Alpha-Beta Drone Model Design Analysis

Bishop Clark

University of South Florida

# Introduction

These series of programs are used to model an alpha-beta system similar in animals like bird flocks, wolf packs, schools of fish, and even ants. It involves a lead member (alpha) followed by one or more other members (beta). The project uses exclusive scripts per drone along with the ROS platform and Python programming language to model these groups. The following discusses a complete overview of the drone properties, installing Ubuntu, initializing Robot Operating System (ROS) framework in host computer, alpha-beta code used to autonomously control multiple drones, configuring and running the code, complication-solution procedure, and external resources.

## Objective

To design a collaborative navigation algorithm that accurately models an alpha-beta system using autonomous drones.

## Drone Properties

Each AR Drone 2.0 (drone) is quadcopter in that it is propelled through the air using four rotors, one for each propeller. Materials included with the purchase of the drone are the battery, the wall charger, and an outer propeller hull, which, power the drone, charge the battery, and protect the propellers from damage, respectively. The drone also transmits is own personal Wi-Fi access point allowing Wi-Fi capable devices like phones and computers to connect to it. In order to begin using the drone:

1. Connect the charged battery to the drone and latch it safely inside with the Velcro
2. Four lights, one beneath each propeller should light red then green.
3. Equip the hull around the drone

If in any case the drone propellers become damaged and the drone is unable to fly, additional propellers may be purchased from Parrot on Amazon or the Parrot website. The propellers must be reattached to the drone, please follow the user manual regarding equipping new propellers.

## Ubuntu Xenial (16.04)

The framework (ROS) used to control the drone requires a Linux Ubuntu device for installation. Refer to the following website for a method to dual boot this operating system alongside your existing one: <https://tutorials.ubuntu.com/tutorial/tutorial-install-ubuntu-desktop-1604#0>

Keep in mind when creating a bootable USB, the USB must be formatted. On Windows a USB formatting software is Rufus, on Mac, it is balenaEtcher.

## ROS Framework Configuration

ROS is a set of libraries offered for the development of robotic systems. The drone can be controlled through this framework installed on an Ubuntu Linux device. This walkthrough explains how to configure the environment on Ubuntu for the AR Drone 2.0 using the ROS software. It is assumed at this point that the reader has the necessary amount of operable Ubuntu Linux devices with ROS installed properly on them.

Visit the following website and proceed through steps, copying the commands, to successfully install Kinetic on Ubuntu:

* ROS Kinetic version (wiki.ros.org)
* If not already in the bash file, add the line “source /opt/ros/kinetic/setup.bash” to enable ROS commands within the terminal; you can open this file by inputting “gedit /.bashrc”
* Clone the repository to the workspace created during the ROS installation procedure: Files (https://github.com/biorobaw/ardrone.git)

## Alpha-Beta Code

The alpha-beta program follows a similar procedure to that of bug algorithms in that it: takes off, searches for the objective (goal/color for each drone), moves toward the objective, and waits for further instruction. If the objective decides to move, the beta drone will move toward the objective. If the objective is motionless, so too is the beta drone if it is within a certain distance. Motion control used the ROS platforms’ preexisting linear and angular motion functions. This program heavily utilizes camera vision for object recognition. Each drone will follow the preceding drone based on the color specified in its code. OpenCV is the software used to process the image captured on the drone. To install OpenCV perform the following:

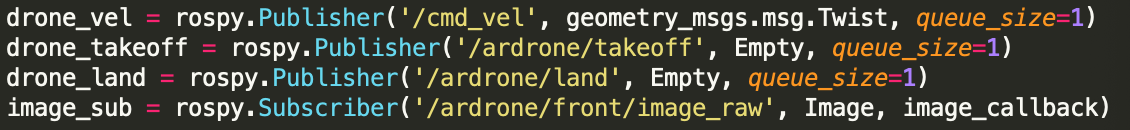
* Open terminal
* Enter “sudo apt-get update” command
* Enter “sudo apt-get install ros-kinetic-opencv3

To test if it was installed properly, open a python shell and enter “import cv2”. If any errors appear, refer to installation on the ROS or OpenCV websites.

This section discusses the program organized by the required imported libraries then data flow in the main function.



The node must be created in order to publish commands to the drone such as: movement, taking off, landing, camera access, etc. The node can be named anything, in this case the name is ‘hover’. The following figure initializes the drones ability to traverse, take off, land, and use the front camera respectively.

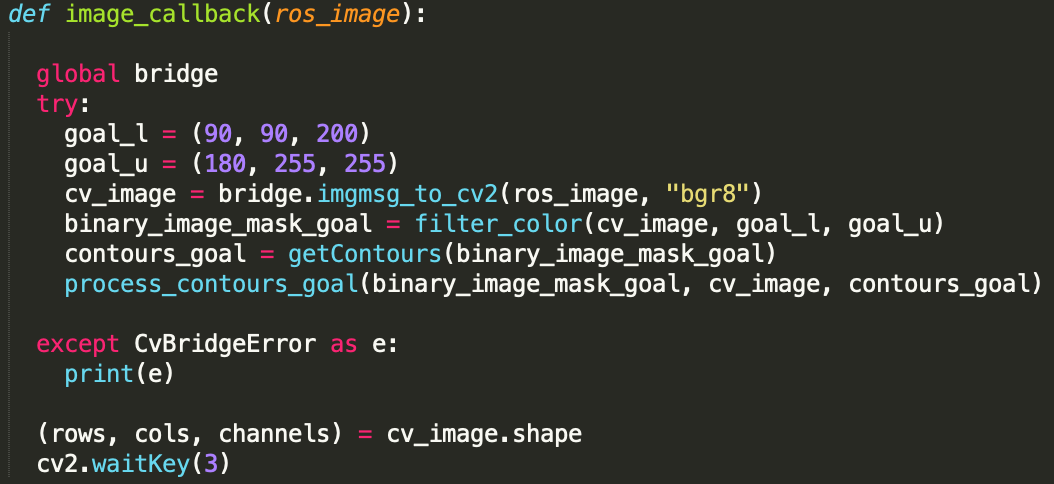


When these objects are initialized the drone is able to perform actions. The first action is for the drone to take off and hover in the air:



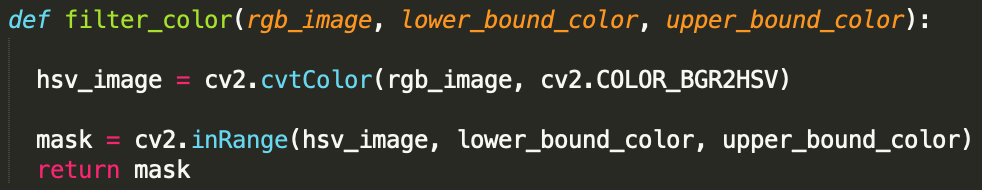
It is published using an Empty() object parameter. Notice in the initialization of ‘drone\_takeoff’ the Empty object is the parameter of the Publisher function. The succeeding explains the feedback loop involving the camera vision and motion aspect.

The camera and OpenCV software allow the drone to transfer images to the host machine (Ubuntu) and the host will process these images. Depending on the condition, it will command the drone to perform certain actions. Because each drone is run by separate computers, each drone also has an exclusive script. A drone should follow the drone in front of it, not two drones in front. This requires each drone to follow a specified color so that no beta drone will follow the incorrect alpha drone.

The following figure is an image call back function that allows the color specified to be processed and contours to hone on. The lists are the upper and lower HSV values of pink and green used in experimentation. The lines beneath create an OpenCV image on the host and creates binary images that designate something to be or not to be the object. It will then process the contours of its specified color. It sends information like the area and perimeter of the contour, the x and y values of the contour on the image, it can even tell you how many contours are in the image. A contour is a designation on an image mask that lets the host understand what the main object of the image is. In this case the contours of each mask are any specific color the drone has specified.

The ’imgmsg\_to\_cv2’ is a function provided by the cv2 library that bridges the images captured on the drone and transmits them to the host.

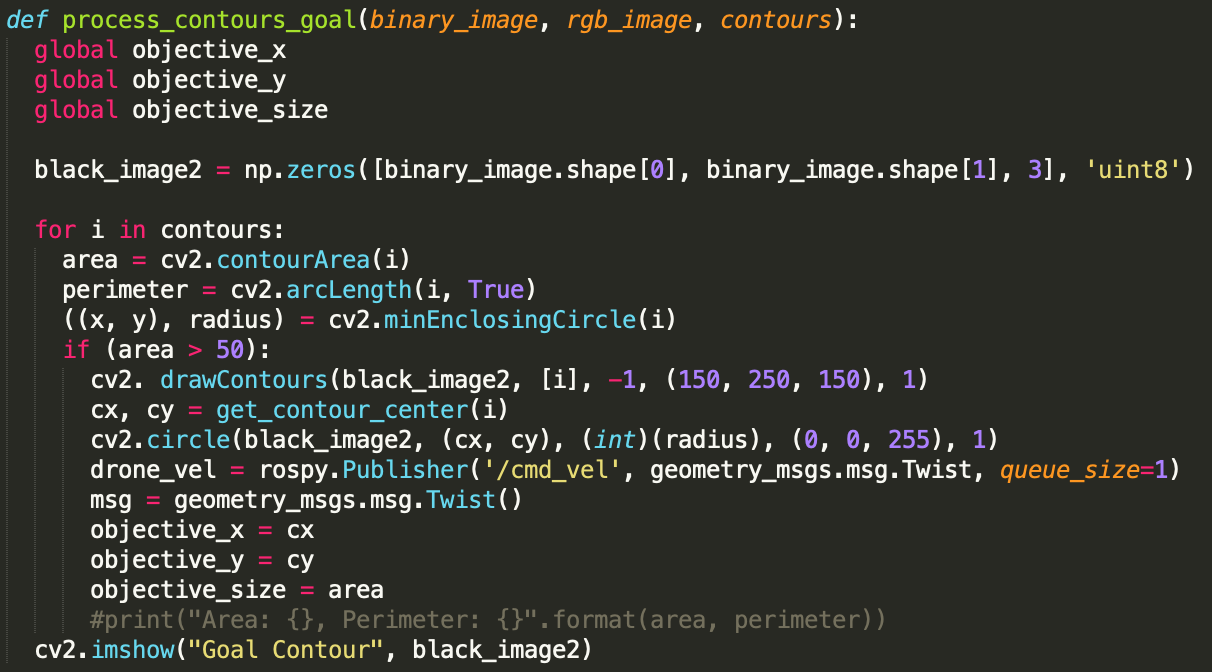
The ‘filter\_color’ function parameters take in the image transmitted by the previous function, the goal’s lower HSV values and upper HSV values which designate the color of the objective and returns a binary image mask.



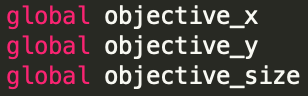
The binary image mask is processed in the ‘getContours’ function and returns an array of contours in the current image mask.



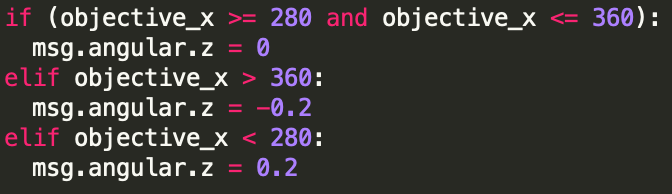
Finally, each contour is processed in the ‘process\_contours\_goal’ function. One by one, a contour center (x, y) and area is calculated and sent to the movement function, ‘hover’, in the form of global variables.



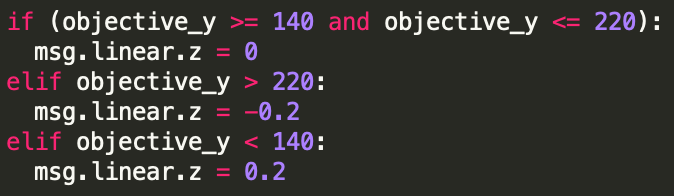
This information is set as a global variable on the host so that separate functions can review this data and process it accordingly. In experimentation, it was necessary to focus on one contour only (the objective) and remove any color from the environment that resembled the objective color.



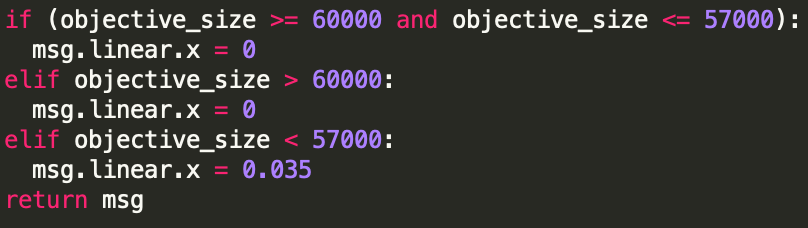
The important data like the x and y coordinates and objective area are used by the motion algorithm. We use the x, y coordinates to determine where the objective is relative to the orientation of the beta drone. For example, if the objective is located at position (420, 80) the beta drone is instructed to rotate right until the objective (contour) is between 280 and 360 pixels.



On the same command it will descend downward until the objective is between 140 and 220 pixels.

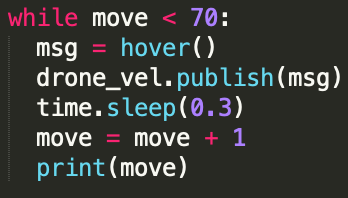


The next data member the drone uses is the area. The area is calculated simply by taking the radius of the contour and squaring it, then multiplying by . The beta drone should approach the objective until the area is greater than or equal to 57000. These figures are variables and conditional statements of the ‘hover’ function. They specify the movements the drone should publish based on the camera feedback



In order to improve a stabilized flight, the drone is given a threshold to abide by. The threshold is the amount of perfection from which the drone can deviate. For example, as mention previously the drone is given and angle of 80 pixels (40 to the left and 40 to the right) to deviate from the x axis center. If the drone was given no threshold in the x axis of the image it would rotate left and right indefinitely because the center of the objective contour would never be perfectly in the center of the image mask. These apply to both the y axis as well. In terms of the objective area, instead of strafing backward, the drone will not change its position if the objective area is greater than 60000 pixels.

The while loop is the feedback loop used to give the drone traversal commands. Each motion performed by the drone is given 300 milliseconds to execute. For example, if the objective area is below 57000 pixels, the drone will move forward for 300 milliseconds. If the objective area is still below 57000 pixels, the drone will move forward another 300 milliseconds. Because the motion values are all a part of the same function, the drone can simultaneously publish movements, like moving forward and rotating at the same time. The ‘move’ integer value is a counter for the loop to check. It can be set to any number or alternatively the while can check ‘True’ Boolean value so that the drone will act indefinitely.



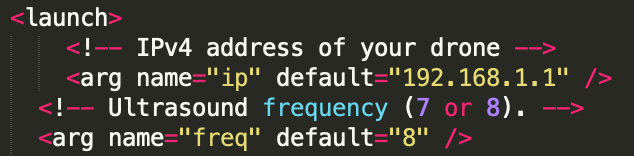
When the move is at its limit, the loop will break, and the drone will land.



## Initial Configuration

The steps outline the configuration, materials, and host setup before running the code.

1. AR Drone 2.0
   1. Verify a fully charged lithium ion battery and equip it to the AR Drones
   2. Equip the wing propeller hull to each AR Drone for safety
2. Linux Machines
   1. Open the AlphaBeta.py file on each Linux device and configure the colors for that drone to follow
   2. The objective pink HSV color used in the tests is lower bound: (90, 90, 200) and upper bound: (180, 255, 255)
   3. The objective neon green HSV color used in the tests is lower bound: (10, 110, 100) and upper bound: (100, 255, 255)
   4. Save each file
   5. Open network connection interface
   6. Connect to the desired AR Drone network
   7. Open a terminal and input the command “roscore” on each machine this will connect to the ROS Master node (do not close)
   8. Open a second terminal and navigate to the directory in which the ‘drone.launch’ file is held on each machine, input command “roslaunch drone.launch”, this file determines what initialized parameters to set when running the program. It returns to the machine various data on the current state of the drone as well, such as battery power, camera usability, etc.



* 1. Note: If “roscore” does not successfully connect, change the IP address in the launch file to “192.168.1.2” instead, if it still does not connect you must change your hostname to this.

## Running Alpha-Beta

Section outlines the steps to properly run the program using multiple hosts and drones.

1. Verify the above steps are functional
2. On a third terminal, navigate to the directory where ‘AlphaBeta.py’ is held on each host
   1. Place each drone in a safe location away from each other in a large environment
   2. To begin flying, run the command “python AlphaBeta.py” on the command line on each machine terminal

Warning: the drones will takeoff approximately two seconds after the command is entered. If for any reason the drones have malfunctioned, stop the program using CTRL+C.

1. If any drone has red lights among any rotors, disconnect and reconnect the battery, the launch file will reconnect with the drone and repeat steps starting with Running AlphaBeta Program step 1.

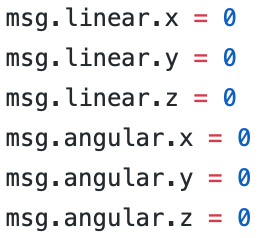
## Complications and Solutions

There were several complications experienced during testing. The most notable ones, and the ones likely to occur for future researchers are red lights on rotors, unstable movement, inability to connect to drone.

Notice that if the drone crashes into an object with enough force, the it will immediately deactivate, fall to the earth, and red lights will show under each rotor. This means that the drone went in to safety mode and cannot fly any longer. To resolve this, disconnect the battery and reconnect it. The “drone.launch” file will lose connection and begin searching again. Eventually it will reconnect, and the drone should be able to fly again.

Occasionally the drone may collide into an object with enough force to damage and compromise rotor integrity by misaligning blades or gears. This incident is considered fatal and the drone should not be flown as its movements are sporadic and dangerous. In order to fix this problem, you will need to purchase tools and parts to repair the rotor.

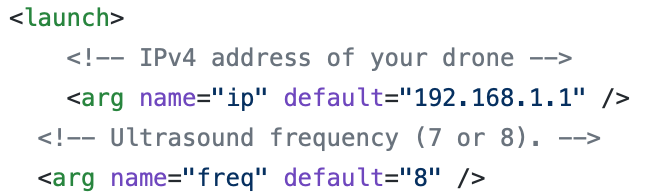
Unstable movement is a very common problem when designing drone motion control. It should be noted that any time a msg.linear or msg.angular member is modified from 0, the drone will be out of stability mode.



The above image is stability mode. In this mode the drone will hover in a single spot after takeoff. Any modification may induce unstable movement to the drone creating undesired results. This is why it is important to stabilize the drone frequently during execution to provide a more stable flight. Another reason the drone flight may be unstable is low battery power. This usually happens when battery power is below 20%. The rotors will not generate as much force to propel the drone any longer and as such, the battery should be recharged for further flight.

Another way to stabilize the drone is by calibration. This requires a smartphone. Download the free AR.FreeFlight app on the App Store, power on the drone, plug in the battery, connect to the drone using its WiFi connection. Open the app on the smartphone and enter the “Piloting” tab. On the bottom of the screen you will see a “Rescue” button, tap it. You can calibrate by tapping both of the bottom two buttons on this interface. This will calibrate all four motors on the drone and the drone should be more stable on takeoff and during movement.

The final and perhaps most frustrating complication is inability to connect to the drone through the “roscore” command on the terminal. To resolve this, the parameter in the “drone.launch” file may need to be changed.



The IP address may need to be changed to “192.168.1.2”. If this does not work, then the computer’s host name needs to be changed. Make sure the computer is wirelessly connected to the drone’s access point. On a new terminal enter the command: “ipconfig -a”, then retrieve IP address. I refer to a Google search to learn how to change your host name to your retrieved IP address.

## External Resources

These resources provided exceptional aid to this project. They also give troubleshooting and solutions to common problems that continuing researchers may have.

* ROS Wiki Documentation (wiki.ros.org)
* GaiTech EDU (https://edu.gaitech.hk/drones/ar\_parrot\_2/ar-parrot-2-ros.html)
* Udemy (https://www.udemy.com/ros-essentials/?couponCode=ROS1GAITECHEDU)

## Conclusion

This study exhibits a successful modelling of an alpha-beta systems. It does however have its limitations. The algorithm stipulates fixed-size objects. Because the area is calculated, a small objective that is very close to the drone would appear the same size on the image mask as a large objective if the drone were very far away. Another limitation are the computers it requires to run all of the drones. A future revision would be to use a USB Wi-Fi module to simultaneously connect to Wi-Fi networks sending each drone separate commands from the host computer. Nevertheless, this project demonstrated many of the applications that can be utilized on the drone creating a complete alpha-beta motion model.