Analysis of Drone Bug Algorithm

Bishop Clark

University of South Florida

# Introduction

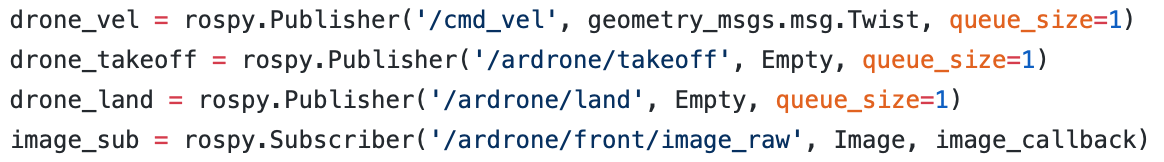
This report details the contents of the program for greater reader understandability and extension. It will cover algorithm design, as well as complications and solutions. Finally, it will mention several external resources for further information and help.

## Objective

To design a program such that a drone is able to safely navigate from one location to another by avoiding obstacles using camera vision technology.

## Algorithm Design

The algorithm was simplified into three different categories: motion control, goal recognition, and obstacle recognition. The motion control used built in ROS functions. The basic idea is for the drone to takeoff, search for the goal, and move toward the goal, avoid, obstacles, and once it has reached the goal, land. These functions initialize every component needed to successfully navigate:

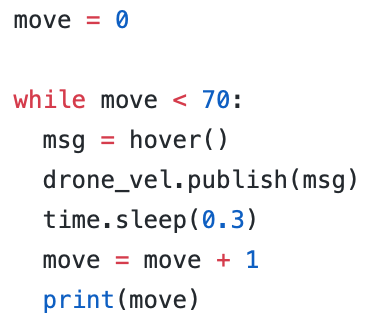


Drone motion capability is stored in the ‘drone\_vel’ variable. The ‘drone\_takeoff’ and ‘drone\_land’ variables store takeoff and landing capabilities respectively, and finally, ‘image\_sub’ subscribes to the camera on board. These variables are used to actuate and desired input on each capability. For example, if it was set that drone altitude is drone should fly forward (msg.linear.x = 1) and fly upward (msg.linear.z = 1) then nothing would happen until it is published by ‘drone\_vel’.

Onward discusses the code in the order that it is processed when executed. Once the publisher and subscriber functions are initialized, the drone takes off. The takeoff function hovers approximately 1 meter in the air.

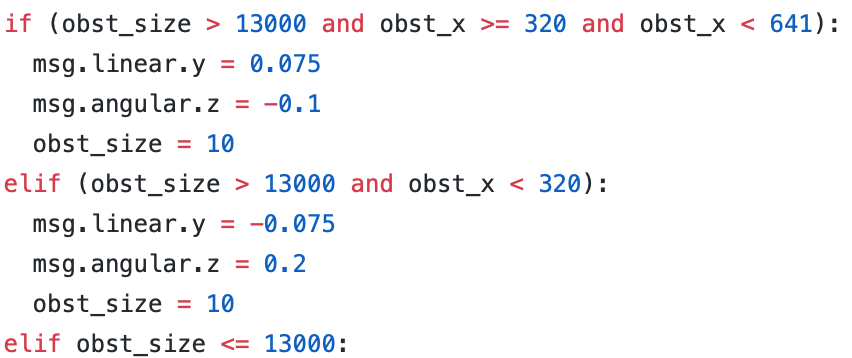


This function will ascend almost directly upward. Then the feedback loop starts. This is where the drone will take input of object locations by camera and move accordingly from conditions in the hover function.

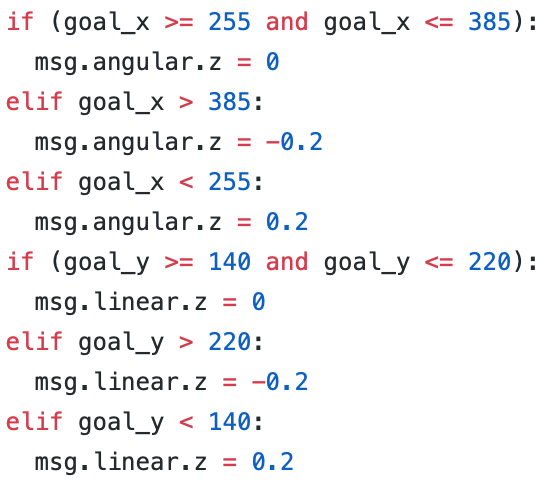


Move is a variable that counts. The while loop will break once move reaches 70. The purpose of the loop is to perform actions on the drone. Each action such as ascend, rotate left, move forward all receive 0.3 seconds to perform. This gives the drone enough time to perform fluidly and not stagger during motion.

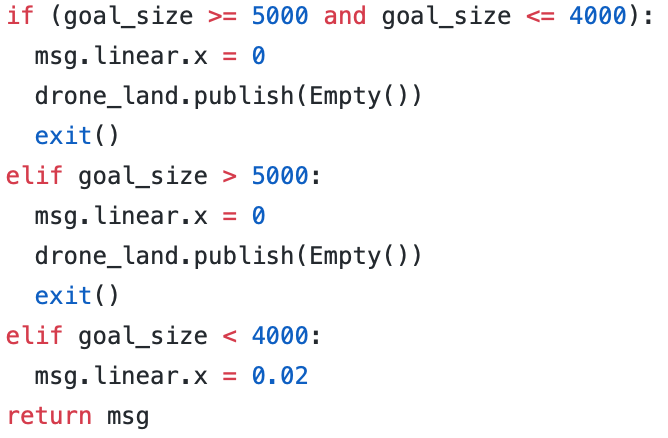
The hover function is where the main algorithm resides. The drone first captures through camera stream, the environment. It will rotate leftward until it sees the goal and will do this anytime it does not see a goal. Afterward it will decide if there is an obstacle in the way. If so, it will move accordingly, an obstacle on the left side of the camera (0pix < x < 320pix) causes the drone to strafe right. Conversely, an obstacle on the right side of the camera (320pix ≤ x < 640pix) causes the drone to strafe left. Similar actions correspond to below the camera status and above the camera status. The variable ‘obst\_size’ holds the feedback size of the obstacle. The size of the obstacles are approximately 9 x 6.5-inch papers. Essentially, the drone will act perform these conditions if the area of the contour is greater than 13,000 (≈ 30 inch). During these actions, the drone will rotate in the opposite direction to better position itself toward the goal. This helps efficiency to the goal. Because drone flight deals with three dimensions of possible movement, it would be safe to rotate before strafing. For certain (concave) obstacles this would be hazardous because the drone is not able to detect any object from its side. However, this project focused solely on convex obstacles and goals.



Next, the drone positions itself on the z and y axes. Simply put, if the goal is to the left side of the camera, the drone will rotate left. If the goal is to the right of the camera, the drone will rotate right. If the goal is below the camera, the drone descends. If the goal is above the camera, the drone ascends. Essentially, this part of the algorithm the drone centers the goal according to the position of itself and the goal. Note: the maximum speed that the drone can move is 1, however, no setting is set above |0.2| because every higher value tested caused the drone to move so fast that it could not properly track objects. Similar to the obstacle recognition, the drone uses the location of the goal on the image to determine where to go (e.g. goal\_x = 100 pixel, the drone will rotate left because the center of the camera is at 320).



Finally, the drone moves toward the goal. The size of the goal used in testing is 3.5 x 4.5-inch paper. The drone uses the image to determine if it is close enough to the goal. If it is greater than 4,000 (contour area), then the drone moves forward. Every above case is considered and returned as a message object that the ‘drone\_vel’ variable will publish to actuate. If the goal meets the distance requirement, less than 4,000 (contour area) then the drone will land and shutdown.

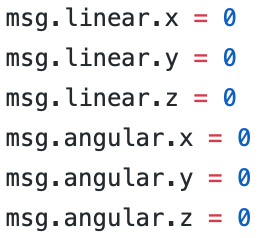


## Complications and Solutions

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.

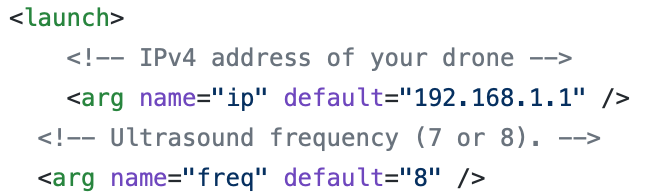
Notice that if the drone crashes into an object with enough force, the it will immediately deactivate, fall to the earth, and red lights will show under each rotor. This means that the drone went in to safety mode and cannot fly any longer. To resolve this, disconnect the battery and reconnect it. The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

Unstable movement is a very important issue when designing a path to a goal including obstacles. 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.

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 project demonstrates a successfully performing algorithm that navigates a drone through obstacles to an end goal. It is not without limitations however, namely, this algorithm does not take into account concave shaped obstacles. Instead it assumes concave obstacles for higher efficiency. A revision is certainly to include concave obstacle possibilities. Also, the algorithm uses fixed-size objects. This is a limitation because for the highest possible accuracy, every obstacle needs to be the exact same size which is unlikely in real world scenarios. A future revision is to recognize obstacles based on shape rather than its color. This would allow for a more realistic approach to navigating through obstacles. Despite, these limitations, the algorithm effectively traverses space in three dimensions to safely arrive at any seeable location.