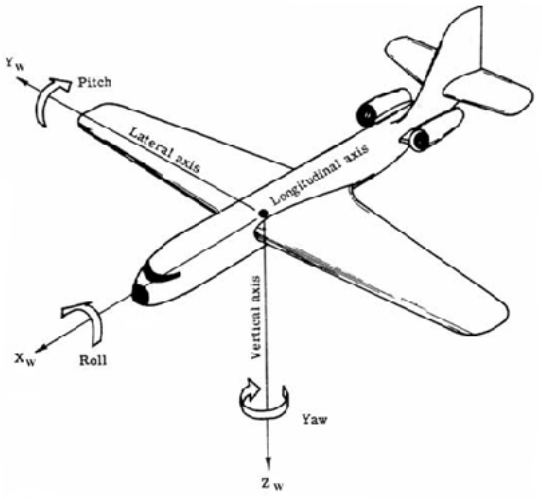


Figure 2: Movement directions for flying vehicles

2.1 ROS And AT Commands

ROS, or Robot Operating System, is an open-source, meta-operating system for your robot. People can write their own packages for all kinds of robots that other people can reuse. This includes modules to

control OpenCV, pathfinding etc. We explored options to control the AR Drone© with ROS but we came to the conclusion that the existing module to control an AR Drone© with ROS does not work completely (although we could have fixed this ourselves). Furthermore it was way more complicated than what we actually needed. This is why we looked into other options such as AT Commands.

AT Commands are commands sent to ports over the network telling the AR Drone© what to do directly. The AR Drone© listens to some ports for these commands and sends the navigation data and the video stream data back to the application using different ports. This is the most direct manner to control the AR Drone© and we were always able to have the AR Drone© take off and land without problems by sending the take off and land commands. However, while sending commands to the AR Drone© was really easy, decoding the navigation data and the video stream was not. We therefore decided to check out the Software Development Kit. Written and documentated by the Parrot team itself.

2.2 Software Development Kit

The Software Development Kit (SDK) did not come with it's documentation attached. We strangely found the documentation by clicking on a link in a remote part of the AR Drone© developer forum and it helped us a great deal.

The SDK is completely written in the programming language C. It allows you to use those features of the AR Drone© you want (or had time to implement). Figure 3 shows us what we need to implement ourselves. We decided to grab an example provided by the SDK and modify this to our needs. What we had to rewrite were:

- The main initiation, update and close functions of the application. This is were we later would be sending the movement commands
- The navigation data processing functions. This had a seperate thread, especially to fetch the navigation data from the drone.
- The video stage functions. The SDK has a complete pipeline that receives, checksummes and processes the whole image. Just like the navigation data example this ran on it's own seperated thread. It uses GTK (the standard C library for GUI's).

This was rather complex C, and a lot of it we did not understand (both the example and the code in the SDK). So we decided to extend C with Python.

2.3 Extending C With Python

After we were able to save an image frame to a file in C we could have Python read the file, process it, and send movement commands back to C. The main functions that took care of the initiation, updating and closing of the application instantiated and closed Python as well as updated Python. When C updated Python it supplied Python with a tuple consisting of the frame counter (to retrieve the frame image) and the navigation data. Python returned pitch, yaw, roll and gaz values, or something to indicate that the application should exit.

While one might think that we decided to stick with the slowest manner to send the frame data from C to Python, it is not as bad as one would expect. Because each frame is saved we could playback a run in which we flew with the AR Drone© and debug it completely and accurately. Later, we even saved the navigation data so we would be able to get an identical, simulated stream of information that was not actually sent by the drone, but rather read from a file. This allowed us to easily debug any run and test probable solutions without flying with the AR Drone© over and over again.

Another something special we did with Python was using OpenCV. OpenCV can be used with Python quite easily, albeit some basic functions did not work even though everything indicated it should be working, we were quite happy with it.

While it was not part of our direct goals, the bridge from the SDK written in C to our code written in Python using only one function that transfer all the necessary data is something we expect could prove to be quite useful to other people as well.

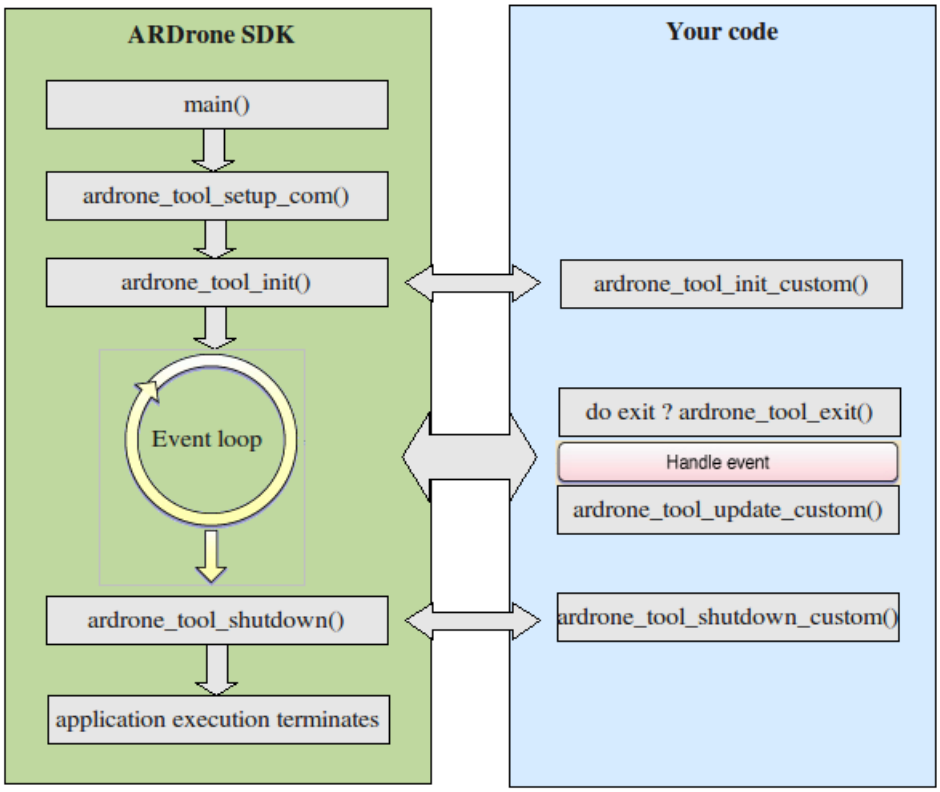


Figure 3: The application life cycle (picture from AR Drone© API Webpage)

3 Methods Used

We divided the first subtask in three subgoals: Finding the object; tracking the object and picking up the object.

3.1 Finding The Object

To find the object we used a simple routine, in combination with a more complicated recognition. The routine is: Start flying at a low height, and then turn 360 degrees. While turning we constantly check the front camera (since we can only check one camera properly at the same time) for our object. If we had no indication of the object at this particular height we would increase the search height and begin again. If the object had been found we would store that height and go back to that when the object was lost. This, in order to speed up the search process later on (it was not unknown for the AR Drone© to lose the object some times).

We chose to make an object that had a bright pink color. This simplified things for us: We only had to recognize a big enough surface of our color. We tried a couple of things to recognize our color:

The first step was to define the values that our color ranged from. We chose to divide our image in HSV values. This divides the image in 3 different layers: Hue, Saturation and Value. These layers represent the values of the image's "Hue, Saturation and Value" as shown in figure 4. OpenCV changes these values to a range of 0 to 255 (to create images it can show) by dividing the Hue by 2, and the saturation and value by $\frac{100}{255}$. The advantage of HSV above RGB (Red Green Blue) values is that it is easier to recognize if your color needs to be a bit lighter (thus, increasing the Value) than that your color needs to be a bit greener.

OpenCV placed the pixels that were in the right HSV range in a new picture. This picture had a value of 1 on the pixel where our color-values were good, and a value of 0 where they were bad. This resulted in a white blob where almost every white pixel was on our object. Something like figure 5. Unfortunately the computer can not understand this. We still need to let the AR Drone© know where the object is in

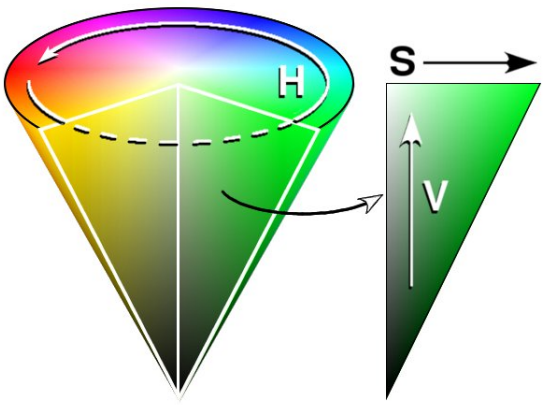
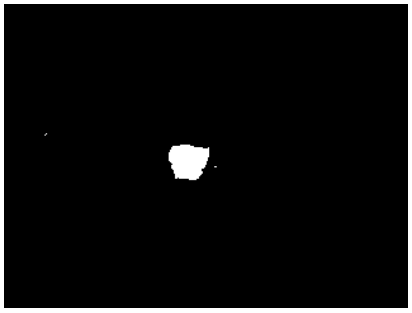
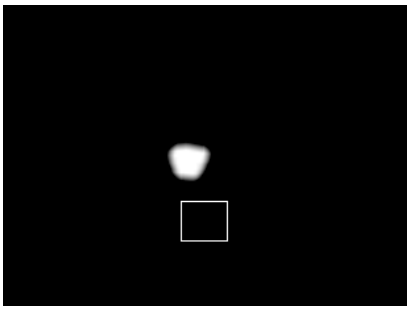


Figure 4: Hue (H), ranging from 0 to 360 degrees; Saturation (S), ranging from 0 to 100% and Value (V) ranging from 0 to 100%



(a) The image after the first processing.



(b) The image after the second processing.

Figure 5: Results of processing a frame, the square in the right image debug-output.

this picture. To find the centre of the object, we tried histogram backprojection.

3.2 Tracking The Object

Histogram backprojection is a technique that takes a histogram (a representation of the distribution of the colors in an image) of your target, and matches it for the histogram of the same size of a part of the image around each position (x, y) in the image. Where the histogram of the target matches the histogram of some location in the image a great deal, the location will have a big number. And, of course, it is the other way around for those locations that do not match. Unfortunately we experienced our first real problem with OpenCV, the function had a bug that we could never solve.

Therefore we tried a different, somewhat simpler method, called "Template Matching". This is a bit like histogram backprojection, but instead of using histograms, we use a template of the object we wanted to see. This is is a somewhat less complicated method and it works a little bit less well but it did the trick. And since we allready had an image that was only black or white (0 or 1), our template simply had to be a white square. This would match the white space in our image, but not a single white pixel. This resulted in an image like in figure 5. As you can see, the edges are a bit fuzzy. Now we can use the OpenCV function "minMaxLoc" to locate the centre of the object, which is the brightest pixel in the image.

At first the size of our template was fixed. It was a 10 by 10 pixels square. This resulted in our second image having too many pixels with a value of 1 (and thus being the brightest). When we were able to calculate the distance of the object (see section 3.3.3), we made the template size variable, meaning that we always exactly found the centre of the object. The result can be seen in figure 6.

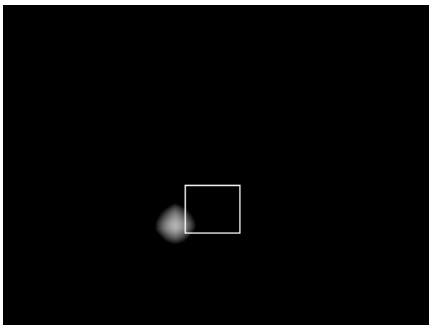


Figure 6: The result of scaling the template image to the distance.

3.3 Picking Up The Object

We chose an object with a handle, and a hook on our AR Drone© . The advantage of this is that we do not have to hover above, or land on our object, but we can simply fly over it, and pick it up with the hook. This is a lot easier, because the AR Drone© has a flying error, which complicates hovering above the object.

To be able to pick up the object we went through this routine:

- Put the centre of the object in a specified square (the white square in figure 5).
- keep it there for a second, to compensate the flying error.
- fly towards the target, while keeping it in the square
- When the AR Drone© is close enough, switch to watching the bottom camera, fly a bit higher and fly forward, to pick up the object.

To be able to do these things, we had to do some tricky things:

3.3.1 Calculate Distance

The most important thing we did was calculating the distance to the object. The ammount of pixels that belong to our object should have some relation to the distance to the object because the object is smaller if we are further away from it. So all We had to do was to count the number of non-zero pixels in the second processed window. We calibrated this by counting the amount of those pixels for a number of known distances and then used an online formula finder [2] to find the formula that calculates the distance.

3.3.2 Variable Steering Commands

Another important factor in moving towards the object were variable steering commands. It is really important to steer towards the object quickly once you can. The faster the better because the flying errors are quite enormous. It is not unusual to see the AR Drone© drifting off quite a bit even if we only want to rotate it 360 degrees. However, once the AR Drone© has positioned itself in front of the object it will overshoot with the corrections if it drifts of. To compensate for this effect we made sure the strength of the steering values are lower as the AR Drone© is closer towards the target. This is the case for pitch, roll and even the gaz commands. For the yaw, we did almost the same, but in stead of letting it depend on the object distance, we let that depend on the place of the object in the picture.

3.3.3 Blind Flight

The AR Drone© can only send one of the two video-streams over the network to our application. When we decide it is time to fly over the object, to pick it up, we switch to the bottom camera. This allows us to steer the AR Drone© in the right direction when we notice it drifting off but in the mean time we do not see any of the more interesting data (the front view). If we miss the object completely we might fly against a wall. To solve this we granted the AR Drone© a limited amount of time on the bottom camera to pick up the object before we switch back to the front camera and restore whatever it might have done.

[2] Zunzun.com function finder

4 Results

We have not reached all of our goals. There were many problems that we encountered along the road that we were able to solve. We decreased the random flying errors of the AR Drone© by putting random stuff on the ground (even though it still flies into walls some times) and we were able to work with the complex SDK and work around malfunctioning OpenCV functions. But in the end the the flying errors of the AR Drone© proved fatal. The program we coded is able to successfully pick up the object, but not always, and then crashes into the nearest obstacle, hard.

So we did accomplish the first sub-task, which was the hardest one. Furthermore we created a bridge between the SDK and Python, allowing any newcomer to simply use that (albeit slower, much easier) from the start. Another valuable resource we coded is the option to replay any run, complete with navigation data. One can not understate how valuable that has been throughout the project.

One can download our whole project from our Google Code repository. A video of a succesfull pick-up (and crash afterwards) can be downloaded here. If one downloads the repository (as-is, at this moment) and runs *python uvhar.py* in the directory *ardroneApi/Uvhar/Release* one can see the replay of that run, including all the navigation and replay data.

5 Future Work

Our project is officially finished, but there are fixes, some minor and other requiring a lot of work that could finish our other sub-tasks.

- A different pick-up method, like magnets or velcro. When the AR Drone© picks up the object the object start swinging under the AR Drone© . This causes a lot of navigational troubles. Landing on the object, and picking it up using velcro might remove navigation troubles caused by a swinging object under the AR Drone© but it also will take away the whole bottom camera, which the AR Drone© uses to stabilize itself.
- A smaller object with less weight. Easier to pick up and to fly with.
- Something to keep the object from being blown away by the wind the propellors of the AR Drone© create. If we make the object smaller and lighter this will eventually become a problem.
- Constant altitude when flying over objects. The altitude is measured to whatever is under the AR Drone© . If we tell the AR Drone© to fly at a height of 2 metres and it flies over, for example, a table, the AR Drone© will see that it's flying over a table (or rather something higher than the ground), and increase gaz to fly 2 metres above that. This shouldn't be too hard to fix, since we can access the navigation data.
- Small adjustments to the code to exit the building. We already have the complete code to recognize bright colors. We could easily put some markers on the exit of the building and have the AR Drone© fly through those markers. This would complete our second sub-task.