Documentation on the GUI

Ronniel Padua

Inside this document contains key features I have not been able to complete due to lack of time. They are sorted by usefulness to the continuation project. To further help you develop the *Drone Management /Visualization Application*, I have made decisions that should make your life a bit easier when adding something new. The current GUI is implemented in Tkinter and uses Python.

Just remember:

Although this is part documentation and part guide, **feel free to ignore my recommendations if you think you have a better way of doing things**. I learnt a lot of Python when developing this GUI, from error handling, running commands, test cases, classes etc. There are some artifacts you may want to remove.

Please keep the GUI framework language to an easy language that electrical engineers were taught.

(STRONGLY RECOMMEND: PYTHON)

How Do I Run This?

First things first.

Make sure, using VS code, that you are in the 'DroneGUI' folder when you are running the GUI code.

This is because the relative imports require DroneGUI to be the top of the directory when it is run.

Make a venv using VS code and activate.

Then, install all of the packages used by the GUI into the venv you created. Installpackages.sh should run this automatically once the venv is activated. Failing that, just run each command individually in the venv.

There are tests in the files that end in "tests". Run them and see if they all pass. The tests that require an actual drone to be tested on are currently disabled. This is because, if you don't update the drone the GUI is sending to, the test will fail.

Under The Hood

The GUI has three 'main' parts to it. These are the 'Configuration Buttons', 'Drone Buttons' and the 'DroneList'. Each of these have their own classes and are designed for different things.

Configuration Buttons is supposed to do something to all of the drones, or to help set up the working environment. DroneButtons includes buttons which only affect that drone. Drone List allows you to command to those drones that you select.

How Does It GO?

When you are working with the drones, you will inevitably be using the linux terminal. As you may know, you put in commands and the terminal does whatever it was told. With the GUI however, instead of typing commands in the terminal, the GUI, using subprocess, runs the same command in the shell. Tkinter simply connects that function to the button through the use of a callback function.

See the notes on subprocess() about shell escaping, because this can be a serious issue with letting the user type things in.

What Have I Not Done?

Quick Setup

A button called 'Quick Setup' (found in droneactionhandlers.py). This is just the basic send file command in the Drone class but adapted to send a known file to the drone. The purpose of this button is to send all of the information needed to the drone. It is not operational, because it only sends the files to the drone but does not run it. The two files are stored in KeyFiles. When that button is pressed, it sends them to the drone.

There are some things that you really want to do:

1. Send **ROS configuration files** to the drone. Currently, it just sends two files to the drone which collect the MAC address and prints it to the terminal and an example webserver.
2. When those ROS configs are sent, the drone should be ready to fly when commanded to.
3. Find a way **to download files using the webserver** when a command is sent from the computer. This will let you download all the text files you want for use in logging, debugging, etc.
    a. Ideally, any file can be downloaded. This way, you can focus on parsing and displaying the data.
4. The above is important so you can get telemetry data.
5. Add the **files you want to send into KeyFiles** and edit the quick_setup function to send it and then **run the files**.
6. **Alter the SwarmActions and DroneListClass (SwarmClass.py)** classes to quick_setup all of the drones. There is a button on their respective pages, but they are not operational.
7. Change the return_wireless_info_from_drone class in DroneClass to run a saner way to get the mac address.
    a. It sends a bash script, runs a command, then collects the info from the bash script.
        i. Seriously, change the method to get the MAC address since it is extremely convoluted.
8. To support above, add more functions to collect_info (in KeyFiles) that get more information from the drones.
    a. It can currently get the OS version and MAC address. Getting the ROS version would also be nice.
    b. Any other information you want could be added to mac_address too.

Live Telemetry

This isn't too necessary because QGroundControl provides some live telemetry. However, I was envisioning some way to visualize the data from the drone.

That would require these steps:

1. Generating the information on the drone
    a. A file that, when commanded to run, starts logging data to a text file would be workable

        b. ROS is already able to do this, so working with the flight controller/path planner to learn how ROS logs information could be helpful.
2. Getting the information from the drone to the computer
    a. The webserver I wasn't able to develop can be a good way to do this, if you learn Flask.
    b. It may also let you connect to the drone using the webserver, allowing you to send commands or files via Flask (probably unnecessary)
3. Visualizing the data
    a. This should be matplotlib or an equivalent. If you've done any courses that used matplotlib, this should not take you long to parse the text files and display them.

You Might Want To Change These

The Four Classes

SwarmClass.py, DroneClass.py, DroneListClass.py have four classes, mostly doing the same things. SwarmActions send a command to all drones, DroneClass to only one drone and DroneListClass to whatever drones you select.

What you might want to do is:

1. Fold the four classes together. They are mostly doing the same thing (sending files/folders, eventually quick setting up the drones)
2. Just watch out for making an omniscient class (also called God class). Since the folded class will be used across most of the program, you might end up with one class that all the important files access.
    a. Since I like separation of concerns, I have made the four classes instead.
    b. This way, altering one class doesn't affect other files in the program.
        i. The downside is that updating the code means four classes need to be updated.

I want to add more stuff!

If you just want to add more methods, add them to the relevant class (SwarmActions for all drones, DroneClass for just one etc). The purposes of the classes are to be extensible by *adding* code instead of *editing* it. That way, you don't have to care about the other code and can focus on getting more methods in.

However, that may end up with a bloated class, especially if you fold all of the four classes together.

I Think Tkinter Looks Terrible!

Why'd I choose this 90's looking program?

I picked Tkinter because it was much faster to develop than Qt (C++). QT was also a serious pain trying to find out what I needed to do to get it to work. Feel free to choose another framework if you want.

I strongly recommend Python for any future GUI framework

I had to restart the entire GUI because the 2023 GUI was in **undocumented** *TypeScript* and used React. That was an impossible task to understand (electrical engineers are not taught TypeScript), let alone develop (I had no idea how that worked). I hope that I have made enough good choices to help you develop the GUI without scrapping everything.

However, if you want to change the GUI framework (to something **Python**, please), then there are some things I have done that might help you.

Separation of Concerns

All of the front-end stuff is separate from the back-end, following good coding practice when making GUI's. That is, adding more methods to the classes does not affect the front end, and making the GUI prettier (within Tkinter's limits) does not affect the magic in the back.

What does that mean for you?

Well, if you think Tkinter looks bad (it does) and you want to change it (**Python** please), then you can remove the Tkinter-related stuff but keep the classes. Those classes carry out the *actual* commands on the drones/do the stuff on your computer. As a result, this lets the new GUI framework (in Python) to call the same methods this GUI uses. You might need to edit the classes, but you shouldn't have to delete everything and start over.

Word of warning:

Keep separation of concerns in mind. The **look** of the GUI should not affect the **'do stuff'** of the GUI. This is why I have used classes, functions, etc etc. I tried to keep the functions and methods as single-purpose as possible and rely as little on other functions as possible, especially Tkinter functions.

Moving On

One Final Recommendation

Learn how to use the unittest or another testing module. Learn how to make unit tests and mocks.

This way, you can develop tests for the code *before* you implement something.

The tests serve as a 'this code should do this'

*You write your code to **fulfil** that test.*

Thus, when you finish your code, you **know** it works. Another benefit is that changing your code isn't too risky, because the tests will tell you if something else has broken.

This is a basic test-driven approach.

Just keep your tests short – one thing to test at a time.

I lied, have more suggestions to help you (maybe)

Keep your functions short so they are easy to test.

Keep your functions/methods as self-contained as possible. This lets you change things without affecting the entire program.

Try and keep class sizes small. If it's huge, changing it will affect the entire program and that will be a serious pain to debug. (My class sizes might be too large already)

14/10/24 – Ronniel Padua

Thank you for reading through this document. Hopefully, along with the comments in the code (and if my coding style helps you, also the code itself).

With any luck, you’ll be able to finish up this GUI and make it do very cool things to the drones.

Best of luck to you!