Autonomous Landing of a Quadcopter on a Predefined Marker

Robbert van Ginkel
10352600
robbert@vanginkels.com

Iris Meerman
10330984
irismeerman@hotmail.com

Timo Mulder
10207511
tmamulder@gmail.com

Jorn Peters
10334793
jornpeters@gmail.com

July 5, 2013

Abstract

This paper describes an autonomous landing procedure for the Parrot AR.Drone 2.0 quadcopter. An autonomous landing is defined as a landing with correct orientation on a predefined target marker without human intervention. The landing procedure should be initiated after first recognition of the specified landing location and untill then manual control is assumed.

Within the available time, the authors were unable to succesfully demonstrate the AR.Drone landing on the marker due to the circumstantial difficulties discussed in section 6. However, using the methods described in this paper they believe any quadcopter equipped with an altitude sensor and full-color downward facing camara should be able to execute an autonomous landing after recognition of a predefined marker.

1 Introduction

Commercial consumer focussed quadcopters are an uprising market. With capable unmanned aerial vehicles (UAVs) like the AR.Drone starting at $299 more and more of these devices find their way into consumers' homes. These devices are often accompanied with very friendly user interfaces to make the flying experience as smooth as possible. However, due to battery limitations flight times are often very short and more time is spent charging and assembling the device, than actually flying it. By designing software that, upon a user's request, can return the quadcopter to its docking station, land, connect, charge and prepare for a next flight, consumer quadcopters will be much less likely to end up in the attic after the owner became frustrated with the downside of his- or her flying dream.

The given problem can be decomposed into three single steps: reach landing site, execute landing manoeuvre and connect with the docking station. This paper focuses on performing the landing. Reaching the landing location is beyond the scope of this paper but could be achieved by using a quadcopter with GPS or running a Simulatanious Localization and Mapping (SLAM) (Durrant-Whyte and Bailey, 2006) algorithm from takeoff. In this paper manual control is used to direct the drone to the landing site. Docking with a landing platform will most likely be a vendor specific problem. This paper assumes landing with correct orientation on a predefined marker will be a sufficient replace-

ment.

The landing procedure can also be broken down into three steps: moving into horizontal position, orienting the direction the drone is facing and decreasing altitude while maintaining orientation and location until the surface has been reached. These three steps will be achieved by a recognition and an actuation module. The vision module will be presented with images from the drone, it recognizes the marker and passes its location to the actuator. The actuator then uses the location data of the drone and the information about the marker from the vision library to calculate the appropriate movement parameters (see figure 1).

By utilizing the Parrot AR.Drone 2.0 drone's downward facing camera and altitude sensor, it should be possible to create a representation of the current situation accurate enough initiate the landing procedure.

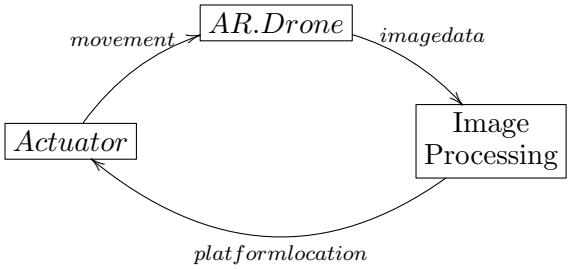

Figure 1: Process cycle

2 Materials

The quadcopter used to experiment is the Parrot AR.Drone 2.0. This quadcopter is equipped with multiple sensors and two cameras. The downward facing camera has a resolution of 640 by 360 pixels and runs at 60 fps, sufficient for detecting our predefined marker. It also contains the required altimeter. For a full specification, see appendix A.

The predefined marker used in this paper is a red 'H' symbol. The red colour should make sure the marker is distinguishable and the 'H' form will allow the quadcopter to align it's front with the symbol. Also 'H' is the universal symbol for a helicopter landing pad.

The communication with the quadrocopter is based on `node-ar-drone` (Geisendörfer, 2013), a Node.js based library with functions to control the drone and receive its video stream.

The vision module is based upon OpenCV (Bradski, 2000) and written entirely in C++.

3 Methods

The implementation proposed in this paper is built as a `ardrone-webflight` (Eschenauer, 2013) plugin. Ardrone-webflight is an implementation on top of node-ar-drone that allows for user and automated piloting of the drone while streaming video and running custom made plugins.

The basic contol flow of our 'landingPlatform' plugin is as follows: listen for new data from the videostream, grab and save that as png, run marker recognition, transform coordinates into movement, send movement correction to drone, re-enable listening for new frames. This workflow saves and analyzes 8-10 frames per second. As ardrone-webflight also hosts a webinterface, it also uses HTML5 canvas technology to draw the location of the found marker over the received videostream.

3.1 Vision

The vision module of the program is responsible for detecting the landing position (see figure 2) in a still image taken with the downward facing camera of the quadcopter. The problem of detecting the landing position is seperated

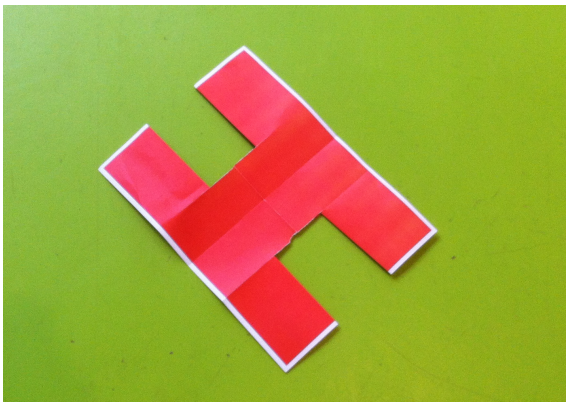

Figure 2: Target marker for landing

in four different steps, namely, colour segmentation, contour detection, contour selection and candidate validation. These steps are described below, the effectivity of the combination of these steps is discussed in section 4.

3.1.1 Colour Segmentation

The first step in the process is that of colour segmentation. The target is red, therefore any object in the photo that is not red can be excluded from the set of potential targets. For this, a thresholded or binary image is created. In this image all objects/areas in the picture that are predominantly red are converted to white areas. Everything else is converted to black. To achieve this, the pixel values are converted from RGB (red, green, blue) to HSV. The HSV colour space is a different way to express colours in which a colour is expressed in hue (the pure color expressed in degrees between 0° and 360°), saturation (the amount of white that is 'mixed' with the pure colour) and value (the amount of black that is 'mixed' with the pure colour). Figure 3 displays a visual representation of the HSV colour space.

After converting the colour data to HSV, a thresholded image can easily be generated by converting all pixels that have a hue value between a upper and lower bound to white and all other pixels to black. The result of this can be seen in figure 5. An upper and lower bound can also be set for saturation and value, this may make the filter less sensitive to different lightning conditions and improve the overall result.

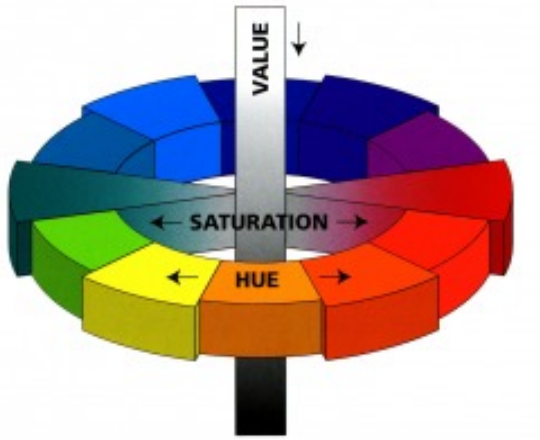

Figure 3: HSV colour space

(taken from: www.profstark.com)

3.1.2 Contour Detection

To find the target in the image, contours are generated of all red areas in the image. A contour is a set of (x, y) coordinates in the image that marks the edge of an area. A contour consists of $c_1, c_2, \ldots, c_n$ coordinates for $n \geq 2$. Every coordinate c_i is connected with a straight line to coordinate c_{i+1} and c_n is connected to c_1. The contours in the image are detected using the border following algorithm as described by Suzuki et al. (1985). This algorithm takes a binary image as input and finds all borders. The algorithm makes a distinction between outer borders and hole borders, with that the contours are created[1].

[1]In the authors implementation the OpenCV implementation of the border following algorithm was used.

3.1.3 Contour Selection

After all the contours in the image have been found, the contour that best matches the shape of the target marker is selected. For this the contour of the target marker is created as described in the previous section using a model image of the target marker. All contours found in the image are compared to this model contour and the best match is selected. To calculate the match between two contours the seven Hu moment invariants are calculated (Hu, 1962). These seven moments are (weighted) averages of the binary image and are invariant to scaling, rotation and translation. An example of a Hu moment invariant is the area of the shape. This makes the Hu moment invariants an easy way to compare two contours. A 'matching' metric is calculated as follows (OpenCV, 2013)

$$\text{Match}(A, B) = \sum_{i=1...7} |m_i^A - m_i^B|$$

This results in a metric that approaches 0 when the shapes are similar to each other. The contour that results in the lowest match metric is the contour that best matches the model contour. A maximum match score is set that ensures that only legitimate matches are returned. If only matches with a high matching score are found it is assumed that there is no valid match in the image.

When there are matches lower than the maximum matching score, the contour with the lowest matching score is selected as the candidate contour.

3.1.4 Candidate Validation

When there is a red rectangle in de picture but no target marker, the contour detection often (falsely) selects the red rectange as the target marker. The contours of the target marker and a rectangle are to similar when compared using the described method. Therefore an extra validation step is performed on the candidate contour. To filter the false contours the convex hull of the candidate contour is computed. The convex hull of a contour can be seen as the shape that a rubber band would take when it is streched around all the points in a contour. An example of a convex hull is given in figure 8.

It is expected that the convex hull of a rectangle and the rectangle self are similar and therefore score a low score when compared using the method described in the previous section.Based on this assumption the convex hull of the candidate is compared to the candidate. If the matching score is below a specified threshold (the convex hull and contour are similar) then the contour is rejected and the conclusion is drawn that there is no matching contour.

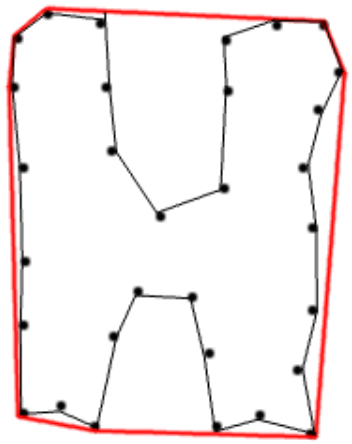

Figure 4: Convex hull (red) of contour (black)

3.1.5 Output

If a contour is selected the center point of that contour in the image measured from the center of the image is computed. Also the radius of the

smallest circle that circumfences the contour and the angle under which the contour is. Because the target marker used is symmetrical the angle is within the range $[-90, 90]$. The sign hints the actuation module in which direction the quadcopter should turn to reach the final position above the marker with the least effort.

3.2 Actuation

After the image has been processed the actuation module, which is implemented as a movement function in our 'landingPlatform' plugin, processes its results and generates the according movement. If the vision module is unable to recognize a marker, the drone is put in hover mode and turned to manual control.

The node-ar-drone movement interface allows for setting rotorspeed for movements such as forward, backward, left, right, clockwise rotation, counter clockwise rotation, up, down. These values can be set individually, after which the control software calculates the resulting speed for all four rotors to send it to the drone. When the drone receives this movement command, it will maintain these speeds for each of its engines until another speed command or the hover command is received.

Movement is not directly controlled by the output of a single frame from the vision module. Instead, a PID (proportional-integral-derivative) controller is used to take previous movement actions into account. A PID controller is based on a mathematical algorithm and calculates an 'error' between a measured value and a target value. The output of a PID controller is the sum of the weighted constants (P, I and D) applied to the input value. Each output of the vision module is routed through the appropriate PID controller (x-position, y-position, angle of rotation), the PID controller then outputs a value to send to the drone.

The horizontal information received from the vision module are the x- and y offsets of the center of the marker to the center of the drone's downward camera. The x-amount of movement and the y-amount of movement are initialized by updating the PID-controllers with the negation of the OpenCV data to correct for the drone's inverse movement commands. The resulting values from the PID-controllers are then scaled to rotor speeds between 0 and 1.

If the resulting rotor speeds are insignificant, this means the drone is hovering above the center of the marker. To achieve a correct orientation, the angle of rotation from the vision module is processed by a PID and scaled like in the xy movement process and a according clockwise or counter clockwise rotation speed is set.

If the rotation speed is also insignificant, the remaining step is to decrease altitude until reaching ground. The default 'land' command of the drone is too inaccurate to achieve the desirable landing, because it just decreases altitude without the requirement of staying in the same position. This makes it prone to errors introduced by previous momentum or wind. Instead of directly using the 'land' command we introduce a downward command in the run loop until a reference height is achieved. This moves the drone closer to the target while still being able to adjust its x, y and angle with reference to the marker. When the reference height is reached (a height from which the default landing can be executed without compromising the achieved location), the default landing procedure is initiated to put the drone on the ground.

4 Results

The project is divided into two different components, namely actuation and vision. Therefore the results will also be discussed separately.

4.1 Vision

As described in section 3 the module for visual recognition uses a combination of techniques. The techniques are disccused incrementally. That is, the techniques fit together like a puzzle and build on the results of previous techniques.

4.1.1 Colour Segmentation

The results of the colour segmentation using the HSV colour space to detect red objects in an image are shown in figure 5. In this figure a tresholded image is created from a photo that contains the red target marker on a green background. The image clearly shows that the red objects are detected. The authors expected that detecting the red colours would pose problems as red is at both ends of the hue wheel of the HSV colour space. However, to detect the red colour marker only the reds in the upper ranges of the colour wheel need to be detected, that is $240° - 360°$. Manipulation the range of the saturation of value that pixels did not result in a better detection of red areas so no further conditions for the selection of pixels where added.

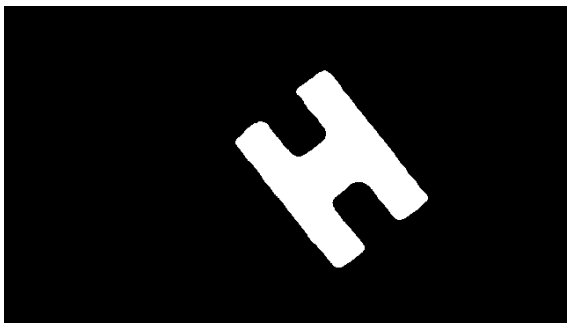

Figure 5: Tresholded image

4.1.2 Contour Detection

The border detection algorithm works as expected and detects all borders in the tresholded image. In figure 6 an example of a detected contour is shown. The contour is drawn in blue on top of the target marker. After computing the contours there is a is a good representation of all (red) shapes in the original image.

Figure 6: Contour detection

When the target marker is in the original photo the contour is detected in nearly all cases, however the border detection algorithm uses the thresholded image as input and therefore is only as good as the generated thresholded image. An example of this is included in figure 7. In this figure two objectes in the original photo are combined into a single blob in the thresholded image. The contour detection algorithm then wrongly detects the combined blob as a single contour. As said this is a problem in the tresholded image generation and not of the border detection algorithm.

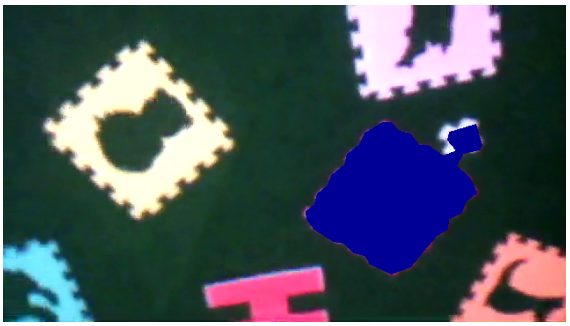

Figure 7: Mismatch contour

4.1.3 Contour Selection

The difference in the Hu moment invariants of two contours is a clear and concise metric to express similarity of two contours. When a target marker is present in the photo and a contour is correctly generated it is always detected as the best matching contour. However, when no target marker is present in the photo but another red rectangle shaped object is, it is often recognized as the target marker with a matching score comparable to the matching score of the real target. This is only observed with red rectangle shaped objects, red objects of any other form are not detected as the target marker. By requiring the candidate contour to have a matching score below a predefined maximum score it is ensured that objects that are unsimilar to the target marker are rejected.

4.1.4 Candidate Validation

As discussed in the previous section, reactangle shaped objects are falsely detection as target markers when there is no target marker present. The similarity of the convex hull of the contour and the contour is a good metric to determine if the contour is rectangle shaped. When the contour is rectangle shaped the similarity score between the contour and the convex hull is low (it approaches zero). The similarity of the convex hull of a contour of the target marker and the contour itself results in a significantly higher matching score. When the similarity score is low the candidate contour can safely be rejected and no target marker is found in the original photo. An example of the computed convex hull is shown in figure 8

4.1.5 Output

Besides these techniques which are applied to find the right landingspot when visible, it was also of importance to find the center of the landing marker and the angle for landing. By detecting the smallest bounding circle, the centre point could easily be drawn. See figure 9. This figure also shows that the right red object is found and the contour.

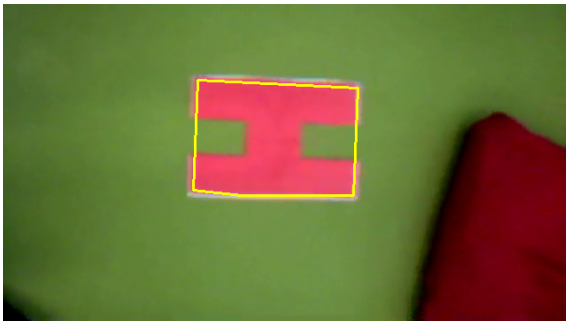

Figure 8: Convex hull

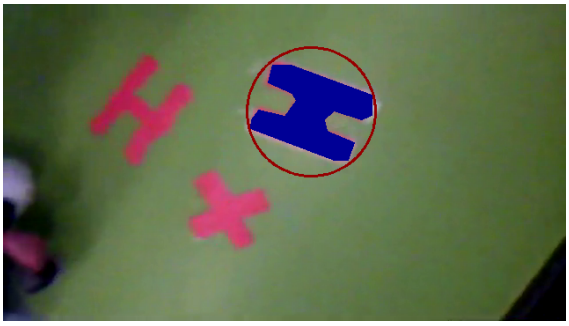

Figure 9: Bounding circle

The angle was calculated by drawing the bounding box which could also be rotated. Drawing this box gave the following result, see figure 10.

4.2 Actuation

The results of the actuation module were not as expected. The problem was that the AR.Drone began to give more and more unexplainable bugs during the projectweek. A minor problem was that after a crash landing, the AR.Drone could still be processing some packets from a previous

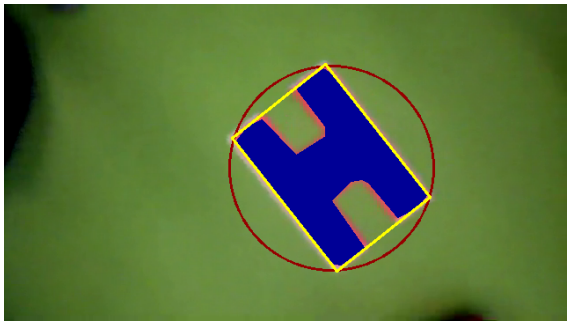

Figure 10: Bounding box

session. When it was reset, it sometimes randomly started to repeat the last command it got. A major problem was that when the actual landing code was implemented, the AR.Drone even did not work as expected while using the standard iOS application (Parrot, 2013). So the only result as far as there could be tested, was that the AR.Drone flew perfectly above the marker, making small turns to get the marker in the right angle.

5 Conclusions

This research focused on the autonomous landing of quadcopter. This task was divided into two subproblems: vision and actuation. The vision subproblem is examined with various techniques for colour segmentation, contour detection and shape matching. The combination of techniques resulted in a satisfying program capable of detecting the target marker in various scenarios. The actuation subproblem is examined with a PID controller that enables the program to keep a tight feedback loop on the movements of the quadcopter.

Based on instructions generated by the program the authors believe that a quadcopter should be able to autonomously land on a predefined target marker using the techniques proposed in this paper. However, as mentioned before, due to technical difficulties and limited time no actual autonomous landing was observed.

6 Discussion

Possibilities for future work could be perfecting the landing marker recognition part of the program, although the authors of this paper are quite positive about what is reached on this subject during four days. The actuation part though was less successful. The authors were quite surprised about the troubles that came with the AR.Drone, because most actuation problems were due to this device. The very last moment of the very last day they had been told that the AR.Drone could cause some problems if the battery was running on a specific level. Such hardware problems are very hard to fix and made it impossible to let the AR.Drone fly properly. An option for better investigation would be using an other device without these bugs.

References

Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

Durrant-Whyte, H. and Bailey, T. (2006). Simultaneous localization and mapping: part i. *Robotics & Automation Magazine, IEEE*, 13(2):99–110.

Eschenauer, L. (2013). ardrone-webflight. `https://github.com/eschnou/ardrone-webflight`. Online; accessed June, 24 2013.

Geisendörfer, F. (2013). node-ar-drone. `https://github.com/felixge/node-ar-drone`. Online; accessed June, 24 2013.

Hu, M.-K. (1962). Visual pattern recognition by

moment invariants. *Information Theory, IRE Transactions on*, 8(2):179–187.

OpenCV (2013). matchShapes Documentation. `http://opencv.willowgarage.com/documentation/cpp/imgproc_structural_analysis_and_shape_descriptors.html#matchShapes`. Online; accessed June, 26 2013.

Parrot (2013). Freeflight for iphone. `https://itunes.apple.com/us/app/free-flight/id373065271?mt=8`. Version 2.4.2.

Suzuki, S. et al. (1985). Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30(1):32–46.

Appendices

A Specifications Parrot AR.Drone 2.0

A.1 Video

The AR.Drone 2.0 is equipped with two camera's.

- HD Camera. 720p 30fps
- Wide angle lens : 92 degrees diagonal
- H264 encoding base profile
- Low latency streaming
- Video storage on the fly with the remote device
- JPEG photo
- Video storage on the fly with Wi-Fi directly on your remote device or on a USB key

A.2 Structure

The quad has several features for protecting himself for damage.

- Carbon fiber tubes : Total weight 380g with outdoor hull, 420g with indoor hull
- High grade 30%
- fiber charged nylon plastic parts
- Foam to isolate the inertial center from the engines' vibration
- EPP hull injected by a sintered metal mold
- Liquid Repellent Nano-Coating on ultrasound sensors
- Fully reparable: All parts and instructions for repairing available on the internet

A.3 Electronic assistance

AR.Drone 2.0 on-board technology contains automatic stabilization features.

- 1GHz 32 bit ARM Cortex A8 processor with 800MHz video DSP TMS320DMC64x
- Linux 2.6.32
- 1Gbit DDR2 RAM at 200MHz

- USB 2.0 high speed for extensions
- Wi-Fi b,g,n
- 3 axis gyroscope 2000 degrees/second precision
- 3 axis accelerometer +-50mg precision
- 3 axis magnetometer 6 degrees precision
- Pressure sensor +/- 10 Pa precision
- Ultrasound sensors for ground altitude measurement
- 60 fps vertical QVGA camera for ground speed measurement

A.4 Motors

The quadcopter is equipped with four propellors to keep him stable and flexible.

- 4 brushless inrunner motors. 14.5W 28,500 RMP
- Micro ball bearing
- Low noise Nylatron gears for 1/8.75 propeller reductor
- Tempered steel propeller shaft
- Self-lubrificating bronze bearing
- Specific high propelled drag for great maneuverability
- 8 MIPS AVR CPU per motor controller
- 3 elements 1000 mA/H LiPo rechargeable battery (Autonomy: 12 minutes)
- Emergency stop controlled by software
- Fully reprogrammable motor controller
- Water resistant motor's electronic controller

B Software Specifications

B.1 Quadcopter interaction

1. Node.js v0.10.12
2. Node Packaged Modules (NPM) 1.2.32
3. Bower (0.9.2)

4. node-ar-drone (commit 9307871755)
5. ardrone-webflight (commit efc3033d7a)