AntwerPi 2.0 Documentation

Release 1.0

Evert Heylen

Mar 23, 2017

Contents

CHAPTER 1

Quick links

- Github: https://github.com/eestec-antwerp/AntwerPi2
- Event website: http://www.eestec.be/antwerpi2/

CHAPTER 2

Index

The documentation is split in two big parts, one about the quad itself, and one about the EESTEC event.

Contents:

Getting Started

About

This quadcopter was originally made for EESTEC Antwerp's workshop "AntwerPi 2.0" in end of June 2016. Hopefully it can find more usecases through this documentation.

The quad was built from scratch, and used rather common (and trustworthy) parts, with the exception of the Navio2 which is rather new in the world of DIY avionics. The combination of a Navio2 and a Raspberry Pi allows for a much greater flexibility than either a traditional DIY drone which lacks a proper way of programming it, or a commercial drone, which lacks an open API (like DJI's drones), or extensibility for sensors or networks (the RPi and Navio2 provide analog and digital (USB, UART, Ethernet) input).

Issues or TODO's about the drone can be found here: https://github.com/eestec-antwerp/AntwerPi2/issues.

Get your code/sensor flying

1. Learn a bit about our drone
    - See the *Links* for a presentation given during the event (recommended!), along with other important documentation.
    - Optionally, a detailed list of the components in the drone can be found at *List of parts*.
2. Learn how to program the drone it in *Software*
    - Optionally, test your software in SITL.
3. Learn how to add or modify hardware in *Hardware*

4. Learn how to prepare for a flight, and maintain the drone afterwards in *Setup and maintenance*
5. Learn how to fly (in case you need to take over).
    - This is not covered by this documentation, but you can get the basics using this video and a cheap commercial quad (that can take a beating) to train with.
6. Fly!

Apart from that, there's also a *Glossary* to help you understand all the terms.

List of parts

The list of parts is splitted in 4 categories:

- **Heavy electronics**: Parts dealing with high amounts of electricity, usually driving a mechanical process.
- **Fine electronics**: Parts dealing with low amounts of electricity, driving computational or sensing processes.
- **The rest**: Frame parts, landing gear, ...
- **Ground equipment (not on the drone itself)**: Transmitter, battery bag, charger, ...

Summarized, the parts of the drone are: a DJI F450 frame, a DJI E305 (800kV) propulsion kit, a Raspberry Pi 3B and a Navio2 on top of it, all driven by a 4000mAh 4S battery.

Heavy electronics

- **Battery**: 2 x ZIPPY Compact 4000mAh 4S 25C Lipo Pack]
- **ESC's, motors, propellors: DJI E305 quadcopter propulsion set**
    - props are 9.4x5.0

Fine electronics

- **Raspberry Pi 3 B** for the computing part (and an microSD of 8GB)
- **Flight controller** (kinda): Navio2, from Emlid
- **GPS/GNSS antenna**: from Emlid
- **Navio2 wire pack**: from Emlid
- **Power module**: from Emlid
- **RC reveicer**: Quanum i8
- **Some servo cables**: http://mikrokopter.altigator.com/servo-extension-cable-5-cm-male-p-42072.html

Basically, we bought this combination from Emlid: https://emlid.com/shop/navio2/

Additionally, there may be an air quality sensor on the drone. The cables are kind of custom made, but the sensor itself is from Groove. The cables can be reused to get input from any analog Groove sensor.

The rest

- **Frame**: F450 (the real one, not a fake one. Also just the frame, not the full kit)
- **Landing gear**: F450/F550 landing gear (be advised, on a F550 this is rather bad).
- **GPS mount**: http://mikrokopter.altigator.com/gps-folding-mount-p-42006.html

Ground equipment

- **Charger**: IMAC B6AC V2
- LiPo bag
- **RC transmitter**: Quanum i8 (same as receiver)

Software

Drone

The drone runs the official image from Emlid, which runs Raspbian. On top of Raspbian, you will usually run one or more of the following software:

- **ArduPilot**: Absolutely necessary, as it provides all to logic for flying. Sometimes referred to as APM, but that is just a popular flight controller running ArduPilot. The software project ArduPilot is split in some parts, of which we only use *Copter* (sometimes referred to as ArduCopter, or APM:Copter). Backed by DroneCode.
- **MAVProxy**: A popular ground station, but in a drone it will be used primarily for proxying.
- **DroneControl**: A small web application developed by Evert, so you can more easily start/stop ArduPilot and shutdown the Raspberry Pi. Source here: https://github.com/evertheylen/DroneControl

Ground

On the ground you may run the following software:

- **MAVProxy**: Both as a proxy, or a GCS. It looks promising as a GCS but has a steeper learning curve.
- **QGroundControl**: A decent GCS, works in all major OS's (including Android, but not recommended). More info in **'their site<http://qgroundcontrol.com/>'_**.
- **Tower**: A very stable GCS, and the de facto standard for Android. Get it here (you'll also need 3DR Services)

There are a lot of other GCS's, but these are tested with our drone (MAVProxy not so much, but their proxy functionality is solid). More GCS's can be found here.

Communication

The holy grail of open-source drone software is the MAVLink protocol. However, some flight stacks (like PX4) speak a different dialect, so watch out for incompatibilities with MAVLink-based drones. MAVLink is binary and can be sent over special telemetry modules, but we just use WiFi, usually encapsulated in UDP.

Proxying

As suggested above, the king of proxying in the drone world is MAVProxy. The proxy is bidirectional. I won't provide instructions for using it as a full GCS, only for using it as a proxy. Let's say Arducopter is running (locally) at port 14550 (as is usual when you want to proxy), and you would like to use **two** Ground Control Stations, one at 192.168.1.2:14550, and another at 192.168.1.3:14550. The corresponding MAVProxy command would be:

```
mavproxy.py --master localhost:14550 --out 192.168.1.2:14550 --out 192.168.1.3:14550
```

Keep this process running or all GCS will lose control of the drone!

Alternatively, you can use the `output` command in the MAVProxy shell to add or remove outputs:

```
MAV> output list
0 outputs

MAV> output add 192.168.1.2:14550
Adding output 192.168.1.2:14550

MAV> output add 192.168.1.3:14550
Adding output 192.168.1.3:14550

MAV> output list
2 outputs
0: 192.168.1.2:14550
1: 192.168.1.3:14550

MAV> output remove 0
Removing output 192.168.1.2:14550

MAV> output list
1 outputs
0: 192.168.1.3:14550

MAV>
```

Of course, you can also do the proxying on your own laptop, rather then on the Raspberry Pi itself. In that case the `master` may be an external IP address, but the `outputs` may be local.

NOTE: The DroneControl program does not use MAVProxy.

Programming

The usual way of programming a drone would be to write a program that sends MAVLink commands to the drone, while the code runs on another PC. However, with the Raspberry Pi as our main flight controller, we can run that program on the drone itself, and theoretically achieve a fully autonomous drone (of course, not recommended for safety).

Manually crafting and sending MAVLink packets is rather tedious. If you're programming in Python, you're in luck: 3DR created DroneKit. Sadly, it's Python 2 only. (Originally Dronekit spanned much more than the Python component but the other parts (like the cloud part) never gained much traction).

It is important to note that a program written with dronekit acts as a full-blown MAVLink receiver/transmitter. To run your program while also being able to use a GCS (*highly* recommended), using MAVProxy is essential.

Tip: The combo Raspberry Pi + Navio can be powered in three ways:

- Through the power module from the battery

- Through the servo rail (not used in our drone)
- Through the Raspberry Pi's power

If you use the last way of powering it (there should be a specific charger for it, since it uses more than most smartphone chargers can provide), you don't have to use any batteries to develop on the drone. Everything will work (including RC and GPS etc), but there won't be power going to the ESC's.

SITL

TODO

http://python.dronekit.io/develop/sitl_setup.html

Maintaining

While I am a fan of bleeding-edge software, it may mean your drone falls out of the sky. Therefore, only install updates to flight-critical software if it's encouraged by Emlid. After major updates, you should recalibrate the drone.

Hardware

Schematic

A big schematic can be found here.

Adding sensors

There are UART, I2C and ADC ports available on the Navio, in addition to the usual ports on the Raspberry. For using the ports on the Navio, it is best to just look at the bottom of the Navio to know which wire has which function.

Adding a Raspberry Pi Camera is **not** recommended, because of the reported noise on the GPS frequencies, making the GPS almost useless.

Setup and maintenance

Preparation

Charging LiPo's

The IMAC charger we use is in fact a so-called 4-button charger with a built-in PSU. If used with the standard batteries, it should already be set to the right settings (4S LiPo). There are 3 modes that might be of interest:

- Balanced Charge: The charger will fully charge your battery, and balance the cells. This is the most used.
- Fast Charge: The charger will almost fully charge your battery, but will **not** balance the cells. This will save time, but is bad if done too often. Therefore, only do one subsequent fast charge, after which you should use a mode *with* balancing.
- Storage: The charger will empty or fill the battery until about 40%, and balance the cell. As the name implies, this is best for storage. A fully charged battery may damage itself over time.

ALWAYS first plug in the two banana plugs, this will prevent the two plugs from accidentally hitting each other and short circuiting and exploding etc. In fact, it's best if you strictly follow this order:

1. connect charger to AC
2. connect cable (JUST THE CABLE banana plugs -> XT60) to charger
3. connect balancing plugs to charger
4. connect battery to cable's XT60 end

LiPo's can explode or catch fire. Therefore, whenever you can, put them in the safety bag. Never store an unbalanced battery for too long!

More info about 4-button chargers can be found here: https://www.youtube.com/watch?v=kvr-25yGeVk. More info about LiPo's can found on wikipedia or so.

Charging receiver

The RC transmitter has it's own Li-ion battery, which is charged through the transmitter itself using a Micro-USB connection (a standard smartphone charger works). While I never had an empty battery, it's not possible to check the level. Therefore, make sure you charge it before every day of flying, since the RC connection is your last controllable fallback.

Flight

Checks

All these conditions need to be true:

- There is absolutely no rain or too strong winds
- The "GPS weather" is good enough to fly: http://www.uavforecast.com/
- The risk to crash into a person is basically non-existent.
- The battery of the drone is in good health, and so is the battery of the RC transmitter

Before

1. Place drone on a flat surface. The red arms should point AWAY from you.
2. Install propellors (a CW motor should have a CW prop, same for CCW). The props are self-tightening, this means you have to screw them on in the opposite direction of how they'll spin in flight.
3. Connect battery (the ESC's will start beeping because they don't get a valid signal)
4. Connect smartphone or laptop to Drone's WiFi network
5. Start Arducopter (the ESC's will – thankfully – stop beeping)
6. Start and connect your GCS (go to *Ground* for options)
7. Check for any anomalies in your GCS

Takeoff

There are a lot of different modes of flying. I will explain the steps assuming you use *Stabilize*, which is the most difficult of them.

1. Arm the drone either through your GCS or by holding both sticks in the bottom right corner for 3 seconds.
2. (not possible in Alt Hold or similar) Give a little bit of throttle, check again for any anomalies (visually).
3. Give it enough throttle to take off (50%). Don't do it too slow, the drone may tilt. Afterwards, the drone will hover at about 40%.

Tower or other GCS may provide you with a button to take off.

During flight

The upper left switch can be used for BREAK mode (fully upwards). The drone will stay where it is, and not respond to any input, unless you switch modes or pull the switch down. Turning the knob on the right WILL make you switch modes! The BREAK mode is GPS-dependent, so if GPS fails this is not a safe fallback. Flying in STABILIZE is a better option then, provided you are a good pilot.

Switching modes can in theory be done through the knob, but this is a bad idea. Some of them might be modes with altitude hold, in which case the throttle should be at 50% to hover, and others may be without altitude hold, in which case the drone will go in a reasonably fast climb at 50% throttle. To prevent "cycling" through the modes, use your GCS to switch modes.

WiFi is not particularly strong. The drone uses the built-in WiFi of the Raspberry Pi, and the GCS may be a smartphone. Do **not** go too far with the drone, and **definitely** never out of LOS.

Landing

Slowly. Tower or other GCS may again provide a button for this.

After

In DroneControl, first stop Arducopter (the beeping will start again), and then shut down the Raspberry Pi. Then, you can disconnect the battery.

Safety

Controlling

The person holding the RC transmitter must be kept in sync with possible warnings the GCS is giving. Either he has a smartphone mounted on top of the transmitter, or someone else must hold the smartphone or laptop. That other person should not be distracted by the drone itself, instead the focus must lie on the GCS.

If someone else is responsible for the GCS, mode switches will also be much easier.

Fallbacks

It is wise to provide as many fallbacks as possible. These may include loss of signal (both Wifi and RC), geofences, and more. If your drone is totally uncontrollable, and the GPS is failing too, consider using the kill switch. It'll crash the drone, but hopefully it won't crash into a window ;) .

More information about this can be found on the Arducopter wiki.

Running a program

When running code on your drone, guaranteeing safety is more complex.

TODO: A way to remove the program from the output list of MAVProxy at the push of a button...

Maintenance

Calibration

Whenever something physically is added, removed or moved from the drone, or the drone is behaving weird, you should consider calibrating the sensors. More information here: http://ardupilot.org/copter/docs/configuring-hardware.html.

Calibrating the ESC's is not needed for our drone.

Maintaining software –> see *Maintaining*.

Links

Documentation

The internet is full of resources, but they can be rather overwhelming. Therefore, here is a collection of interesting resources:

- How to make a Drone: Presentation by Evert, given during the workshop. 54 slides.
- The Beginner's Guide to Multicopters (Polakium Engineering): pdf, 19 pages. Ignore all the stuff about flight controllers.
- Arducopter wiki: Contains some generic info about drones too.
- Navio2 wiki: Specifically about the Navio2. Sometimes confusing. The forums are rather responsive.

Shops

There are a lot of shops to get spare parts. Here are some of more popular ones, and comments for each one:

- Hobbyking: Cheapest of them all. Has a number of different warehouses around the world. If you wait long enough while looking at a product, they may suggest an even lower price (with a warning that it won't appear again – it will).
- Altigator: Rather expensive, but based in Belgium so shipping is cheaper/faster.
- MHM Modellbau: Random shop in Germany, where the frame and landing gear is bought.

Apart from this, Raspberry Pi's and Navio stuff is bought from the usual places.

Glossary

Hardware

DF-13 A kind of connector (for data/low power), with a varying amount of pins. Somewhat of a standard among drones. The Navio has a lot of them.

XT60 A connector for batteries, rated up to 60A.

UART Universal Asynchronous Receiver/Transmitter: see http://electronics.stackexchange.com/a/37817

I2C (Officially written as I^2C), Inter-Integrated Circuit: see http://electronics.stackexchange.com/a/37817

SPI Serial Peripheral Interface: see http://electronics.stackexchange.com/a/37817

ADC Analog-to-Digital Convertor. The data is the voltage.

ESC Electronic Speed Controller

LiPo Lithium-Polymer [battery]

CW Clockwise

CCW Counter clockwise

C-rating A confusing way to rate batteries on max current, has nothing to do with Coulombs. See http://www.revolectrix.com/tech_data/lipoCalc/Battery_C_Rating.htm

Software

MAVLink Binary protocol over which all the *DroneCode* related projects communicate.

GCS Ground Control Station, eg. QGroundControl or Tower

SITL Software in the loop

APM ArduPilot Mega. Sometimes used to refer to the hardware, sometimes used to refer to the entire ArduPilot project (which is software).

PX4 An alternative Open Source flight stack, again often used to refer to the hardware running it (Pixhawk). Also speaks MAVLink.

General

DroneCode Organisation started by the Linux Foundation that organizes open-source efforts in the UAV community.

3DR A very prominent company in DroneCode. One of the few brands competing with DJI that isn't Asian.

DJI Basically the biggest drone company out there. Software is very closed, but they sell good hardware.

Rx Receiver

Tx Transmitter

Start here

Docs for the EESTEC project during AntwerPi 2.0.

These will be updated throughout the event. Check the sidebar to find the info you need.

LoRa

We use the following configuration:

```
Raspberry Pi
         |
         | (UART)
         |
LoRa module
         |
         | (LoRa)
         |
Proximus
         |
         | (HTTP)
         |
server UA
         |
         | (HTTP)
         |
own VM (also at UA)
```

The code for the VM at the end of the chain can be found in *lora_server*.

The server is available at http://lora-endpoint.mosaic.uantwerpen.be/.

Currently, it is just a very simple Tornado server that logs all POST requests it gets.

Python Intro

We strongly advise this introductory tutorial. It's meant for students at Berkeley, but it works for you too. You can go through it at your own pace, ask questions where needed. If you're done, please let us know as well.

Note that we will be using Python 2.7 for most of the programming.

Coming from C

A quick comment for those coming from C: Python is quite different, and you'll have to relearn a lot of stuff. Luckily it'll go fast, Python is an easy language. Just try to not compare everything with C.

One quick tip: C is not usually considered an Object-Oriented language, but if you've ever done something like this in C:

```
struct drone {
    int x;
    int y;
}

void move(struct drone* drone, int x, int y) {
    drone->x = x;
    drone->y = y;
}
```

Would translate to this in Python:

```
class Drone:
    def move(self, x, y):
        self.x = x
        self.y = y
```

You can of course extend the Python example with an `__init__` function, and you get a lot more OO tricks than in C.

Coming from Java

You're in luck; Object-Oriented programming is Java's bread and butter and Python will happily handle any OO programming you throw at it. An interesting article discussing how you should change your coding style can be found here: Python is not Java.

While Python is also OO-oriented, try not to put everything in classes, free standing functions are better than static methods in `Util` classes. Also, MVC has no place in drones :) .

Drone simulation

In this exercise, we will try to optimize the path of the drone, given a set of positions and other constraints. For this you should download tsp.zip and extract it. The rest of this project will assume that the files are in the current directory.

Efficient paths

Drones fly. Well that?s an obvious statement. But they shouldn?t just fly around randomly, they often have a purpose.

The part of the code you will have to modify is in `main.py`, from **line 239** to **line 258**, meaning this code:

```
########################
# GAMELOOP (with a random approach)
########################

# PUT YOUR CODE HERE
# By default, this has a random algorithm.

########################
# SCORE
########################
```

Level 1

Consider as a purpose: "picking up cargo". We can assume this cargo is weightless, meaning that a drone can carry infinite amounts of cargo. The only thing the drone is picking up every cargo and doing this as efficient as possible.

In the simulation you?ll get a drone and you?ll be able to coordinate this drone. The purpose of your coordination is to reduce the distance needed to collect all cargo.

In the implementation, we already provided some code that already picks up every cargo, but it isn?t efficient at all (random).

When you run main.py, you can see some cargo appearing and then you see a drone flying around and ?picking up cargo?. Picking up is visualised as an upward arrow appearing on the cargo. In the code fragment you can see there

are three main parts of the code (that you need to know of). The initialisation part, where you can select the level and the amount of cargo. I?d recommend to change the amount of cargo to 10 while experimenting, and the speed to slow or even slowest. This way it?s easy to see what happens.

The actual coordination happens in the gameloop part, and you can immediatly see you have some methods at your disposal.

- `field.getCargoList()` returns a list with all cargo to be picked up.
- `droney.flyTo(x,y)` will fly to the x and y coordinate on the screen. (note that both are both are between 0 and 100)
- `droney.pickupCargo()` will pick up cargo at the current location if available.
- `field.endCondition(level)` is a method that for a certain level can help you decide when you?re done.

You see things inside an if, but you can actually ignore the largest part of this code. Then you have the score part, which will simply print your score when the simulation is over. The lower the score, the better your coordination is!

Level 2

(only read after completing level 1)

Level 2 is an extension to level 1. The main difference is that each cargo has a destination. Changing the level to 2 in the initialisation part and you will see `main.py` dropping of some cargo to certain places. (symbolised by a cargo with a downwards arrow).

Notice that you should use the `droney.dropOffCargo()` method now as seen in the example code.

Level 3

Level 3 is another extension on level 2. The difference now is that cargo isn?t weightless anymore. Every piece of cargo has a weight between 0kg and 1kg, and a drone has a carrying capability of 1kg.

Waypoints

We can transform a route you programmed into an actual route for the drone, using the file waypoints.py (right-click and choose 'Save link as'). Download it and put it next to your `main.py`.

To use it, you need to import it first, add the following line at the **top** of your file:

```
from waypoints import WaypointFile
```

So now it should look like this:

```
from waypoints import WaypointFile

import turtle
import random
import math
```

And then you can generate a waypoints file with these lines (somewhere at the bottom of your file):

```
f = WaypointFile.import_route(droney.route, "waypoints.txt")
f.write()
```

So it should look like this:

```
######################
# SCORE
######################

print("\n============================\nSCORE (lower is better)")
print(droney.score())

f = WaypointFile.import_route(droney.route, "waypoints.txt")
f.write()
```

As you can see, the only important method is `import_route(route, filename)`.

Each time you execute it, you will get also generate a `waypoints.txt` file. You can use it with APM Planner.

QGC format

(not important, skip this unless you're really bored and want to extend the functionality of ''waypoints.py'')

We will be exporting files in the QGC format:

```
QGC WPL <VERSION>
<INDEX> <CURRENT WP> <COORD FRAME> <COMMAND> <PARAM1> <PARAM2> <PARAM3> <PARAM4>
↪<PARAM5/X/LONGITUDE> <PARAM6/Y/LATITUDE> <PARAM7/Z/ALTITUDE> <AUTOCONTINUE>
```

Where:

- `<VERSION>`: just use 110
- `<INDEX>`: starts at 1, count up
- `<CURRENT WP>`: leave this at 0
- `<COORD FRAME>`: leave this at 3
- `<COMMAND>`: + 16: waypoint (also home?) + 82: spline waypoint + 20: Return To Launch (location doesn't matter) + 21: Land + 22: Takeoff
- `<PARAM1>`: ??????????????
- `<PARAM2>`: acceptance radius
- `<PARAM3>`: leave it at 0
- `<PARAM4>`: yaw at waypoint
- `<PARAM5/X/LONGITUDE>`: longitude, for example 51.1836405291789518
- `<PARAM6/Y/LATITUDE>`: latitude, for example 4.41937923431396484
- `<PARAM7/Z/ALTITUDE>`: height < 10m !
- `<AUTOCONTINUE>`: leave it at 1

CHAPTER 3

Contact

Please direct all your questions/suggestions to Evert Heylen on evertheylen@gmail.com.

Index