Beacon Detection and Tracking

with the A.R. Drone 2.0

November 6, 2016

Saffat Shams Akanda, z5061498

Jonas Dammen, z5127817

Benjamin Faul, z3422539

David Michael Hindmarsh, z3459407

Ben Soh, z5059295

University of New South Wales

**Abstract**

Unmanned Aerial Vehicles (UAVs) are becoming a popular option for a number of common commercial, civilian and military applications, including remote sensing, situation monitoring and tracking. This paper presents a system where a UAV is used to track ground targets using a ground facing camera. The experiments performed were able to successfully track a single ground marker using circle and line detection methods.

Table of Contents

[1. Introduction 1](#_Toc466224240)

[2. Background Information 1](#_Toc466224241)

[3. Methodology 2](#_Toc466224242)

[3.1. Image Processing 2](#_Toc466224243)

[3.1.1. Method 1 - Edge Based Detection Using OpenCV 2](#_Toc466224244)

[3.1.2. Method 2 - Colour Based Detection Using OpenCV 2](#_Toc466224245)

[3.2. Controlling the Drone 4](#_Toc466224246)

[4. Evaluation 5](#_Toc466224247)

[5. Conclusion 6](#_Toc466224248)

[6. References 7](#_Toc466224249)

[Appendix A: Steps on how to operate/run the drone 7](#_Toc466224250)

# Introduction

Today, UAVs are the most widely used platform for a number of applications including remote sensing and tracking [1]. Ground tracking is an important task in both commercial and military applications which UAVs are increasingly used for [2]. Quadcopters in particular are widely used due to their superior flight and control qualities, including simple mechanical design, high manoeuvrability and agility as well as fine control and stability. In this paper the Parrot A.R. Drone 2.0 is used to track a ground target using the quadcopter’s built in ground facing camera. This work aims to demonstrate the advantages of using a quadcopter for ground tracking targets by using common image processing algorithms and readily available off-the-shelf hardware and software.

The following sections of this paper are laid as follows: Section 2 provides further background into the use of UAVs, and more specifically for ground target tracking, Section 3 provides the methods used for image processing and flight control, Section 4 explains the results of the demonstrations and section 5 gives a conclusion and proposes areas for future work.

# Background Information

The ground target that was used to track was taken from the Robocup Small Sized League. This was decided since it provided the option to have multiple unique beacons on the ground that could potentially be tracked independently. The other more important reason for using this was to allow the use of the SSL Vision software provided to teams in the league. Upon acquiring this software, it was clear that it was not written in ROS. There were some initial attempts in investigating ways we would transfer the software into a ROS node, however it was quickly decided that this would take too much time to implement.

To control the drone, a tutorial by Mike Hamer on robohub was followed to assist in getting the drone operational [3]. The only file that was modified from the ardrone\_tutorials git repository was the file originally called ‘keyboard\_controller.py’. The changes made to this file are outlined in Section 3.2.

# Methodology

We split the project into two simpler sections, those being the processing of the images and the other the corresponding movement of the drone.

## Image Processing

There were many different ways that this problem could have been approached from. The two methods that were investigated and implemented were an edge based detection, and a colour based detection, which are discussed in further detail below.

### Method 1 - Edge Based Detection Using OpenCV

This method relies on filtering out colours in the image to detect beacons using OpenCV. It takes the sample image and applies a filter to remove all non-black cells and does a count, if the number of cells is greater than a threshold it is assumed that a beacon is in the image.

A blue filter is then applied and uses a Hough Circles algorithm to detect any circles in the image, since a beacon should only have one blue circle we mark this circle’s centre as the location of the beacon. We then proceed to apply a mask of 2 \* Radius of Circle to filter out all information other than the beacon itself. If no circle is detected (a false positive was detected) we stop processing the current image and send a ROS message saying that we cannot find the beacon currently.

Next a black filter is applied and Hough Lines is used to detect all lines in the image then the lines closest to the edge of the mask are chosen as these are most likely to be the edge of the beacon. We then apply a tangent function to the location of the two points to determine the angle of the line.

### Method 2 - Colour Based Detection Using OpenCV

A more functional implementation of image processing was also developed in Python with the OpenCV library. This implementation identified the ID of the RoboCup pattern from any angle as well as the direction it was facing, allowing the drone to adjust its direction correspondingly. This implementation takes advantage of the fact that all the possible patterns were enclosed by a large, easily detectable, black circle.

A Circle Hough Transform algorithm was applied to the BGR image to filter out unnecessary pixels. The largest detected circle was selected since this process would pick up the smaller coloured circles as well, and the rest of the frame was cropped out to avoid future false positives. On rare occasions, a different circle was detected and selected, but these were filtered out later.

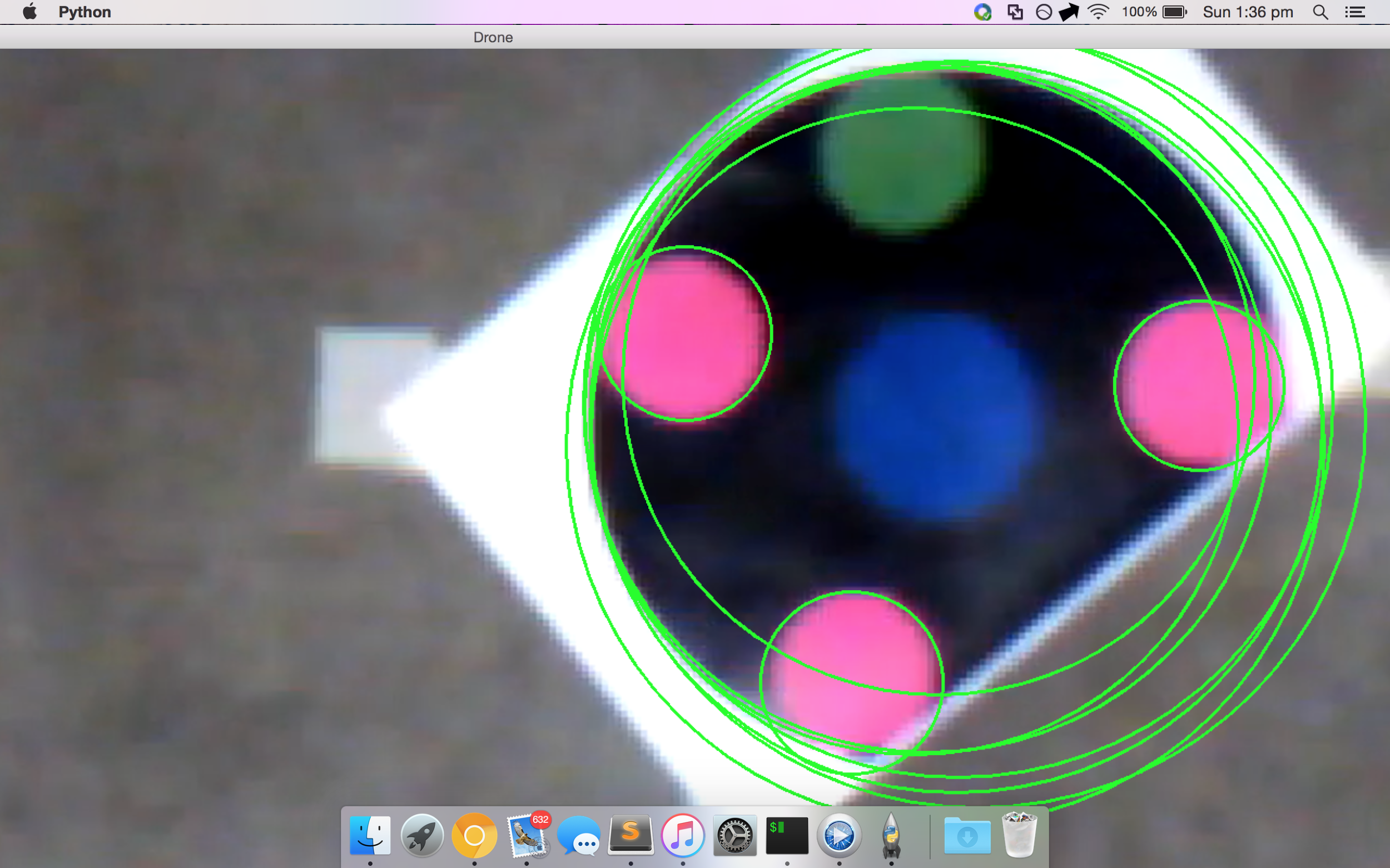


Figure : The circles detected by the first few steps

Using the boundary HSV values for the magenta and green colours, the cropped circle were thresholded to get only those colours. An opening then closing morphological transform is applied to the mask in order to smoothen out the edges. As a final step, the mask is applied to the original circle image to obtain the 4 smaller coloured circles.

Each circle is converted into an object with x, y, and colour attributes and stored into a list. If the number of coloured circles detected differs from 4, the pattern detection will fail at this stage. The centroid of the four circles is also calculated here and is stored as the coordinate of the pattern on the screen.

To calculate the angle, the distance from each circle object to every other circle is calculated and selects the minimum distance to be between the two bottom circles. Since the bottom facing camera is used, the view in non-skewed, so this will always be true. A line is drawn between these two circles and the angle this line makes with the x-axis is calculated if the beacon is the opposite way round, then the angle will be off by 180 degree. To counter this, the y-values of the circles are compared, and if the bottom circles are above the other circles, then the calculated angle is adjusted by 180 degrees.

The final stage is to identify the pattern ID. Since there is a limited number of RoboCup pattern, each pattern was hard coded and checked with the detected pattern to find a match.

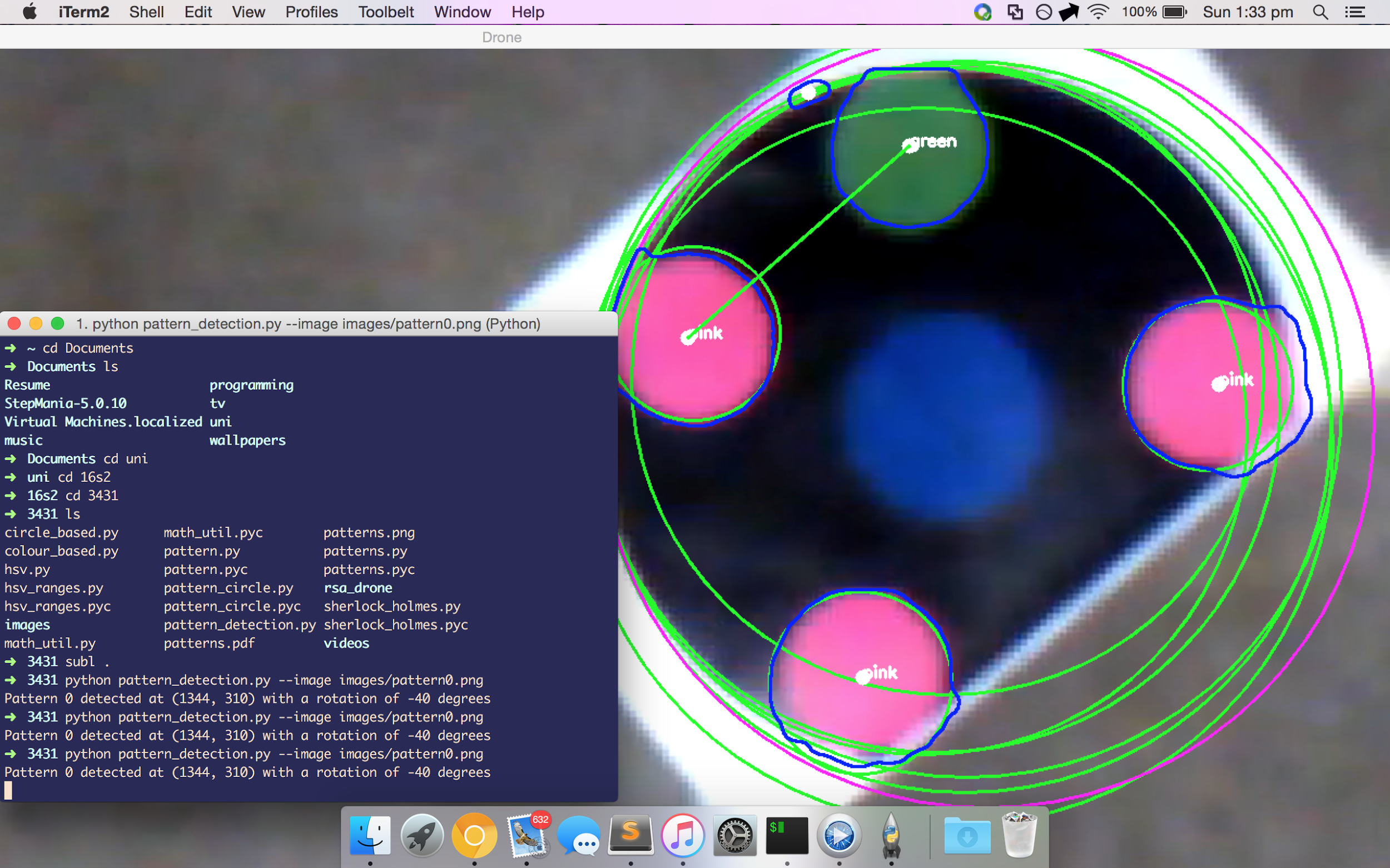


Figure : The resultant image detection

This method proved to be very accurate at detecting pattern IDs. A slight issue which occurred was that sometimes, in not-perfect frames, the black circle was not detected. In hindsight, another approach which could be taken to fix this issue would be to skip detecting the black circle since it is unnecessary if the colours we are dealing with (magenta, green) will not appear in the surrounding environment (like they did not for this task).

This method was not used because pattern ID and angle detection were not required for this task so the simpler and faster edge detection-based method was preferred.

## Controlling the Drone

Once the drone was operational, the file ‘keyboard\_controller.py’ was extended to allow the program to enter (and exit) autonomous flight, and renamed to ‘autonome\_controller.py’. This file initially just allowed commands to be entered to control the drone. A class, drone\_controller, was constructed to act as an interface between the drone and the keyboard controller, and was also where the main part of the autonomous controlling was performed.

Three values would be passed from image processing to the drone\_controller class corresponding to pixel coordinates of the centre dot (x and y), and the orientation of the beacon to the drone. From these coordinates we determined the speed at which the drone should move, and in which direction, by using the following diagram

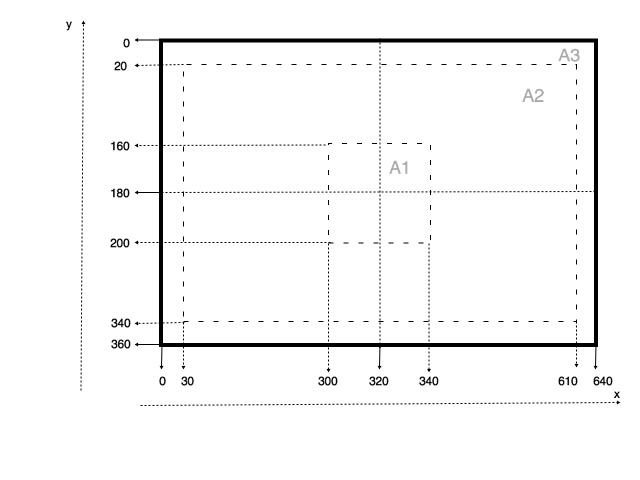


Figure : A diagram showing the separate areas of the image

If the coordinates corresponded to a location inside the inner box then the drone would hold its position. Between the two boxes, the speed would increase linearly from 0 to ±1, and would saturate at ±1. This was implemented in order to get the drone to start centring the beacon early and allow it to make small movements as well as larger movements if the beacon moves rapidly.

Another feature added was a small memory bank that would remember the previous 20 locations the beacon had been seen. Once the drone lost sight of the beacon it would hover in place for 40 frames to ensure the beacon had indeed been lost and would move for 20 frames towards the last known direction. If the beacon doesn’t come into view in this time then the drone will land. This was set as a safety precaution since we were flying in an enclosed space.

# Evaluation

During the first demonstration the drone operated in a satisfactory manner. It was able to track a fast moving beacon extremely well, and a slow moving beacon still relatively well, however it would at times lose track of the beacon even when the beacon was not moving. There were also times when the drone would not fly in the correct direction; however this was very difficult to reproduce. This demo was done before we implemented the different regions of tracking with the gradual increase in speed (see Figure 3) and attempting to reacquire the beacon. As a result, we were given an extra week to improve the tracking of the beacon, and develop the beacon search in a very small scale for safety reasons.

While attempting to further test the tracking of the beacon, as mentioned before, the drone would fly off in the wrong direction and would not be able to reacquire the beacon. We finally found the cause of the problem. When sending movement commands to the drone, a twist message is sent, which contains linear and angular x, y and z values. We had assumed that linear x and linear y corresponded to the standard x and y of the image. However, for the drone this is reversed. This is because sending a linear x command (movement only in linear x) would make the drone rotate around its x-axis causing it to move in the y direction, and similarly for the linear y command. Once this issue was identified and with the additional tracking features implemented, the tracking of the beacon improved dramatically.

The drone controlled itself in a great manner during the final demonstration. It was put through 4 subtests in order to test its autonomous control. The first test involved the drone hovering over a stationary beacon to test its stability. The second test involved moving the beacon at a slow pace in different direction. The third and fourth tests involved moving the beacon at increasing paces, and try to get the drone to lose track of the beacon to show the drone attempting to re-find it.

The drone handled the first two tests very well. At one instance during the test, the drone suddenly flew off in the wrong direction. It was reported that the video to the computer had frozen, and it was concluded that there must have been some sort of error in the sending of data to and from the drone which caused this issue.

When testing the tracking by moving the beacon rapidly, it performed very well. The most difficult part was trying to get the drone to lose sight of the beacon to test the finding feature. When the drone did finally lose sight of the beacon, the drone did start moving in the direction of the beacon, however due to safety concerns, the limit we had imposed on this feature was too great and it decided the beacon was lost to early.

The other thing that was noticed during both demonstrations was that the height of the drone would vary a lot. Since we were only sending commands to rotate around its x and y axis, this should not have been happening. This was likely caused by turbulence from its own motors since the drone was being operated in an enclosed space.

Three videos were taken of the drone during the demonstration and can be found at the following links.

[Demo Video 1](https://drive.google.com/open?id=0BzC9wSPsnjtGSEx3WVphZXdPT1U), [Demo Video 2](https://drive.google.com/open?id=0BzC9wSPsnjtGRkNNSUhNZ1VJZ3c), [Demo Video 3](https://drive.google.com/open?id=0BzC9wSPsnjtGWkRwRGxVY0l4NVk)

# Conclusion

While the drone was successfully able to complete most of the tests it was given, there are always improvements that can be made. There were also a number of limitations in the project. The openCV method of image processing was not tested with multiple beacons, and while it could find all beacons in the current image, it would not be able to determine which beacon it was meant to follow. There is always the trade-off between complexity, efficiency and the time available to implement it, and the method we implemented we believe was appropriate. Improvements to this detection by using sift or surf using contours to be able to track the same beacon, along with a method to keep the drone at a particular height would assist in both the stability and accuracy of the beacon detection.

Improvements to finding the beacon once it has been lost in the current setting would be difficult since only one camera can be activated at once and so there is no way of knowing if the drone is about to collide with something in front of it.

# References

1. “The Use of UAV Platforms for Remote Sensing Applications: Case Studies in Cyprus”
2. “Ground Target Tracking Using UAV with Input Constraints”
3. M. Hamer, "Up and flying with the AR.Drone and ROS: Getting started | Robohub", *Robohub.org*, 2016. [Online]. Available: http://robohub.org/up-and-flying-with-the-ar-drone-and-ros-getting-started/.
4. M. Hamer, "mikehamer/ardrone\_tutorials", *GitHub*, 2016. [Online]. Available: https://github.com/mikehamer/ardrone\_tutorials.

# Appendix A: Steps on how to operate/run the drone

Make sure you have gotten the ardrone autonomy and AR Drone Tutorials packages before attempting to run the following steps.

<https://github.com/AutonomyLab/ardrone_autonomy>

<https://github.com/mikehamer/ardrone_tutorials>

1. Turn on drone, ~10 seconds after it rotates its blades it should be fully on.
2. It will now create a Wi-Fi network that starts with “ardrone2”, connect to it
3. Enter the following commands:

roscore

rosrun ardrone\_autonomy ardrone\_driver

rosservice call /ardrone/togglecam

roslaunch drone bottomBeaconDetector.launch

rosrun drone autonom\_controller.py

1. Click on the camera feed of the drone to control it via keyboard:

Move Forward (E)

Move Backward (D)

Move Left (S)

Move Right (F)

Rotate Left (W)

Rotate Right (R)

Increase Altitude (Q)

Decrease Altitude (A)

Takeoff (Y)

Land (H)

Emergency Stop (Space Bar)

Go Autonomous (Z)

Manual Control (X)

1. Drone will start in manual control mode, fly over beacon placed on ground
2. After drone is ~1.5-2m above beacon press Z to engage autonomous flight and track the beacon; pressing X at any point will return control to you
3. If the drone crashes and enters emergency mode, you can start flight again (after safely moving to an appropriate launch area) either by disengaging Emergency by pressing Space Bar again or restarting the drone