Autonomous Flight With the Ar Drone©

Maarten Inja & Maarten de Waard
5872464 & 5894883

January 31, 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.
- One horizontal camera, pointing forward
  This is a 91° angle camera.
- Ultrasound altimeter, to measure the altitude

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

Figure 1: The Ar Drone©

- 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.

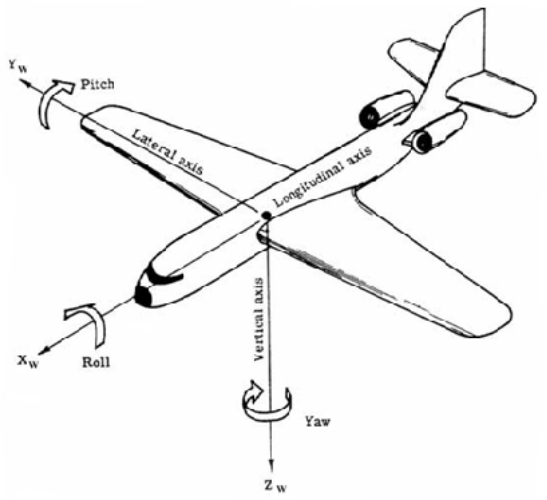

Figure 2: Movement directions for flying vehicles

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 (but we could fix this ourselves) but also that 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 send 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 therefor decided to check out the Software Development Kit. Writen and documentated by the Parrot team itself.

2.2 Software Development Kit

The Software Development Kit (SDK) did not came 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 need. 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.

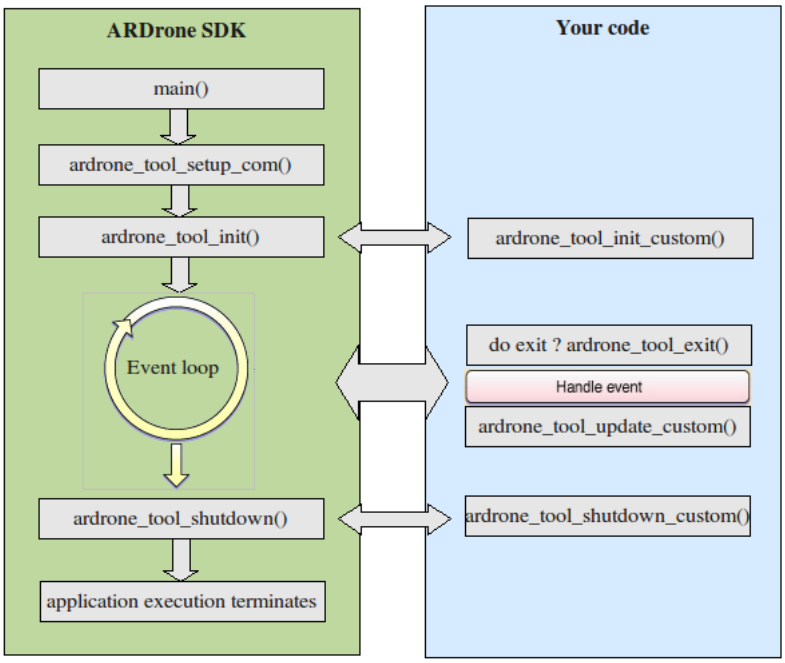

Figure 3: The application life cycle (picture from Ar Drone© API Webpage)

2.3 Extending C With Python

After we were able to safe an image frame to a file in C we could have Python read the file, process it, and send movements 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, and 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 send by the drone but rather read from 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.

3 Methods Used

We divided the first subtask in three subgoals: Finding the object; tracking the object and picking up the object. This shows step by step how to do that.

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 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.

We chose to make an object that was bright pink. 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 of 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 this picture. To find the centre of the object, we tried histogram backprojection.

3.2 Tracking The Object

Histogram backprojecting is Unfortunately the function for this in OpenCV, though being on the OpenCV API, did not exist in our installation.

Therefore we tried a different, somewhat simpler method, called "Template Matching". This is a bit like histogram backprojection, but in stead of using histograms, we use a template of the object we wanted to see.

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

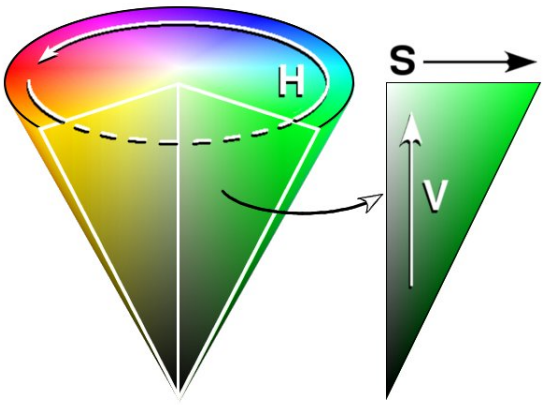

Figure 4: Hue (H), ranging from 0 to 360 degrees; Saturation (S), ranging from 0 to 100% and Value (V) ranging from 0 to 100%

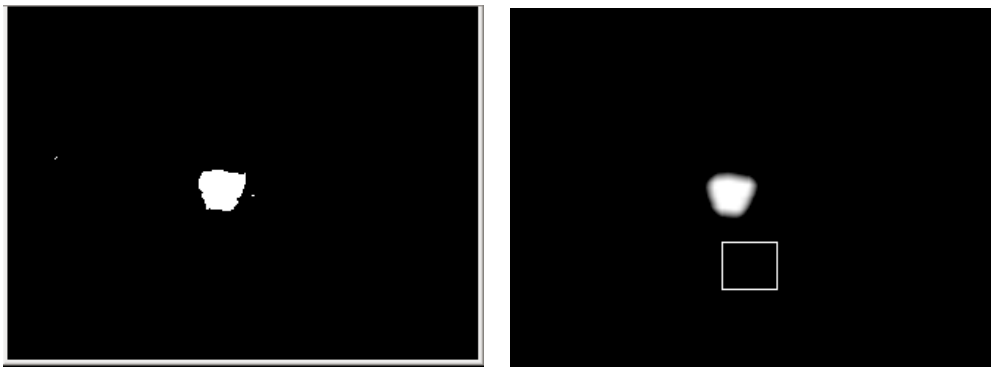

Figure 5: Left: An example of our image after the first processing. Right: After the second processing.

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), we made the template size variable, meaning that we always found exactly the centre of the object.

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. We did this by counting the number of non-zero pixels in the second processed window. We calibrated this by counting it for a number of known distances and then used a formula finder [2] to calculate a formula for us. This is the formula it came up with:

3.3.2 Variable steering commands

3.3.3 Blind flight

4 Results

5 Future Work

[2] This one