piDrones

Version 0.9 – 19th July 2018

Don't Do It!

The aim of this documentation is to share details that could be interesting or useful for other projects. It lacks details about how to do your own DIY drone.

My target was to test the capabilities of the Raspberry Pi and python. I chose creating a piDrone not as a flying machine as such, but as an intellectual challenge requiring hardware, software and electronics.

My DIY piDrones has cost me £20k+ of research, development and testing over 6 years. Even building a DIY drone based on the instructions below will cost >£1.5k yet it supports (a little) less than a professional drone costing less than £1k.

If you want a flying machine, just buy a DJI Mavic or Spark and get out there flying.

However, if you want to know how flying machines work, read on…

Internet Media

There are various links if you're interested:

- pistuffing.co.uk has everything done over 6 years including non-drone projects and even non-Raspberry Pi projects
- pidrone.io/blog links to only drone posts
- pidrone.io/video links to only the videos
- pidrone.io/posts/xxx links to specific xxx posts for twitter
- vimeo.com/pistuffing covers all videos directly.

Legal Constraint

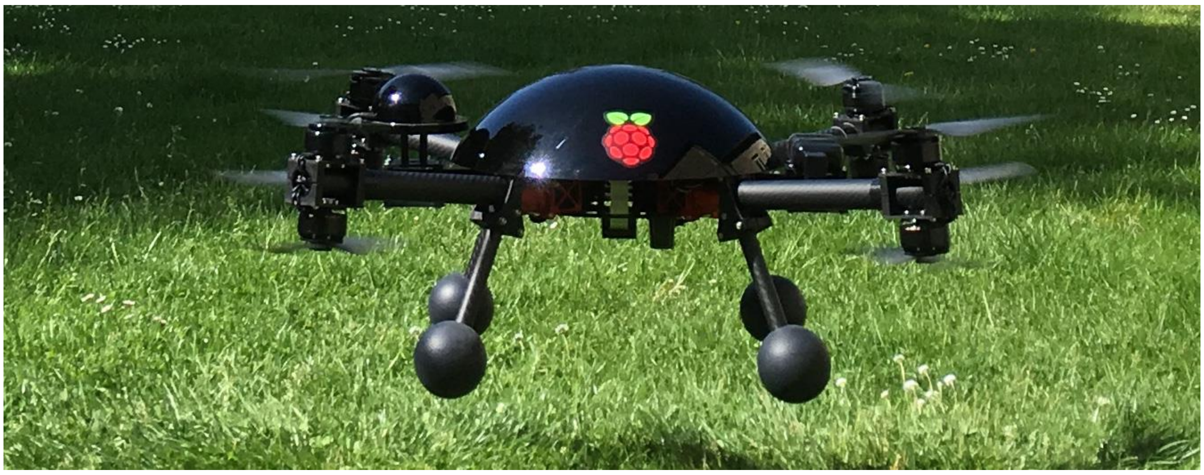
The project, defined by the GitHub pistuffing/Quadcopter content, is for non-commercial usage. If you want to use this project commercially, contact me via andy@pistuffing.co.uk and let's talk. After all, any financial compensation would be gratefully received!

Obviously, I'll happily accept voluntary donations from anyone at http://blog.pistuffing.co.uk/donations/ 😊

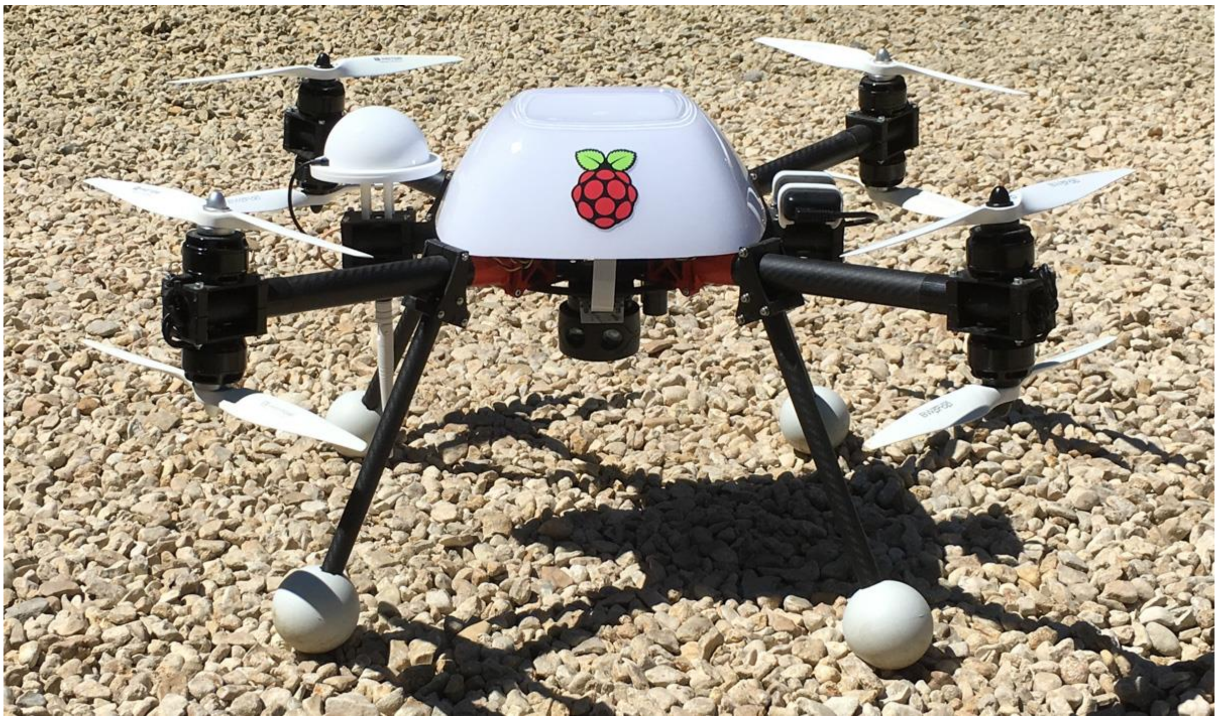
Raspberry Pis

The functions supplied by each piDrone depends primarily on the number of CPUs' performance and the sensors they carry.

Penelope (Pitstop) is a Raspberry Pi 3B+ 4 1400MHz CPUs, Stretch O/S piDrone supporting human-RC-, and autonomous file- and GPS-control flights. She has a custom lids designed by me and made by www.projectplastics.co.uk.



Hermione (Granger) is a Raspberry Pi 3B 4 1200MHz CPUs, Jessie O/S piDrone supporting human-RC-, and autonomous file- and GPS-control flights with object avoidance. Her lid is an inverted plastic salad bowl I've trimmed to size.

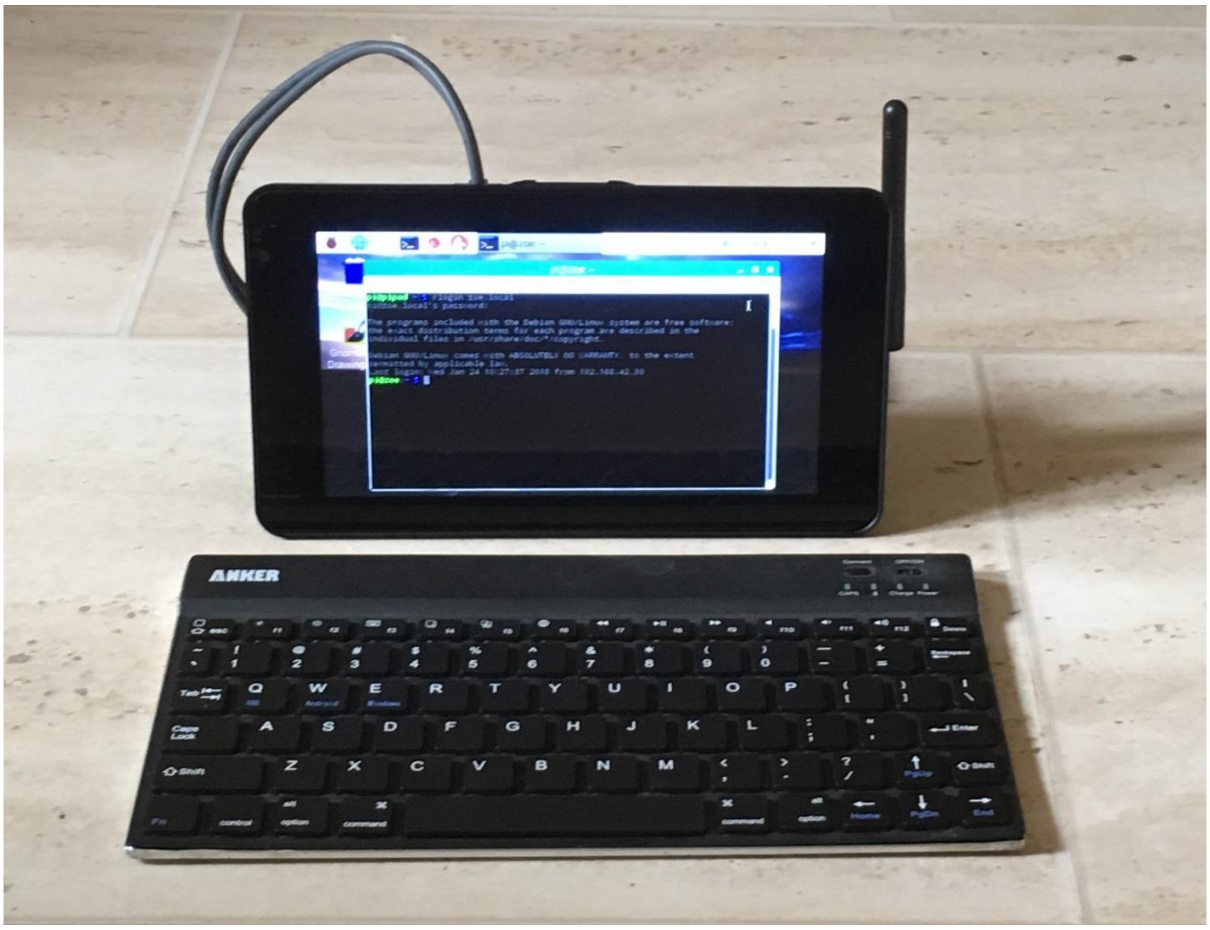


Zoë (Johnny Ball's daughter) is a Raspberry Pi0W single CPU, Stretch O/S piDrone supporting human-RC-, and file-control flights. She has a DIY lid with two 100mm clear dome and a tube from www.projectplastics.co.uk via ebay and joined, cut and painter myself.

The piDrones' naming convention is female (like ships and 'planes) and end with a phonetic '-ee'. No ideas why for the latter, it just emerged. Phoebe was my first, Chloë the second, both of whom sadly are deceased.

Ivy is a Raspberry Pi B3+ Stretch piDrone supporting the human RC (remote control). A B3+ is not necessary, it's only used as it was lying around unused. The case if a standard Ninja Pibow from pimoroni.com with a few custom extra layers to increase her thickness to include the joysticks.

A fourth Raspberry Pi, piPad, is required to remote login to the piDrones (Penelope. Hermione or Zoë) and RC (Ivy) for control their executable. It’s a standard Raspberry Pi case + screen.



```
pi@piPad ~ $ rlogin zoe.local
pi@zoe.local's password:

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed Jan 24 10:27:37 2018 from 192.168.42.30
pi@zoe ~ $ 
```

Hardware

The hardware frames are no longer available. You'll have to find your own.

Penelope uses eight T-motor CF 13x4.4 props, U3 KV700 motor and Air 40A ESCs, all available from electricwingman.com. Her frame is 75cm diagonally.

Hermione uses eight T-motor wooden 12x4.7 props (unavailable), U3 KV700 motor and Air 40A ESCs available from electricwingman.com. Her frame is 75cm diagonally.

Zoë uses four T-motor CF antiGravity! 9x3 props (unavailable), Air 2005/2000KV motors (unavailable) and Air 15A ESCs available from electricwingman.com. Her frame is 40cm diagonally.

Power

LiPo batteries are explosive if shorted or damaged. Treat them with utmost respect.

Hermione and Penelope have two power sources: a 5V 2A battery bank for the Raspberry Pi and sensors, and a 4S1P (four series, 1 parallel) 14.8V (nominal) 6600mAh power LiPo batteries.

Zoë has just the 3S1P (three series, one parallel), 11.1V (nominal), 2300mAh power LiPo battery, along with a 5V 2A voltage regulator to run the Raspberry Pi and sensors. There is an internal switch to run just the Raspberry Pi and sensors to passive testing.

Sensors

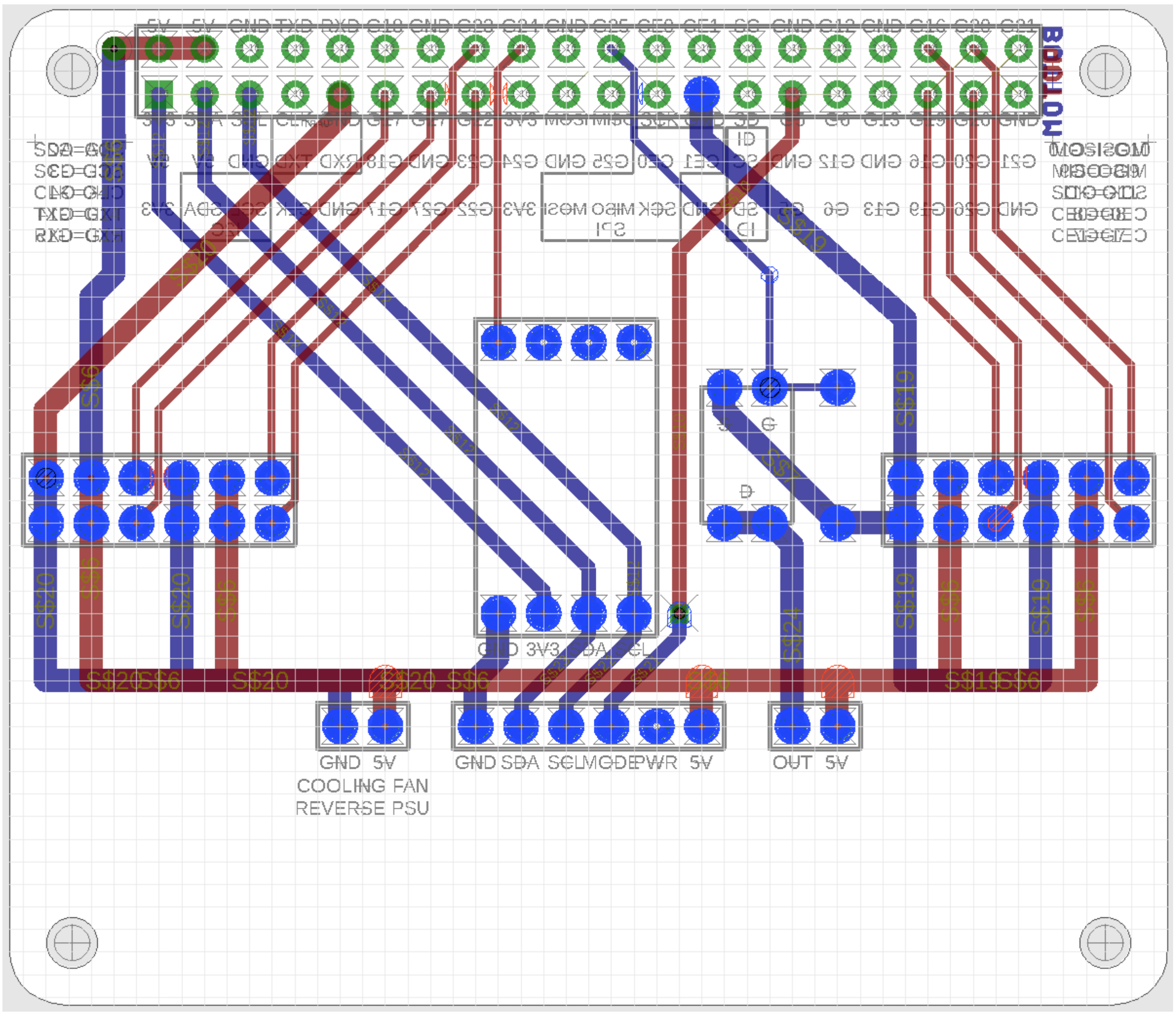
Here's the list of sensors used:

- Inversense MPU-9250 IMU is for the accelerometer, gyrometer and magnetomer sensors and is available from pimoroni.com
- Garmin LiDAR-Lite v3HP is for the ground-facing vertical distance tracking and is available from robotshop.com
- RPi camera is for ground-facing lateral distance tracking and is available from pimoroni.com
- u-blox NEO-M8T is for GPS tracking and is available from drotek.com.
- Grayhill 67a is for RC joysticks and is available from avnet.com.
- ABI-009-RC buzzer is available from uk.farnell.com.
- IRLD024PBF MOSFET is for the beeper-switching and is available from uk.farnell.com.
- RECON R_78D5.0-2.0 5V 2A power supply is available from uk.farnell.com.

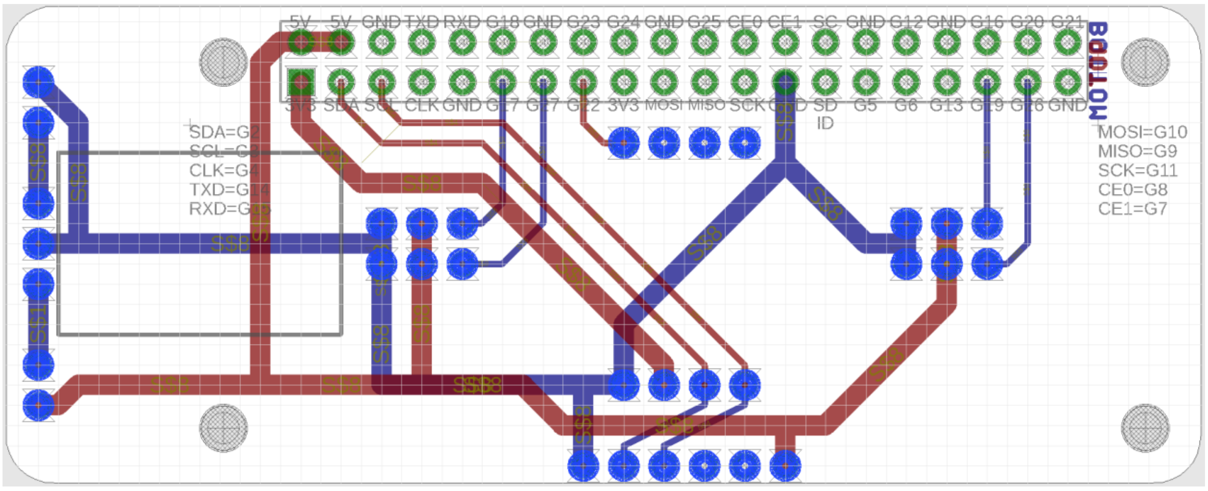
PCBs

Eagle (autodesk.com – free use for individuals) is used to generate the CAM files required by ragworm.eu to generated the PCBs. They are available on GitHub/com/pistuffing/quadcopter/ as the following brd (board) files.

RPi3-PCB.brd is for a Raspberry Pi 3 supporting 4 or 8 motors. The IMU is on the middle with eight ESC connection sockets. The Garmin LiDAR-Lite v3HP is bottom center with 5v and output pins either side for constant and switchable 5V outputs respectively. The latter is currently used for the beeper.



RPi0-PCB.brd is for a Raspberry Pi0W supporting 4 motors. The IMU is on the middle with four ESC connection sockets. The Garmin LiDAR-Lite v3HP connection is to the right with the 5V regulator at the left.


5V 5V GND TXD RXD G18 GND G23 G24 GND G25 CE0 CE1 SC GND G12 GND G16 G20 G21
3V3 SDA SCL CLK GND G17 G27 G22 3V3 MOSI MISO SCK GND SD ID G5 G6 G13 G19 G26 GND
BOTTOM
SDA=G2
SCL=G3
CLK=G4
TXD=G14
RXD=G15
MOSI=G10
MISO=G9
SCK=G11
CE0=G8
CE1=G7

Math(s)

I found this article useful reminding me of long forgotten trigonometry and rotational matrices: https://scholarsarchive.byu.edu/cgi/viewcontent.cgi?article=2324&context=facpub

Processes

This is the five process layout for an autonomous system:

```
+-------+
|Sweep|--->---+
+-------+     |
          +-------+---+    +--------+
          |Autopilot|-->--|Motion|
          +-------+---+    +---+--+
+---+             |            |
|GPS|---->-----+                |
+---+                           |
                                |
               +------+         |
               |Video|-->--+
               +------+
```

This is the two process layout for a human-RC system:

```
+-------+
|  RC   |.......
+-------+      .
               .
               .
               .
           +---+--+
           |Motion|
           +---+--+
               |
               |
+-------+      |
|Video|--->--+
+-------+
```

The difference is defined by the code constants autopilot_installed or rc_installed. The Pi0W is only capable of the latter due to the single CPU.

Code

The code is a single file, Quadcopter.py. It's 'self-documenting', split into

- 1700 comments
- 900 blanks
- 2700 lines of code.

The code structure from top to bottom is:

- I^{2}C
- Sensors
- PIDs / ESCs
- Trigonometric and rotation matrices
- Control function
- *Process & *Manager pair control process management
- class Quadcopter().

The code lives at https://github.com/PiStuffing/Quadcopter/blob/master/Quadcopter.py. In "class Quadcopter():", start at "while self.keep_looping:" in "def fly()" which runs the flight itself; dig in the details from there.

ESCs

Electronic speed controllers (?) are the firmware units which translate from Raspberry Pi PWM signals to the power from the motors. The PWM signal is from 1000us to 2000us signals with 1us resolution, signaled up to 500Hz. For this reason, software PWM is insufficient as supplied by RPi.GPI and instead, the PWM python module RPIO must be used.

1000us pulses synchronize with the motors and stops them whining.
1150us pulses starts the motors at minimum rotation rate.
1500us pulses are approximate the hover value.
1999us pulses at full speed.

The middle pair listed above require tuning based on the ESCs and motors used.

Installation

Basic Confirguration

- Use Etcher to write the SD card with the latest Raspian Lite image
- In windows in the root directory of the SD card, create an SSH file to enable it

```
echo hello > ssh
```

- Boot up your RPi with SD card, keyboard, mouse, monitor and WAP / soft AP supporting WiFi dongle installed (no WiFi dongle needed for 3B and Zero-W)
- **`Sudo raspi-config`** to set hostname, GPU memory (128MB), enable I^{2}C, overclock to 1GHz, disable overscan, disable serial UART, enable camera support and SSH server. Do this incrementally to make sure each change takes effect in /boot/config.txt
- **`sudo apt-get update`**
- **`sudo apt-get dist-upgrade`**
- **`sudo apt-get install i2c-tools python-smbus python-dev python-setuptools python-picamera python-gps python-serial git ftp`**
- edit /boot/config.txt adding

```
dtparam=i2c_arm_baudrate=400000
```

- while in /boot/config.txt, disable pointless functions thus

```
dtparam=audio=off
start_x=0
```

- If you are using a Pi3 or Zero-W, you may want to disable Bluetooth in /etc/modprobe.d/raspi-blacklist.conf thus:

```
# WiFi
# blacklist brcmfmac
# blacklist brcmutil
# Bluetooth
blacklist btbcm
blacklist hci_uart
```

Likewise if you're using an WiFi USB dongle, you can disable the internal one by removing the '#'s for that pair of lines.

- If you are using a Raspberry Pi 2 B V1.2, it can be overclocked to 1.2GHz thus in /boot/config.txt

```
arm_freq=1200
core_freq=500
sdram_freq=600
force_turbo=1
over_voltage=6
```

- If you are using multiple USB UART devices (i.e. GPS and Scanse Sweep in Hermione's case), it's necessary to link the USB /dev/ttyUSB* to a unique name for each. In /etc/udev/rules.d add a file such as '99-usb-serial.rules containing something like this:

```
SUBSYSTEM=="tty", ATTRS{idVendor}=="0403", ATTRS{idProduct}=="6015",
ATTRS{serial}=="DO004VY5", SYMLINK+="ttySWEEP"
SUBSYSTEM=="tty", ATTRS{idVendor}=="067b", ATTRS{idProduct}=="2303",
SYMLINK+="ttyGPS"
```

The details of what to fill in come from /var/log/messages as a USB UART is plugged in. Also /etc/default/gpsd and Quadcopter.py to reference these tty* names instead of ttyUSB*. More details here: http://hintshop.ludvig.co.nz/show/persistent-names-usb-serial-devices/

- Add hardware PWM

```
git clone -b v2 https://github.com/metachris/RPIO
cd RPIO
```

check mailbox.c code exists in source/c_pwm/mailbox.c to conform you have the right version, then install thus

```
sudo python ./setup.py install
cd ~
```

- Add I^{2}C variant for RC joysticks

```
git clone https://github.com/kplindegaard/smbus2
cd smbus2
sudo python ./setup.py install
cd ~
```

- Add Quadcopter code

```
git clone https://github.com/PiStuffing/Quadcopter
cp Quadcopter/qc.py .
cp Quadcopter/pidrone .
cp Quadcopter/fp.csv .
cp Quadcopter/Quadcopter.py .
cd ~
```

Wireless Access Point Network Configuration

This is based on https://www.raspberrypi.org/documentation/configuration/wireless/access-point.md.

Install required software elements:

- `sudo apt-get install dnsmasq hostapd`
- `sudo systemctl stop hostapd`
- `sudo systemctl stop dnsmasq`

Set up the static IP address:

- `sudo vi /etc/dhcpcd.conf`

```
interface wlan0
    static ip_address=192.168.42.1/24
```

- `sudo service dhcpcd restart`
- `sudo mv /etc/dnsmasq.conf /etc/dnsmasq.conf.org`
- `sudo vi /etc/dnsmasq.conf`

```
interface=wlan0
usually wlan0
    dhcp-range=192.168.42.2,192.168.42.20,255.255.255.0,24h
```

Set up the access point:

- `sudo vi /etc/hostapd/hostapd.conf`

```
interface=wlan0
driver=nl80211
ssid=HoGWAP
channel=1
wmm_enabled=0
wpa=1
wpa_passphrase=HoG3.141592654
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
auth_algs=1
macaddr_acl=0
```

Enable hostapd:

- `sudo vi /etc/default/hostapd`

```
DAEMON_CONF="/etc/hostapd/hostapd.conf"
```

Comment out any "NETWORK={" in /etc/wpa_wpa_supplicant/wpa_supplicant.conf.

Edit /etc/hostname to ensure the domain name is included - in my case, the domain is called local, and the hostname is pidrone, so /etc/hosts reads

- `pidrone.local`

Next assign static IP address for the piDrone, where pidrone is zoe, hermione or penelope.

- sudo vi /etc/hosts

  ```
  127.0.1.1 pidrone.local
  192.168.42.1 pidrone.local pidrone
  ```

Check, double check, and triple check that you've done all the above steps, and then finally

- `sudo reboot`

To backout temporarily to allow for O/S updates etc

- `sudo vi /etc/default/hostapd`

  ```
  DAEMON_CONF=""
  ```

- Reinstate any "NETWORK={" enties in /etc/wpa_supplicant/wpa_supplicant.conf
- Comment out the AP interface IP details in /etc/dhcpcd.conf
- reboot

Flights

Run the piDrone with the main command-line sh script:

```
./piDrone
```

This enforces python optimization thus:

```
sudo python -O ./qc.py
```

This then calls the library Quadcopter.py. The method means the library with actually run the optimized Quadcopter.pyo if available.

The following are the primary commands; fine tuning parameters are available.

Initial Setup

`--tc 1 -h 1150` turns each blade one by one so you can check all are working, and are rotate the connect direction. My code assumes the top front left prop is anti- (counter-) clockwise.

`-d` diagnostics logged to qcstats.csv which when fed into a spreadsheet (e.g. Excel) can show graphs about all key parameters from sensors and resultant actions.

`--cc` checks or calibrates magnetometer i.e. a 3D compass

Autonomous File Flight Control

`-f flightplanfile`

The flightplanfile is based on a series of lines like this while flies on a horizontal square. Takeoff and landing are automatic:

```
# - evx is fore / aft velocity in m/s; fore is positive
# - evy is port / starboard velocity in m/s; port is positive
# - evz is up / down velocity in m/s; up is positive
# - time is how long to maintain that speed in seconds
# - name is an arbitrary name put out to console to show flight progress

# evx, evy, evz, time, name

 0.0,   0.0,   0.0,   1.0, HOVER
 0.25,  0.0,   0.0,   4.0, FORE
 0.0,   0.0,   0.0,   1.0, HOVER
 0.0,  0.25,   0.0,   4.0, PORT
 0.0,   0.0,   0.0,   1.0, HOVER
-0.25,  0.0,   0.0,   4.0, AFT
 0.0,   0.0,   0.0,   1.0, HOVER
 0.0,-0.25,   0.0,   4.0, STARBOARD
 0.0,   0.0,   0.0,   1.
```

Autonomous GPS Flight Control

`--cwp` – clear previous set of GPS waypoints
`--awp` – add another GPS waypoint to the list
`--gps` – fly from an arbitrary take-off locations to the series of preset waypoints in turn

Manual File Flight Control

`-f flightplanfile`

The flightplanfile is based on a series of lines like this while flies on a horizontal square. Takeoff and landing are automatic:

```
# - evx is fore / aft velocity in m/s; fore is positive
# - evy is port / starboard velocity in m/s; port is positive
# - evz is up / down velocity in m/s; up is positive
# - time is how long to maintain that speed in seconds
# - name is an arbitrary name put out to console to show flight progress
# evx, evy, evz, time, name

 0.0,   0.0,   0.0,   1.0, HOVER
 0.25,  0.0,   0.0,   4.0, FORE
 0.0,   0.0,   0.0,   1.0, HOVER
 0.0,  0.25,   0.0,   4.0, PORT
 0.0,   0.0,   0.0,   1.0, HOVER
-0.25,  0.0,   0.0,   4.0, AFT
 0.0,   0.0,   0.0,   1.0, HOVER
 0.0, -0.25,   0.0,   4.0, STARBOARD
 0.0,   0.0,   0.0,   1.0, HOVER
```

Manual RC Flight Control

`--rc`

The piDrone is booted up first to establish the WAP. They the RC and control RPi are booted, and connected to the piDrone WiFi. From in the control RPi, via a pair of terminal windows, rlogin to the piDrone and the RC. Start the piDrone (./pidrone) and the RC (python ./RC.py). When you type in "—rr" on the piDrone command line, the RC and piDrone connect, and an auto-takeoff takes place. Thereafter, the RC is under your control. To trigger an auto-landing, push the joysticks inwards towards each other until the landing is initiated; the landing height is also automated.

What next?

Any further development is primarily restricted by WiFi and battery constraints: WiFi range is 40m maximum, and for the larger models, batteries last a few minutes maximum; without these resolved, nothing further is possible.

Assuming those could be resolves, then:

- include higher resolution video lateral tracking with a future version of the Raspberry Pi – there's a balance between IMU and Video processing; the two need to be in sync – more CPU speed would allow higher Video resolution without a synchronization hit
- long distance flights with a 2nd camera for taking photos / video
- fuse GPS with IMU and LiDAR / Video for lateral and vertical measurement when unavailable for current set e.g. flying over a lake or altitude high than range of the LiDAR.

Fundamentally though, that ain't gonna happen without the primary constraints resolved, hence this is the end on of this project for now.

"So long, and thanks for all the fish!"