Introduction to Armstrong, the ARDrone 2 Quadricoper

Comp 380

Susan Fox

February 14, 2014

1 Overview of the ARDrone 2

The ARDrone 2 is a basic quadricopter, a flying robot that uses four horizontal rotors to provide lift and stability. It can be controlled by a free app for the iPhone/iPad, and perhaps also for Android. As such, it is a fancy remote-control toy. It comes with a distance sensor that points downwards, so it can estimate its height above ground level. And it has a single high-definition camera facing forwards. It also provides its own WiFi access point: you control the robot by connecting to its WiFi signal. The makers of the ARDrone released information about how it communicates, allowing roboticists to create tools for controlling the drone from any computer.

We'll be using a library for Python originally created by a student from Harvey Mudd College for the first-generation ARDrone. It has recently been updated for the second-generation drone we have. There are also tools for controlling the drone through Calico, and also through ROS, but I haven't used those before.

Note that when you work with the drone, a wide-open space with few obstacles and as few drafts as possible is ideal. The atrium is wide-open, but there are lots of drafts. The downstairs courtyards may also work. I saw someone fly a drone off a balcony like the ones between levels, but I don't recommend it!

2 The `libardrone` library

Get a copy of the folder `python-ardrone-master` from Moodle. It is written to work with Python 2.7; I will work on a port to Python 3, but I don't have that yet. It is set up to install the library into your Python system, but you don't have to do that unless you want to.

Look inside this folder, at the subfolder called `libardrone`. There are a lot of Python code files in there, but you don't need to read them. If you don't install the library into your version of Python, then I recommend that you put your code files in this folder to keep life simple.

The `libardrone.py` file contains the main code for the library. What you need to know is that you control the robot by created an `ARDrone` object, and it has methods and data for everything else you do.

The next set of subsections will summarize the tools you need to manipulate a drone. Particularly because of its volatility in the air, and because of the nature of the movement commands, I don't recomment that you try these out in the Python shell. Instead, write small script programs that handle the whole "life-cycle" of the robot in the air: taking off, moving, and landing again. Use the `time` module's `sleep` function to insert time delays between movements.

I am also providing you with a copy of my demo program. This will let you control the robot using the keyboard. Use this to get a feel for how the robot works, and read the code to see how to use the commands in the library.

2.1 Creating an ARDrone object

Start by creating the ARDrone object. You must pass the input `True` here, because that tells the class that the robot it is connecting to is an ARDrone 2, not an ARDrone 1.

```
import libardrone

myDrone = libardrone.ARDrone(True)
```

2.2 Movement commands

Below is a table that lists the main flight methods in the `ARDrone` class. Most of these are self-explanatory. Just be aware that the movement commands, once turned on, stay on until you tell the drone to hover! You must turn on a movement, sleep for the amount of time you want the movement to take, and then tell the drone to hover, to cause the movement to stop.

Method call	Description
`myDrone.takeoff()`	Causes the robot to take off
`myDrone.land()`	Causes the robot to land
`myDrone.reset()`	Causes the robot to go into emergency mode and reset (generally catastrophic)
`myDrone.hover()`	Causes the robot to stop moving and hover in place
`myDrone.move_left()`	Causes the robot to start moving to the left, not changing heading
`myDrone.move_right()`	Causes the robot to start moving to the right, not changing heading
`myDrone.move_up()`	Causes the robot to start moving up
`myDrone.move_down()`	Causes the robot to start moving down
`myDrone.move_forward()`	Causes the robot to start moving forward
`myDrone.move_backward()`	Causes the robot to start moving backward
`myDrone.turn_left()`	Causes the robot to start turning to the left, in place
`myDrone.turn_right()`	Causes the robot to start turning to the right, in place
`myDrone.trim()`	Causes the robot to straighten itself in the air
`myDrone.set_speed(speed)`	Causes the robot to change its speed, valid values are floating-point from 0.0 to 1.0 inclusively.

2.3 Data and Images

We are supposed to be able to get a video stream from the robot's camera. I haven't gotten that to work yet, but the `demo.py` illustrates how it is supposed to work. However, we can get the robot's navigation data and still images from its camera at will.

Method call	Description
`myDrone.get_image()`	Returns the robot's camera image, as a Numpy array
`myDrone.get_navdata()`	Returns a dictionary containing navigation data

Note that the image comes back as a Numpy array, and must be converted into another form for image processing or display. The `FoxDemo.py` code shows the steps in converting the array to a Python Imaging Library (PIL) Image, and from there to a form displayable using `Tkinter`. PIL contains many tools for basic manipulation of images.

3 Tasks to try out

1. Start with a simple program that makes the robot take off, fly forward about 4 second, and then land. Use this program to experiment with how speed and time convert to distance for the robot. Vary the robot's speed, and record how far the robot goes in horizontal distance. For each speed, try multiple runs (at least 5) and mark where the robot ends up. How precise is its flight? How close together are the ending points.

2. Write a program to make the robot pace in the air. It should fly forward, turn around, and then fly back to its starting point. Try this in a variety of settings, to see if you can assess how feasible it is.

3. This last task asks you to write a program that is like my demo, using tele-operation, but that is a little less controlled by the human user. Use `FoxDemo.py` as a model, and write a GUI program using `Tkinter`. In this case the robot should take off, and then slowly circle in place, showing images as it goes. When the user sees her target, she should type a key to make the robot stop spinning. Then the robot should move forward (presumably toward the robot), until the user types a key to make it stop. The user should be able to type another key to make the robot go back into scanning mode, and a key to tell the robot to land and quit. See me if it's not clear what I want here.