An unofficial Bebop drone hacking guide 1.6

Latest update: 1 June 2016

Table of contents

Foreword

Even if we tried to explain each procedure in detail, this guide is intended for advanced users who have some background in unix shell commands. If you are not in this category, use Google and build yourself a geek culture before reading further. Modifying the Bebop files in the onboard memory can cause serious damage and make your precious aircraft unusable. Furthermore, be aware that irreversible changes to the onboard software invalidate the warranty.

We take no responsibility for any damage
you may cause to your bebop by hacking into its internal memory.
USE THIS GUIDE AT YOUR OWN RISK

Most of the information in this guide can be found in different forums, websites and languages. As users, we felt the need to cluster as much information as possible in one place. This was originally meant to be a personal record of useful information, and only subsequently evolved in a more structured text that we thought could be useful to others.
We herewith acknowledge all the Bebop users who have shared their tests and comments online; we apologize for not mentioning the names of these pioneers. This was often impossible because we have edited the original information, merged suggestions coming from several different sources and added our own tests. At any rate, if you recognize your work in this guide, post a comment and we will be happy to acknowledge your name in the corresponding section. For the same reasons we do not hold any right on this text, and anyone is free to copy, distribute and edit this guide.
It is crucial to underline that the information gathered in this guide does not come from Parrot, but rather from independent experimentation of Bebop users largely based on a trial-and-error approach. The whole content of this guide should therefore be considered as an educated guess of what is going on inside the Bebop software and how the user can influence it. We wrote here what we and others have understood, hypothesized and experimented this far. Consequently, keep in mind that what you read in this guide might not be true nor applicable to your own case.
All comments, corrections and additions to the guide contents are welcome. To send your comments please reply to the post where you found the link to this file. You will be contacted in case your suggestions involve extensive modifications.
Most of the mods discussed in this guide were originally tested on the Bebop drone, running ARDrone3 version 2.0.57. Following Bebop 2 introduction and the new ARDrone 3 release (3.x) for both models, we are now testing and updating the mods for the new firmware/hardware. Specific remarks in the text indicate which firmware/hardware version a particular mod is known to work with. In any case, be aware that some of the mods might not work with the new firmware: please report to the forum any issue and possible correction.

1. Before we start

The Bebop user guide provided in the box or available for download from the Parrot web site gives rather succinct information on several aspects of the Bebop software and hardware functioning. This makes sense, if one considers that Parrot does not want to scare their customers off with complex manuals and rather present their product as an easy to fly drone. Nevertheless, as soon as you start piloting the Bebop you realize there is a lot more going on under the cover than you are aware of. If you are willing to experiment a bit and customize the Bebop controls, some of this information is crucial for you to anticipate the effect of any modification you may introduce to the software.

1.1. The Bebop operative system

The Parrot Bebop is essentially a battery operated computer, specialized in flying and shooting video. The Bebop OS, called ARDrone3, controls a number of devices and interacts via wifi with a remote controller, and ultimately the pilot. ARDrone3 is a customized UNIX-based system (in an Android incarnation). As such, it maintains the typical UNIX filesystem hierarchy as well as its open architecture, allowing any user to introduce changes to practically anything, which is both exciting and very dangerous to do on a pricey flying machine.
All of the instructions that allow ARDrone3 to interact with on-board sensors and actuators, as well as the control device, are stored in the directories that populate the Bebop internal memory. This is where you want to connect in order to introduce any change and customize the Bebop behavior to your own needs. The way into the Bebop memory is through its wifi connection and the use of communication protocols such as *telnet* and FTP.
In short, before you take any step into Bebop hacking, get yourself a minimal expertise in UNIX shell and FTP commands.

1.2. The Bebop filesystem

A complete list of the Bebop filesystem can be obtained after connecting to the drone via *telnet* (see section 2) and typing:

```
ls -R
```

Most of such files and directories are anyway of little interest to the scope of this guide. The following paragraph offers an inexhaustive list of the files and directories that have shown useful so far.
It is important to stress that since the 3.0 release of ARDrone, the whole filesystem is write-protected. As a consequence, before introducing any change to the onboard files, you need to change the filesystem permissions to read/write (see also par. 4.4). This can be easily done from a *telnet* connection (see section 2)

1.3. Files and directories of interest

The Bebop filesystem contains a number of configuration and service files, located in different directories. Once you access the filesystem every change you introduce affects the software performance. It is therefore vital to understand the function of each file and directory before stepping onto this dangerous ground.
Editing a configuration file introduces irreversible changes: you will only be able to restore the original condition by changing back the configuration file to its previous form.

WARNING: Be extremely careful before you edit such files,
because the risk of bricking your drone is very high.

`/bin/onoffbutton` is a directory containing four shell scripts that are run by pressing the Bebop power button. See section 4 to learn how to edit and expand this set of scripts.

`/data/dragon.conf`
This file stores all the values for the FreeFlight controller software settings. Since these parameters can be set through the FreeFlight software, editing this file makes little sense, but it can be useful to make a backup copy of your settings, just in case.
It may anyway be interesting to edit the following:

`"absolute_control"` sounds like a very intriguing hint at a piloting mode where the controller stick commands are not relative to the drone orientation but 'absolute', or rather relative to the pilot position: moving the left stick forward pushes the drone away from you and pulling the stick moves the drone closer. Unfortunately this feature -which is available for other drones - does not seem to be implemented in the current version of the control software, so we do not recommend changing this line. Hopefully future releases could include this option.

`"picture_format" : 0,` changes the photo recording options so that when you take a picture you will get both a JPG and a raw DNG of the fisheye. Such a choice cannot be operated from the FreeFlight settings window, and as soon as you change the recording options through the software, *dragon.conf* will be modified accordingly.

`"preferred_home_type" : 0,` suggests the "return to home" destination can be set (e.g. to the take-off location, rather than current controller position). This has not been tested yet.

`/data/ftp`
This is the directory you can access via FTP protocol. It originally contains a single directory called `internal_000`.
The ftp directory can also be used to make backup copies of file that you want to edit or download to your personal computer via FTP (see section 4).

`/data/ftp/internal_000` contains several interesting directories:

`/Bebop_Drone`	contains three directories where the Bebop stores your Drone Academy data (`academy`), videos and photos (`media`) and the tiny preview icons that appear in FreeFlight (`thumb`). You can easily download these files to your personal computer using the FTP command *get*.
`/Debug`	contains log files for the latest power-on sessions (see par. 5.6).
`/flightplans`	is where your flight plan data are uploaded from the FreeFlight app
`/gps_data`	contains .bin and .md5 files with GPS information
`/log`	contains lengthy files with feedback messages generated by the onboard software. Likely more useful for developers than end users.

The use of the `/data/ftp` directory is anyway not completely straightforward, and before you use this directory to store files that you may want to keep onboard, do read the following.
As it turns out, `/data/ftp` only has about 8MB of allocated space, so it fills up quickly and when it does, config data like FreeFlight settings or magnetometer calibration data cannot be written (they just get skipped with no obvious notification of the failure). This may cause all kinds of issues (e.g. hard landings with bouncing was shown to be the result as an empty magnetometer calibration file being written). A secondary issue is that files that if you copy a file to `/data/ftp` from a different location, the file will be truncated (without notice) as soon as the 8MB limit is reached.
Now for the good news. The `/data/ftp/internal_000` directory accesses memory allocation on a different device (/dev/mmcblk0) than `/data` and has way more allocated space, about 8GB. This is in fact where recorded videos and photos are stored, and can more appropriately be intended as a user managed storage area.
The only issue is that if for any reason you reset the Bebop (see section 5), `/data` (and its underlying structure) and `/data/ftp/internal_000` (and its underlying structure) get wiped and re-initialized to factory default config - meaning all user stored data is gone.
For these reasons, `/data/ftp/internal_000` should be the directory to use for writing debug info or other sort of output data such as live video record, raw video, etc (see par. 3.3).

`/data/system.conf`
This file contains a few system parameters. We recommend to leave the file untouched.

`/etc/debug.conf`

This configuration file contains a number of options, originally useful for Parrot developers during software debugging. The options set in this file include blackbox function (see par. 4.3.6) and can be activated by editing the file or by launching the script:

```
/usr/bin/DragonDebug.sh
```

`/etc/default-dragon.conf`
`/etc/default-system.conf`
These files contain the factory presets that are restored upon hard resetting the Bebop (see section 5). DO NOT TOUCH THESE FILES

`/etc/init.d/rcS`
This file is run during the Beebop booting process and controls a number of features that can be edited (see par. 4.3.5).

`/etc/init.d/rcS_gps`
This file is also run during Beebop boot-up and sets the behavior of the GPS chip, including where GPS data are saved (see par. 4.4.4).

`/tmp/gps_nmea_out`
This file is created during each bootup by `/etc/init.d/rcS_gps` and gives you a live display of the NMEA records the Furuno GPS is currently sending while you are watching.
The following command:

```
cat /tmp/gps_nmea_out | grep GNRMC
```

will give you "the Recommended Minimum Navigation Information". The free NMEA online Decoder (http://freenmea.net/decoder) can be used to view a graphical representation of the Bebop raw GPS track.
Also see par 4.5 for automatically saving and retrieving full GPS data.

`/usr/bin/BLDC_Test_Bench`
Runs control tests on the brushless motor controllers (see par. 3.5.1)

`/usr/bin/diagnostic`
Runs a sensor diagnostic test and outputs the results (see par. 3.5.2).

`/usr/bin/dos2unix`
This is a utility that converts files to unix-compatible format (see par. 3.5.3)

`/usr/bin/dragon-prog`
Dragon-prog is the main Bebop process governing the whole system of flight control and data management. This is the process you interfere with if you want to change the Bebop behavior (see par. 3).

`/usr/bin/DragonStarter.sh`
This is a script that starts dragon-prog “and stop what need to be stopped when it stops..." (sic!). The first lines contain succinct instructions on the options that can be activated when launching this script:

```
echo " Dragon Starter start dragon prog and stop what need to be stopped when it stops..."
echo " By default, core dump are enabled."
echo " Options :"
echo "   -h             show this help"
echo "   -nfs           Use nfs dir as base directory"
echo "   -prog <prog>   Use <prog> instead of default prog \"$DEFAULT_PROG\""
echo "   -gdb           Start dragon with gdbserver"
echo "   -nogdb         Do not start dragon with gdbserver"
echo "   -out2null      Send std and stderr output to /dev/null"
echo "   -noout2null    Do not send std and stderr output to /dev/null"
echo "   -nocoredump    Do not enable coredump"
echo "   -coredump      Enable coredump"
echo "   -B             Enable Blackbox"
echo "   -b             Disable Blackbox"
echo "   -N             Enable Navdata"
echo "   -n             Disable Navdata"
echo "---------"
```

The following line is of particular interest:

```
POST_CMD=""
```

dragon-prog option strings can be placed between the double quotes to permanently change *dragon-prog* execution (see section 4).

`/usr/sbin/bcmwl`
This executable controls the whole wi-fi connectivity of the Bebop. Launching:

```
bcmwl -h
```

Produces a long help screen including all the commands that can be run from within bcmwl.

`/www/index.html`
Even if an HTML server is not active by default in the Bebop, the www directory contains this file, where you can find a long list reporting information on the Bebop software version, build date, compiler, etc.

1.4. Camera

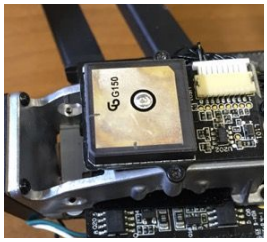
The main Bebop camera (also called horizontal camera) faces forward and is the camera you see in the front of the Bebop nose. It is equipped with a fish-eye lens, but the camera will normally grab and stream only part of the fisheye field of view. This trick allows the amazing Bebop image stabilization (which is performed via software and image manipulation algorithms; option `-S`) as well as the capability to virtually orient the camera up, down left or right.

The camera sensor is turned vertically, not horizontally as in most digital cameras. As a consequence, raw video appears turned by 90° and has to be rotated via software before the live stream is sent over to your device. This is achieved via a process called *reprojection* (options `-R`; `-I`). The original orientation and reprojection are important aspects to keep in mind when changing the camera resolution, because image formats such as 1024x768 refer to vertical (1024) and horizontal (768) dimensions of the image (not horizontal x vertical as in most digital cameras or displays).
As in any digital camera, the exposure time, white balance, gain values, etc. can be set automatically or manually. Several *dragon-prog* options (see par. 3.3.2) concern this kind of adjustments (e.g. `-m`; `-A`; `-E`).

1.5. Sensors

The onboard software controls Bebop flight by combining inputs from several sensors. These include:

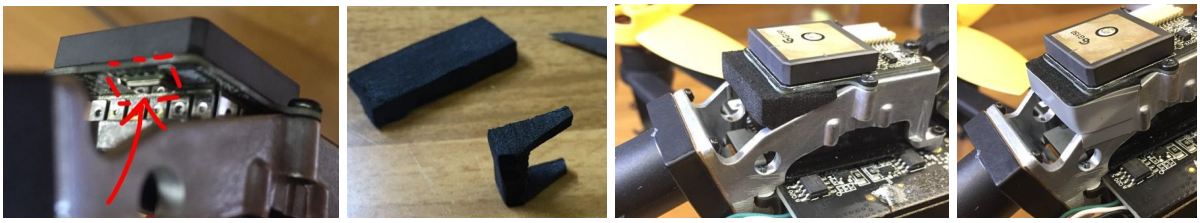
1.5.1. GPS chip



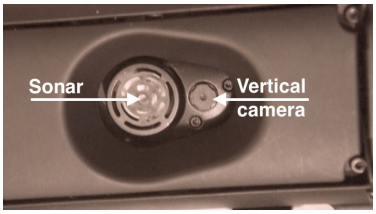
After removing the Bebop plastic nose, the Furuno GN-87F GPS chip (or Ublox Neo 8M in Bebop 2) is located on top of the camera mount block. The ***antenna*** is directly plugged on the chip.
The GPS connects to several satellites to determine the Bebop position in latitude, longitude and altitude. The use of different satellite networks can be changed by editing a configuration file (see section 4).

1.5.2. Barometer

This sensor is fixed to the same board that hosts the GPS, but on the lower side. The sensor is easy to recognize as a ***small metal component*** (about 2x5mm). The MS5607 barometer mounted onboard the Bebop has a known problem with light: when hit by strong light, the barometer undergoes a sudden shift. This generates important altitude changes when flying between lit and shaded areas (e.g. among trees or buildings). The acknowledged solution is to cover the barometer with a small piece of dense ***black foam*** and fix the foam to the GPS board or the metal scaffold using tape. The barometer chip is used by the Bebop software to estimate relative changes in altitude that are too small to be measured by the GPS and compensate for unwanted vertical drifts, normally due to wind.



1.5.3. Sonar



This sensor is located under the belly of the Bebop, next to the small lens of the vertical camera. It is integrated to the main board. It is used to measure the Bebop height from the ground or any ultrasound-reflecting surface. The sonar range is limited to 8mt; above this height the GPS and barometer take over. The sonar grid should be kept clean of dust and particles and never covered.

1.5.4. Vertical camera

Bebop drones have a second second camera pointing down to the ground (vertical camera), which is used by the Bebop software to maintain the drone position when hovering below an altitude of 8 mt. Above this altitude, the Bebop will only rely on its GPS sensor.
You can find the tiny camera under the belly of your Bebop, just beside the ultrasound sensor (used for ground distance measurements). The vertical MT9V117 camera is reported to shoot a frame every 25 ms. By comparing each frame with the previous one, the software adjusts the Bebop orientation and X-Y position. Since such images have only a temporary use, they are not normally stored in the Bebop memory. Nevertheless, *dragon-prog* can be set to save a selection of these images (*dragon-prog* option `-D`).

2. Connecting to the Bebop

In order to introduce any temporary or stable change to your Bebop behavior you need to access its internal filesystem. This can be done through the Bebop own wifi network, via standard protocols such as telnet or FTP. Please refer to online information about the use of each of these protocols.

2.1 Requirements

- A fully functional Parrot Bebop Drone
- A device (personal computer, tablet or smartphone) with wifi connectivity*

  * *specific software may be required for portable devices*

2.2 Setup

- turn on you Bebop by pressing on the power button. As the cooling vent starts spinning (ARdrone3 2.x) or restarts for the second time (ARdrone 3 3.x), the Bebop wifi network should be operational. Turn on wifi on your device to connect to the Bebop wifi (no password required).
- If your drone is running ARDrone v. 3.x, the telnet daemon is not running at startup. To launch it you should press the Bebop power button four times in a quick sequence (see par. 4.4).
- You can now start a telnet session:

  OSX: go to /Applications/Utilities/ and launch Terminal.

  Windows: Click Start. Enter *cmd* in the Search field in the Start menu. A command prompt is displayed.

  Unix-based OS: open a shell window.

  Mobile devices: you need to install a telnet or FTP capable application. Find one in your app store and follow its instructions to open a telnet session.

2.3 Connecting via telnet

Once you have opened a shell window and started the telnet daemon via power button, type:

`telnet 192.168.42.1` and press ENTER

The display will output a few lines meaning you are now connected to the internal Bebop memory and you are able to explore its directories.
As we mentioned above, since the 3.0 release of ARDrone, most of the filesystem is write-protected, with the esception of part of the `/data` directory. As a consequence, before introducing any change to the onboard files, you need to change the filesystem permissions to read/write (see also par. 4.4).

To do so, just type:

```
mount -o remount,rw /
```

WARNING: you now have full permissions to edit, move or delete any file.
THIS CAN BRICK YOUR BEBOP!!!
If you have are not familiar with unix or have any doubts STOP HERE

FURTHER WARNING: you should be aware that after the remount command, any file copies you make get read/write permissions by default. So, if you make copies and want the copied files to work like the originals, you must first check file permissions of each original file and then set the same permissions to the copied file (e.g. `chmod 775` for exacutable scripts, see par. 4.2.2).

2.4. Connecting via FTP
In a new shell window, type:

```
ftp 192.168.42.1
```
and press ENTER

You will be asked to login. Just press ENTER again. You now have access to the `/data/ftp` directory in the Bebop filesystem (see section 1).

Due to factory settings, FTP access does not allow you to navigate the whole filesystem, but only the *ftp* directory, which is where images and videos are recorded, together with other useful files. Please see par. 1.4 for more details and refer to a guide to FTP commands (Google can help you with that) to know how to manage files and directories via the FTP protocol.
The limited FTP access can be taken over by editing the file `/etc/inetd.conf` (you must first enable read/write permissions as described in the previous paragraph). After the following lines, where the first number is the TCP port:

```
21 stream tcp nowait root ftpd ftpd -wSS /data/ftp
51 stream tcp nowait root ftpd ftpd -wSS /update
61 stream tcp nowait root ftpd ftpd -wSS /data/ftp/internal_000/flightplans
```

just add:

```
71 stream tcp nowait root ftpd ftpd -SS /
```

Since the next reboot,

```
ftp://192.168.43.1 71
```

will give you FTP access to the whole filesystem. Make sure you omit the `w` in the added line, to prevent the authorization to overwrite any file via FTP. Use with caution!!!

2.5. Retrieving flight data

Since we have seen how to access the drone memory, it may be interesting to download the flight data that it stores. There are two sets of files that provide details about your Bebop's flight. The first is positional data in the form of *pud* files. A new *pud* file is written on every flight but they are all deleted at startup, so if you need the data, get it before shutting down the drone.
A second type of files record more detailed information but only appear after you turn on an option called *blackbox* in the drone internal configuration files (see par. 4.3.6).

2.5.1. Downloading pud files

Low resolution flight data are recorded for every flight in files that bear a .pud extention and are saved in:

```
/data/ftp/internal_000/Bebop_Drone/academy/
```

To download pud files, just open an FTP session, navigate to:

```
/internal_000/Bebop_Drone/academy/
```

Take a look at the directory contents with *ls* and use the *get* command to download the file you want to your home directory on your PC. For example:

```
get 0901_2016-04-02T193216+0000_B45B95.pud
```

Such pud files can be used with the beautiful FlightData Manager tool (unfortunately only available for windows PC), which allows you to ***overlay flight track*** and other data to your recorded videos. Also refer to this clear post on the Parrot blog.

3. Reversibly changing dragon-prog options

This section explains how to modify the Bebop behavior in a reversible way: switching off and on again the Bebop will reset it to its original setup.
In order to change the Bebop behavior you must stop the Bebop control software called *dragon-prog* and relaunch it with the desired options. Before doing anything, read carefully paragraph 3.3, where all known options and their effects are listed.

3.1 Killing the dragon

Once you know what options you want to activate, proceed with the following.
Connect to the Bebop via telnet (see section 2), then type:

```
ps
```
and press ENTER

A list of all active processes will be displayed. Search for the *dragon-prog* process, which should be somewhere between lines 600-800 (highlighted in bold in the example below) and remember the corresponding process number.

```
  658 root       0:00 [flush-ubifs_0_0]
  659 root       0:00 [flush-ubifs_2_0]
  660 root       0:00 [flush-179:0]
  672 root       0:00 poll_file -w /sys/devices/platform/ci_hdrc.0/udc/ci_hdrc.
  677 root       0:00 /usr/bin/bcm-watchdog
  680 root       0:00 {DragonStarter.s} /bin/sh - /usr/bin/DragonStarter.sh -ou
  682 root       0:00 macgyverd -f
  688 root       2:50 //usr/bin/dragon-prog
  697 root       0:00 init
  698 root       0:00 init
/ #
```

To stop the *dragon-prog* process, type:

```
kill -9 [NUMBER]
```

In our example:

```
kill -9 688
```

3.2. Restarting dragon-prog

You can now restart dragon-prog and activate the desired option(s). To do so, type the following and press ENTER:

```
/usr/bin/dragon-prog [OPTIONS] &
```

Where `[OPTIONS]` should be replaced with a string of characters that activates the desired options, as discussed in paragraph 3.3.
For example:

```
/usr/bin/dragon-prog -S 0 &
```

Disables image stabilization.

```
/usr/bin/dragon-prog -S 0 -s 1024x768 &
```

Disables image stabilization and switches the resolution of streamed video to 1024 x 768 pixels

You can use a single option or several of them in a raw. Just remember respecting the UNIX syntax (e.g. putting blank spaces before and after each option and argument). The final `&` sign ensures that *dragon-prog* keeps running after you close the telnet session.

After hitting ENTER, the display will fill up with output lines (reporting on different phases of the process startup) and eventually stop.
The Bebop is now ready for FreeFlight to connect.

3.3. Dragon-prog options

Not all commands and options have been fully understood by the community yet. The following paragraphs present our understanding of dragon-prog hacking and can be used as a reference to proceed with your own experiments. Please share your results with the community.

3.3.1. General options

`-h`
Displays a list of all available options with a short comment to each.

`--version`
Displays information on the software version.

3.3.2. Camera options

`-c`
This option sets the front camera resolution. The resolution is set by two numbers, separated by an x mark. Remember that the camera is turned on one side, so the sensor longer side is vertical and the shorter side is horizontal. Consequently, by setting:

```
/usr/bin/dragon-prog -c 1280x2048
```

1280 and 2040 represent the vertical and horizontal resolution, respectively. The raw image acquired by the camera based on the `-c` option argument is then reprojected (see `-R`) and streamed according to the `-s` option. As a consequence, the final image aspect depends on the combination of `-c` and `-s` options, making the results difficult to predict. Trial and error is the way to go if you want to adjust the camera settings to suit your needs. Par. 4.4.5. Describes a few scripts that exploit these options to obtain full-frame fisheye video recording and FPV-friendly video stream. The maximum tested values for `-c` are 3744x3744.

`-s`
This option sets the resolution of the live video stream. The resolution is set by two numbers, separated by an x mark.
Example: `/usr/bin/dragon-prog -s 1280x720`

Similarly to `-c`, changing the values in the `-s` argument may result in oddly shaped images, no image at all, or diagonal stripes. In such cases just restart the Bebop and try again with different numbers. The camera (and the h264 video compression) do not like just any number: standard resolutions for digital videos are a good start (e.g. 1280, 1024, 640).
This parameter has revealed particularly interesting for adjusting video settings for FPV piloting. By setting a lower resolution, the video stream may be more fluid. Furthermore, an appropriate horizontal/vertical ratio should be set to best fit with the field of view resulting from changing the `-V` option. Try with:

```
/usr/bin/dragon-prog -s 1024x520 -V 115
```

`-r`
This option sets the resolution of the recorded video. This option works in concert with -c and sets the final resolution of the recorded video. Also in this case any number can not be entered, and x and y dimensions should fit with digital video standards.

`-a`
The short explanation given for this option in the `-h` screen is rather obscure:
`ratio (<1) between camera output dimensions and acquisition window dimensions`
This sounds intriguing, but testing values of 1, 0.9 and 0.8 gave identical results in the live video stream. Smaller values appeared to crash dragon-prog.

`-b`
In ARDrone3 v. 2.x, this option sets the bitrate for live streaming (useful for adjusting streaming fluidity), but firmware upgrades have introduced an automatic adjustment depending on the wifi signal quality, so this option appears of little use right now. The argument is expressed in bits per second, so 30Kbps video streaming is set by:

```
/usr/bin/dragon-prog -b 30000
```

Since ARDrone3 v. 3.x, -b sets video recording bitrate, expressed in Kbps, so 30Mbps video recording is set by:

```
/usr/bin/dragon-prog -b 30000
```

`-q`
In ARDrone3 v. 3.x, sets the video streaming bitrate in Kbps.
Examples:

`-q 100` (0.1 Mbit) for ultra long range flights using low wifi bitrate
`-q 5000` (5 Mbit) for best streaming quality and short distance drone racing

`-f`
This sets the video frame rate. By default, Bebop videos run at 30 (or rather 29) frames per second.

`-R`
This sets the reprojection process on and off (see section 1). Reprojection can be disabled by `-R off` (resulting in a 90° turned image) or activated as by default via GPU (`-R gpu`) or CPU (`-R cpu`). CPU option anyway returned a black screen in our tests. A successful attempt at using this option is presented in paragraph 4.4.5.

`-I`
This changes the settings of an image manipulation filter that the Bebop applies before streaming or recording. During reprojection a noise reduction filter is applied by default to smoothen down camera noise. Beside turning the filter off with `-I off` , it is possible to visualize the effect of image filtering by entering `-I half` . This generates a video where half of the image is filtered and half unfiltered: a tool for developers, a curiosity for users.
The combined use of `-I off` and an edited version of dragon-prog that increases the recorded video bitrate (see par. 4.5) has been proposed to improve overall video quality, but the results in terms of image quality have not been published so far.
Since ARDrone3 3.0, two more arguments are possible: `-I luma` and `-I chroma`, respectively enabling noise reduction on image luminosity or color.

`-S`
Switches image stabilization (see section 1) on or off. Turning stabilization off with `-S 0` is considered essential for FPV piloting, where a direct feeling for the aircraft orientation in space is required. As a curiosity, this option can be used to demonstrate the effectiveness of the image stabilization algorithm.

`-V`
This option changes the field of view captured by the camera. The Bebop fisheye lens is capable of almost 180° field of view (FOV). The camera will anyway capture only a portion of this, correct the image for the wide-angle distortion, stabilize it and stream it to the SkyController or device. Adjusting the -V option allows the user to have a narrower or wider FOV, similar to a tele- or wide-angle objective. A narrower FOV can be useful to get zoomed-in images of distant objects; a wider FOV can be precious for FPV setup, because a wider view allows the pilot so see more objects and obstacles surrounding the lens and provides an overall more immersive experience.

Example: `/usr/bin/dragon-prog -V 111`

Sets the camera FOV to 111°. Smaller values reduce the FOV: the image looks more natural (reducing the wide angle distortion) but the screen frames a smaller portion of the lens view. Higher numbers squeeze the image sides in (increasing the wide angle projection). Highest numbers also produce black semi-circles in the left and right sides of the screen (see par. 1.1). Such black areas are more or less evident depending on the size ratio of the device you are using for live video streaming. For example, if you use an iPad, -V 115 generates a wide FOV image that will fill the screen without any black area on the sides. Nevertheless, the black areas will appear in the recorded video.

`-F`
This option records the live streamed video to the Bebop memory, in the location defined by the path argument.
The Bebop front camera generates both a low resolution video (only used for streaming to your device), and a high resolution video (recorded in the Bebop memory). If for some reason you want to save the low resolution streaming, this option may come in handy.

Example: `/usr/bin/dragon-prog -F /data/ftp/internal_000/`

Should save the live-streamed video in the `internal_000` directory (see par. 1.3). Anyway, we have no information of successful attempts at this so far.

`-d`
This option is stated to record a raw video to the memory location indicated in the argument. This should be a cool option for video professionals who prefer post-processing to the limited capabilities of the onboard Bebop hardware. We have not reports of successful tests yet.

`-e`
Sets a maximum limit to the exposure time (in milliseconds). This option puts a limit to the automatic image adjustments for video and photo acquisition. It can be useful to avoid oversaturated highlights (sky, clouds, artificial lights...) in high contrast conditions.
Example : `-e 20`

`-g`
Sets a maximum limit to the camera sensitivity, or gain (in gain units). This option puts a limit to the automatic image adjustments for video and photo acquisition. It can be useful to avoid oversaturated highlights under high contrast conditions.
Example: `/usr/bin/dragon-prog -g 2`

`-m`
Switches camera mode from automatic to manual or semi-automatic. Such features should be familiar to expert photographs. Briefly, the Bebop will normally use automatic image adjustments to obtain the best image exposure. Under particular conditions (e.g high contrast due to a bright sky with setting sun and dark shadows in the trees below), the resulting image may not suit the user taste. This is when semi-automatic or manual modes come in handy. With `-m 1` the camera operates in fully manual mode: exposure time and sensitivity have to be set with the corresponding options (see `-x, -y, -z`). With `-m 2` the user can only set the camera

sensitivity, while exposure time selection will be automatic. With `-m 3` the user sets shutter speed, sensitivity is automatic.

Example: `/usr/bin/dragon-prog -m 1 -x 20 -y 2`

> Will set the camera to manual mode and images are acquired with an exposure time of 20 ms and gain 2.
>
> `/usr/bin/dragon-prog -m 2 -y 2`
>
> Sets the camera to manual sensitivity mode (ISO priority): images are acquired with gain 2 and automatic exposure. Manual mode could also be interesting for shooting photos at night, with long exposures.

`-x`
Sets the exposure time (shutter speed) - in milliseconds - when manual mode is on (see `-m`).

`-y`
Sets the digital gain when manual mode is on (see `-m`).

`-z`
Sets the analog gain when manual mode is on (see `-m`). It is recommended to stay within the 1-3 range: higher values introduce strong noise.

`-u`
Sets the number of regions of interest (ROI) used for autofocus. This option has not been tested.

`-E`
Shifts the exposure time up or down by the number of EV (exposure value) stops indicated in the argument. *EV stops* is professional photography jargon, indicating units that increase or reduce the camera sensitivity by a power of 2. For the rest of us, this simply means that positive values generate brighter images, negative values generate darker images.
It is not clear yet if this option applies to automatic mode or all modes (see `-m`).

`-A`
Sets the white balance mode to automatic, manual or different presets (see `-P`). It remains unclear where the presets are stored in the internal memory.

`-P`
To be used in combination with `-A` . This option should allow the selection of white balance presets, but their definition or file location in the Bebop memory remain unclear.

`-G`
Unclear option to separately set the white balance for red and blue channels.

`-C`
According to the *dragon-prog* help message, this option should overlay a color bar to the camera images. This does not seem to work in our hands, and might be obsolete or require debug mode activation (see par. 1.3).

`--jpeg_quality`
Sets the compression quality for JPG photos in percent units (maximum quality is obtained with `--jpeg_quality 100`)

`--jpeg_path`
Sets the location where JPG photos are saved. Not tested yet.

3.3.3. Other options

`-M`
This is a potentially harmful option to change motor rotation. This option has not been tested so far. WE RECOMMEND NOT TO TEST IT

`-i`
Sets the IP address for the Bebop. This option is potentially dangerous, as changing the IP might hamper connections between your control device and the Bebop.
WE RECOMMEND NOT TO TEST IT

`--bcm_country_rev`
This option provides information on country-specific software versions. Not very exciting in our hands.

`-k`
This overlays several technical info to the recorded video. Such on-screen data only appear in the streamed and recorded video, not in photos. The displayed info include:

- *Time since boot-up*, in seconds (including decimals down to millionths of a second)
- *Camera exposure time* and *gain*
- *White balance*
- *Magnetometer-measured orientation: phi* (roll), *theta* (pitch) and *psi* (yaw) angles
- *Image cropping position* (likely with reference to the full 180° frame)

And a couple of more obscure parameters

`-v`
This option sets the verbosity (amount of information) displayed in the telnet terminal window during *dragon-prog* startup. The argument can range between 0 (no information) and 3 (full feedback).

Example:

```
/usr/bin/dragon-prog -v 2
```

`-t`
Running dragon-prog with this option active starts a preset test routine, but the routine must be set in the option argument and no info is currently available about this.

`-l`
This option sets which configuration file will be used by *dragon-prog*. By default this file is *dragon.conf* in the `/data` folder (see par. 1.3 and 4.3.8).

`-B`
Activates blackbox recording (see par. 4.3.6). All flight data are saved to a file located in `/data/ftp/internal_000/blackbox/`

`-U`
Activates usblackbox recording. At present it is not clear what is different compared to `-B`.

`-T`
When activated, this option generates a more verbose output during *dragon-prog* startup, apparently monitoring the CPU, if we trust in the -h information: `cpu charge analyse`

`-D`
Saves images recorded by the vertical camera to the Bebop internal memory. The argument `<fWait>:<fDump>` defines how many frames are skipped (`<fWait>`) and how many are saved (`<fDump>`). In our hands this option did not work, but it might be active only in debug mode (see par. 1.3).

`-H`
Obscure option related to multiple sensor configuration:
`0 disable sensor multiple configuration mode`
Whatever the option activates or deactivates, we do not recommend to fiddle with tsensors that are responsible for flight control.

`-n`
Is reported to ‘dump the *frameInfo* into eMMC memory’ (ARDrone3 v. 3.x only)

`-o`
Is reported to ‘serialize the *frameInfo* in the video streaming’ (ARDrone3 v. 3.x only)

`-O`
Is reported to ‘serialize the *frameInfo* in the video recording’ (ARDrone3 v. 3.x only)
What exactly is *FrameInfo* remains to be understood

3.4. Back to piloting

Once you have relaunched *dragon-prog* you can now exit the *telnet* session. Verify that your device is still connected to the Bebop wifi and launch FreeFlight.
If you have used a personal computer to run the *telnet* session, disconnect it from the Bebop wifi and connect your mobile device or SkyController as usual.
After the normal connection dialog windows, you should be ready to go.
If anything goes wrong at this point (no image, no connection, etc.) a problem must have occurred during *dragon-prog* restart (either a mistyped command or an invalid argument for the selected options). Switch the Bebop off and back on, and repeat the whole procedure correcting your command line.

Remember that when you manually restart *dragon-prog* with options, all your changes will be canceled at power-off. This is both a security (if you mess things up, you can easily restore the Bebop to its original settings) and a hassle, because if you often need a customized setup, you have to go through the *dragon-prog* restart procedure at each startup or battery replacement. Permanent changes to the Bebop behavior can be introduced by editing the configuration files (see section 4).

3.5. Additional onboard software

We present here a few things that can be done by launching other pieces of onboard software. The corresponding executables are located in */usr/bin* and can be launched from within a telnet session, by simply entering the executable file name.

3.5.1. BLDC_Test_Bench

This executable runs a series of tests on the Bebop brushless direct current (BLDC) motors. Before playing around with this, it is critical to remove all the propellers: during such tests the motors will do all sort of odd things (like oscillating between min and max speed, running backwards, etc) and will be completely out of your control, so you better have the bird's four feet on solid ground.

Slow motion

Two of the Bebop motors rotate clockwise and the remaining two counterclockwise. During stationary flight, this ensures that motor activity will virtually transmit no effective rotation to the drone. Unfortunately, high rotation speed makes appreciating propeller spin direction almost impossible to the naked eye. Nevertheless, there is a command for that. After setting up a *telnet* connection (see section 2), place the drone in a clear area for the propellers to rotate freely and enter the following command (all in one line):

```
BLDC_Test_Bench -S > /dev/null;sleep 15;BLDC_Test_Bench -H >
/dev/null
```

This will give you 15 seconds of motor rotation, at very slow RPM. No effective lift is generated and you can sit back and enjoy the show. Yes, this is an exception to the recommendation we

gave above: only for this test you can be safe with the propellers in place. In fact you will not appreciate motor rotation (albeit slow) without the propellers.
If the command doesn't work the first time, try again. `BLDC_Test_Bench` is reported to be a little buggy.

Calling Elvis
Brushless motors can be used to generate sounds. This is what the Bebop does at every startup: four short beeps confirming each motor is armed. In fact the Parrot engineerss have found room for some improvement. Try this:

```
BLDC_Test_Bench -M 3
```

The bebop motors will start playing the first notes of Elvis Presley's success 'Be-bop-a-lula'. Something totally useless but not to be missed by any Bebop ~~geek~~ owner.
For your curiosity, -M 1 plays the ordinary startup beep, -M 2 plays a more acute, short sound. And that is it. Parrot engineers appear to have run out of fantasy after 1-2-3.
There's one more thing you can do, anyway:

```
BLDC_Test_Bench -M -3
```

starts an endless loop, until you cancel with

```
BLDC_Test_Bench -M 0
```

Other BLDC options
`BLDC_Test_Bench` can do more than this, but most of the remaining options are aimed at testing motor funcionality and appropriate connection during product development or post-assembly.

WARNING: If you want to explore the BLDC universe, we recommend you first remove all four propellers to prevent shooting your drone around the room. Remember you are bypassing both remote and internal controls: generating rotor lift without control is guaranteed to crash your Bebop and can damage objects and/or hurt you or other people

---

In case you are curious, typing

```
BLDC_Test_Bench -h
```

In a *telnet* session will output a list of all available options. Among them, the `-f` option, is stated to activate the use of a Futaba RC with serial bus protocol and might deserve further attention.

3.5.2. Diagnostic
This executable runs a series of diagnostic tests and if launched with the `-v` option will output the results on screen. Open a telnet session, then enter:

```
diagnostic -v
```

The resulting output will give you diagnostic info on several onboard systems such as motor control (BLDC), barometer (MS5607), accelerometers (MPU6050), compass (AK8963), CPU

(P7MU), including min/max reference values. `diagnostic` can likely be launched during flight too, but down here no one dared trying yet.

3.5.3. dos2unix

This small command line utility is extremely useful for Windows users. It converts any text file (including scripts and configuration files) that has been edited on a Windows PC to a format that is compatible with the unix standard. In particular, the invisible characters that mark line breaks (carriage returns / line termination) use different encoding in dos-based vs unix-based operative systems, and this can cause an edited script to crash with unpredictable consequences. There is a desktop version of *dos2unix* too, but having it onboard means that you can upload your files right after editing, and then run *dos2unix* from a telnet connection.
To convert your file, open a *telnet* session, navigate to the directory where you have saved your file, then type:

```
dos2unix -u FILENAME
```

The same utility can be launched with the `-d` option instead of `-u` to convert a file from unix to dos format.

4. Irreversibly changing the Bebop behavior

This section explains how to introduce changes to existing files that will not be canceled at the next power-up, but are maintained until you change the files back to their original version. We also discuss the creation of additional scripts that can be loaded to the Bebop memory and then launched by the user to perform specific tasks.

WARNING: in case of serious problems that require drone replacement by Parrot the presence of altered files may invalidate your warranty

---

4.1. Backing up and copying files

Before you introduce any change it is essential to make a backup copy of all the files you are about to edit. This can be achieved in different ways.

4.1.1. Making local file copies

This procedure allows you to save a copy of your file of interest in the same directory as the file you want to edit. The method is particularly useful if you plan to edit the file via *telnet*.
Open a *telnet* session and activate read/write permissions if you are running ARDrone3 v 3.0 or above (see section 2), then go to the directory that contains the file you want to edit and type:

```
cp filename filename.old
```

where *filename* is the name of the file you want to backup. This will create a copy of the file and add the `.old` suffix to its filename. From now on, any modification to *filename* will affect the Bebop behavior, while the file tagged with `.old` will not be recognized by the Bebop software and just sit there as a local backup copy.
In case you want to restore the pristine condition, all you have to do is delete the edited file:

```
rm filename
```

and create a new copy of the original untouched filename.old file:

```
cp filename.old filename
```

We strongly suggest to create copies of all the files you are about to edit in an emergency directory, and first of all create (and keep updated) a failsafe script that will restore the original files in case things go terribly wrong. See par. 5.2 for more details.

4.1.2. Copying files to and from a computer

An alternative option is to transfer a copy of the files you want to edit to your personal computer hard drive. This has several advantages: you can open, edit and manage your files when you are not connected to the Bebop; you can make further copies or save different edits for future use.

To copy a file to your hard drive, anyway, you have to use an FTP connection (see section 2), and FTP connections only provide you an access to the `/data/ftp` directory in the Bebop filesystem.
Consequently, any file you want to copy to your hard disk must first be copied to the `/data/ftp` directory (or a sub-directory). As previously mentioned (see par. 1.3), it is recommended to use the `/data/ftp/internal_000` directory for file transfers and storage, because of its larger memory allocation.
To this aim, open a telnet session, navigate to the directory containing the file you want to copy and type:

```
cp filename /data/ftp/internal_000/filename
```

This will create a copy of your file in the `/data/ftp/internal_000` directory.
You can then open an FTP session and download your copied file to your hard drive:

```
get filename
```

Will transfer a copy of the file to your home directory.

4.2. Editing Bebop files

Once you have created a backup copy of the original files (see par. 4.1), editing can be done either online, directly in the Bebop memory - during a *telnet* session - or offline, by editing a copy of the file in your hard drive and then copying back the edited file to the appropriate directory within the Bebop filesystem.

Important notes on file editing
ARDrone3 is a unix-derived system. As a consequence, it is of UTMOST IMPORTANCE that all files edited on your personal computer are saved in unix-compatible encoding, in particular concerning the line termination characters (invisible characters that encode a carriage return). If this is not the case, all sort of odd behaviors can appear and make your hacking experience very frustrating. Some scripts will simply not launch, in other cases they will be run, but come to an early stop. So, always remember to save your files in a unix-compatible format, or convert them before uploading the edited file back on the Bebop. This should not be an issue for Linux and Mac users, but Windows users should take advantage of a conversion utility such as dos2unix (see par. 3.5.3).

If you choose to copy and paste the text from any of the scripts and files presented in this guide, beside the above comment on line termination characters, pay extra attention to wrapped text: some code lines can be longer than the page width, forcing the extra text to drop down in the following line. In all such cases, verify that the code line is kept together after pasting it into your text editor, or the file execution will end with an error.

Under ARDrone3, remember to enable read/write permissions to the whole file system (see par. 2.3)

4.2.1. Online file editing

To edit a configuration file during a *telnet* session, you should use the file editor called *vi*, which is included in the Bebop operative system.
Navigate to the directory containing your file and type:

```
vi filename
```

If you are not familiar with *vi* use and commands (which are not very intuitive), information can be found on the internet.

It should be possible to install a more friendly editor, such as *nano*. A suitable version of *nano* can be downloaded from this forum.

4.2.2. Offline file editing

Once you have a copy of the file you want to edit in your hard drive, it is recommended to make a further backup copy before editing.
All files can be opened and edited by any text editor. Introduce the desired changes and save the edited file.
The edited file must then be copied back to the Bebop filesystem, but first, please read the *Important notes on file editing* in the previous pargraph.
Place the edited file in your home directory. Open an FTP session, navigate to the target directory in the Bebop filesystem that contains the original file.
Type:

```
put filename
```

This copies the edited file from your home directory on the computer to the target directory in the Bebop filesystem.

After replacing the original file with the one you have edited on your personal computer you have to change the file permissions, or the Bebop software may not be able to use it. The same applies to any new file you create and upload, such as scripts that perform specific tasks when launched by the user.
In a *telnet* session, navigate to the directory containing your file and type:

```
chmod 755 filename
```

to give appropriate reading, writing and executing permissions to the edited file.
Verify that permissions are ok with:

```
ls -la
```

Hitting enter will list the contents of the current directory, including permission information. For example, `ls -la` of the `/data` directory gives:

```
/data # ls -la
total 68
drwxr-xr-x    6 root     root           800 Jan  1 00:00 .
drwxr-xr-x   21 root     root          1528 Jan  1 00:06 ..
-rw-r--r--    1 root     root           798 Jan  1 00:00 dragon.conf
-rwxr-xr-x    1 root     root          8192 Jan  1 00:00 eeprom.dat
-rw-r--r--    1 root     root         45655 Jan  1 00:00 eeprom.dat.txt
drwxr-xr-x    2 1000     1000           160 Jan  1 00:00 exceptions
drwxr-xr-x    3 root     root           376 Jan  1 00:00 ftp
drwxr-xr-x    3 root     root           232 Jan  1 00:00 lib
-rw-r--r--    1 root     root            47 Feb 14  2016 magneto_calibration.conf
-rw-r--r--    1 root     root           495 Jan  1 00:00 system.conf
drwxr-xr-x    2 root     root           240 Jan  1 00:00 updater
/data #
```

configuration files like *dragon.conf* should have `-rw-r--r--` permissions, while shell scripts like *Dragonstarter.sh* should display a `-rwxr-xr-x` tag (like `eeprom.dat` in the example above).
You can also add the 'executable' tag to any chosen file with:

```
chmod +x filename
```

WARNING: Always double check permissions if you edit the files offline: an error here can hamper script execution and block the Bebop boot-up process. Also note that all files must be saved as raw text files and use UNIX-compatible encoding for end-of-line marks etc., otherwise they may not be executed (see par. 3.5.3)

4.3. Tests and results

This section presents a few examples of edits that may address common problems or add custom features. This list is obviously not exhaustive, and is meant to inspire further experiments and modifications. Please share your own results with the community.

4.3.1. Editing DragonStarter.sh

The *DragonStarter.sh* script, residing in the `/usr/bin` directory can be edited to make it launch *dragon-prog* with the desired options active. To this aim, the line

```
POST_CMD=""
```

must be edited by adding the desired string of options commands (see par. 3) between the quotation marks.

For example:

```
POST_CMD="-S 0 -k -I off"
```

will cause dragon-prog to always run with image stabilization off, on screen data display, noise reduction filter off.

To go back to the original settings, just replace the edited line with

```
POST_CMD=""
```

4.3.2. Launching dragon-prog from a shell script

There are alternative ways to change the *dragon-prog* launch procedure. For example open a *telnet* session and go to the directory:

```
/etc/init.d/
```

Then create a file called *startup_script.sh* containing the following lines:

```
#!/bin/sh
kill -9 `ps | grep dragon | grep '/' | awk '{print $1}'`
/usr/bin/dragon-prog -S 0 -k -I off &
```

When launched, your *startup_script.sh* will kill *dragon-prog* and restart it with the desired options, in analogy to the manual procedure described in section 3.

If you want the script to be run every time the Bebop is powered up, you can now edit the following file:

```
/etc/init.d/rcS
```

After the line

```
sleep 1
```

add the following line:

```
/etc/init.d/startup_script.sh
```

The result is the same as editing *DragonStarter.sh*, but you can save several different scripts in the init.d directory, each with different settings and then choose which one to apply by editing the *rcS* file.

4.3.3. Protecting the wifi network

ARDrone3 v. 3.x has introduced WPA password protection to the Bebop wifi network. Before that, the default Bebop wifi network was not password-protected and alternative methods to improve wifi security were developed, such as hiding the network SSID and/or restricting wifi access to a preset list of devices, identified by their MAC address. Both strategies can still be valid, especially because wifi password protection via WPA is expected to reduce the signal range and quality. Hidden SSID and MAC address filtering can both be activated transiently or irreversibly.
In order to reversibly hide the wifi SSID, open a telnet session and enter the following command:

```
bcmwl closed 1
```

This makes it more difficult for others to connect to your drone, since the wifi network will not appear in the list of available networks. This does not affect the ability of your own control device to connect to the Bebop. Nevertheless, concerns have been raised by users about possible issues when the drone is flying close to the wifi range limits, or upon temporary disconnections. To activate MAC address filtering, you should enter the following commands:

```
bcmwl mac MA:CA:DD:RE:SS:01 MA:CA:DD:RE:SS:02
```

where you enter all the MAC addresses you want to authorize (including the SkyController, if you own one), followed by

```
bcmwl macmode 2
```

which sets MAC filtering to only accept connections from the previous list of devices. MAC filtering and hidden SSID can both be activated by entering all three commands.
The same command lines can be written inside a script and run by launching the script.

WARNING for SkyController users: the SkyController generates two distinct networks, one for the Bebop and one for your device. It therefore has two MAC addresses, and the one you should enter in the bcmwl list can only be found by using network analysis software such as *Wireshark*.

In order to permanently change the wifi settings you should edit the following file:

```
/sbin/broadcom_setup.sh
```

The command lines described above should be entered after the line that reads:

```
bcmwl sgi_tx 0
```

WARNING: Be extremely careful, because this introduces permanent changes in a file that will NOT be reset to its original version (*broadcom_setup.sh*) by the hard reset procedure (see section 5). Furthermore, do not forget to introduce as many MAC addresses as possible, to cover all the devices that you own and can connect to the Bebop (e.g. in case you lose your smartphone).

4.3.4. Improving GPS accuracy

The Bebop GPS chip can recognize and connect to a number of different GPS satellite networks. The settings that select which networks it should use are stored, with other parameters, in the file

```
/etc/gps_config.txt
```

at the line that reads:

```
PERDAPI,GNSS,GN,2,2,0,-1,-1
```

This is what the parameters that follow GNSS indicate:

TALKER_ID
GPS mode
GLONASS mode
GALILEO mode
QSZZ mode
SBAS mode

The *talker_id* is how the system identifies itself, and it is set to GN. We do not have to bother about this.
The four following digits configure how the chip interacts with each of the four global positioning satellite networks called GPS, GLONASS, GALILEO, QZSS and SBASS. Such geostationary satellite networks have been put in orbit by different countries: GPS (USA) has global coverage; GLONASS (Russia) has global coverage too; GALILEO (Europe) is planned to be partially operational in 2016 with 8 satellites and achieve global coverage in the following years; QZSS (Japan) is planned to have four operational satellites in 2018; SBAS is a system that coordinates the networks of several countries, including the US-based WAAS, Indian GAGAN, and others.
By editing the *PERDAPI* line you can set the GPS chip to access to each of these networks, depending on what digits you enter in the corresponding position.

This is what the digits mean:

`-1` keep current configuration
`0` disable
`1` enable tracking but do not use for position fix
`2` enable tracking and use for position fix
`3` enable tracking and use for position fix only after first fix

By factory settings, then, the Bebop drone uses GPS for tracking and position fix (2); does not use GALILEO (0), which makes sense, since the network is not yet operational; and keeps the 'current configuration' (-1) for QZSS and SBAS. These last two settings likely act on upstream choices linked to the location that you select in FreeFlight.
In other words, the GPS is always used, while Japanese and international networks are activated when you tell the piloting application that you are located in a specific country (e.g. Japan or India) where it can be useful to exploit local satellite networks.
By editing this line you can however force the chip to also use these last two satellite networks (and GALILEO, if your drone survives until the network gets operational).
All you have to do is change the corresponding digits.

For example:

```
PERDAPI,GNSS,GN,2,2,0,-1,3
```

can be useful to activate SBAS connectivity, because you simply make a larger number of satellites potentially available to the drone for position fix and tracking. In short, this can speed up the GPS fix (green GPS icon) and also provide higher precision when using return-to-home or flight plan functions.
Note that all this has only been tested on ARDrone3 v 2.0.57.

4.3.5. Editing a dragon-prog configuration file

Part of the settings used to configure dragon-prog are recorded in this configuration file:

```
/data/dragon.conf
```

As mentioned in section 1, most `dragon.conf` settings can be adjusted from the FreeFlight options screen. Nevertheless a few hidden options are present and one in particular is worth considering.
By changing the `picture format` value to 0 (zero), in fact, every time you press the photo button, a 180° image will be shot and saved in two formats: JPG plus DNG raw format. This uncompressed image file format dramatically improves picture quality (and proportionally increases file size). A semi-professional image editor is required for opening and rendering a DNG image, but the results are far better than the JPG standard.
To achieve this, you must edit the original `dragon.conf` file, located in the `/data` directory, by changing the picture format line to:

```
"picture_format" : 0,
```

Then turn off the Bebop. At next power-up, the edited *.conf* file will be used.
Video recording will not be affected and you will simply have to stop it before you can shoot 180° DNGs + JPG pictures. Remember anyway that as soon as you change the video/photo settings in FreeFlight options, the picture format value will be changed back and your modification will be cancelled. Just do not touch the recording options in FreeFlight or, if you do so, go through file editing again.

You can also create a script that restores your *dragon.conf* editing at every startup. In this way, if you change the recording options in FreeFlight (either because you want to shoot a normal picture or just by mistake), reboot will set the picture format option back to zero.
To this aim you should create a copy of your edited `dragon.conf` file in the `/data` directory with:

```
cp dragon.conf dragonBKP.conf
```

Then create a script like the following:

```
#!/bin/sh
echo "Restoring picture format to 0"
# Copy dragonBKP.conf file over dragon.conf
cp /data/dragonBKP.conf /data/dragon.conf
```

and save it as `reset_conf_script.sh` in `/etc/init.d/`

Now edit the following file:

```
/etc/init.d/rcS
```

After the line

```
sleep 1
```

add the following line:

```
/etc/init.d/reset_conf_script.sh
```

Upon each power-up, *rcS* will run the script for you and replace the *dragon.conf* file with your edited version.
The same approach can be adapted to run any script at startup, but should be used with extreme care, since a flawed script might crash the software at every startup and hamper your attempts at recovering the original conditions.

4.3.6. Turning on the black box

While the bebop software normally records low resolution flight data in *pud* files (see par. 2.5), higher resolution flight data can be obtained by editing the configuration file *debug.conf* to activate the blackbox function. The file is located in the `/etc` directory.
Near the bottom of the file, change

```
BLACKBOX = 0
```

to

```
BLACKBOX = 1
```

Now, at each takeoff, a file named `light_run_*` will be written with highly detailed flight information in the `/data/ftp/internal_000/blackbox` directory. The file can directly be retrieved via an FTP connection (see section 2).
Each *light_run* file starts with an ASCII header, which contains two sections. The first section reports the firmware version and some additional information; the second lists the headers corresponding to the data columns that follow in the recording. There are over 120 columns including raw sensor data, position, wind estimates, motor commands and more. A data header precedes the block of recorded data. Pyton scripts are available online to convert blackbox files to comma separated value files (csv) that can be opened and edited in any spreadsheet application.
Blackbox recording can also be transiently activated by launching *dragon-prog* with the `-B` option active (see par. 3.3).

4.4. Power to the power button

The only physical interface between the user and the Bebop is the power button. Not surprisingly, Parrot has introduced the possibility to activate different tasks by pressing the power button, including a few that are crucial under critical circumstances (see section 5). Beside turning the drone on and off, in fact, the power button can be used to start different scripts depending on *how* the button is pressed. As described in par. 1.3, the `/bin/onoffbutton` directory contains scripts that are launched this way.

`longpress_0.sh` is launched when the button is pressed for 3 seconds. The script switches the wifi band from 2.4 to 5 GHz and back, as reported in the Parrot user guide.

`shortpress_1.sh` shuts down the Bebop. Do not edit this script.

`shortpress_4.sh` activates telnet and usb networking. Do not edit this script.

`shortpress_13.sh` activates USB networking (see par. 5.4). Do not edit this script.

`verylongpress_0.sh` launches a Bebop hard reset to factory defaults when the power button is pressed for 10 seconds. This is a very useful way to bring a bricked Bebop back to life. Do not edit this script.

If you do not find the wifi band switch function useful, you could edit `longpress_0.sh` to have it launch the *startup_script* described in par. 4.3.2. This allows you to have a standard Bebop startup at powerup, and switch to your special configuration with a 3 sec pressure of the power button.
This is obtained by editing the file:

`/bin/onoffbutton/longpress_0.sh`

as follows:

```
#!/bin/sh
echo "Restarting dragon-prog via startup_script" | logger -s -t "LongPress" -p user.info
# Orange LED for 2 seconds
(BLDC_Test_Bench -G 1 1 0 >/dev/null; sleep 3; BLDC_Test_Bench -G 0 1 0 >/dev/null) &
./etc/init.d/startup_script &
```

A more direct - albeit possibly less versatile - way to trigger *dragon-prog* relaunch upon long power button pressure can be:

```
#!/bin/sh
echo "Restarting Dragon for FPV" | logger -s -t "LongPress" -p user.info
# Red LED for 3 seconds
(BLDC_Test_Bench -G 1 0 0 >/dev/null; sleep 3; BLDC_Test_Bench -G 0 1 0 >/dev/null) &
kill -9 `ps | grep dragon | grep '/' | awk '{print $1}'`
/usr/bin/dragon-prog -S 0 -s 1024x520 -V 115 -I off &
```

Always remember to set the correct permissions to the files you edit or create, with `chmod 755 filename` (see par. 4.2.2).

Furthermore, it is possible to create extra scripts and start them with a combination of button presses, as described below.

4.4.1. Additional power button scripts

In addition to editing the *longpress_0.sh* script, more scripts can be created that are started by different numbers of repeated short pressures of the power button. A native example is the *shortpress_13.sh* script, but it is not a good idea to edit this script. Instead, create the script that does the job you need and save it with the following file name:

```
shortpress_X.sh
```

Where `X` is replaced by the number of button presses you want to use to launch the script.
In fact you can add as many scripts as you want in the `/bin/onoffbutton` directory and run them on demand by 'dialing' the corresponding number of button presses. For example you could make a failsafe script like the one described in par. 5.2 and name it

```
Shortpress_9.sh
```

to have it launched by 9 short pressures of the power button, instead of editing the 3 second long press script.

4.4.2. Forcing GPS cold start

If you travel a long distance between flight sessions, this may cause the Bebop to spend minutes in search of the GPS satellites before the GPS icon turns green. The reason is that by default the Bebop is not performing a so-called *GPS cold start* when booting and will therefore search for the satellites it used last time.
To force a GPS cold start when the power button is pressed 7 times you should first telnet to your Bebop (and activate read/write permissions) and enable input to the GPS chip configuration file by typing:

```
chmod 0+rw /dev/ttyPA1
```

Note that this operation only needs to be done once.

You can then upload the following script to `/bin/onoffbutton/`

`/bin/onoffbutton/shortpress_7.sh`

```
#!/bin/sh
echo "Cold Resetting GPS" | logger -s -t "shortpress_7" -p user.info
# Orange LED for 3 seconds
(BLDC_Test_Bench -G 1 1 0 >/dev/null; sleep 3; BLDC_Test_Bench -G 0 1 0 >/dev/null) &
echo -e "$PERDAPI,STOP*6F" > /dev/ttyPA1
echo -e "$PERDAPI,START,COLD*1F" > /dev/ttyPA1
```

Since next reboot, a 7 presses of the power button will force the GPS cold start and quickly find your GPS fix.

4.4.3. On-the-go USB file transfer

NOTE: The following procedure has only been developed for ARDrone3 v. 2.0.57.
Version 3.x and above manage USB thumb drives in a different way, automatically copying the whole `/data/ftp/internal_000/Bebop_Drone` directory to the USB memory stick. More tests are needed before we can propose a reliable script-operated method for the new operative system.

If you think the internal Bebop memory is too small to host all the videos and pictures you record, and you do not have your laptop within reach, it can be very useful to plug a large USB stick into the Bebop and have the drone move all your files to the device.
You will need a passive USB OTG cable (easy to find on eBay), with a male micro USB plug at one end and a female USB socket at the other. When such a cable is used to plug a USB stick, the stick memory is mounted in the Bebop filesystem, inside the `/data/ftp` directory, and a few directories are created on the stick:

```
Bebop_Drone/academy
Bebop_Drone/media
Bebop_Drone/thumb
```

You can now open a telnet session from your device and manually copy the content of the Bebop `/data/ftp/internal_000/Bebop_Drone/media` directory to the corresponding USB stick directory.

You can anyway prepare a couple of scripts that do the job for you and have them run by specific power button press codes, as described for other scopes in previous paragraphs.
In this case we suggest to use one script for copying the files to the USB stick and a second script to delete the originals from the Bebop memory. This allows you to unplug the stick after the first script has finished, plug it into your mobile device to check that all files are OK, and only then run the second script that frees the Bebop memory.
Please note that this procedure has been successfully tested with Kingston 64mb and 16mb USB sticks.
Here is a tested example of what the scripts may look like.

Script 1 (copy): `/bin/onoffbutton/shortpress 5.sh`

```
#! /bin/sh
# This script copies your media files to an external USB stick
# Red LED = script starts
(BLDC_Test_Bench -G 1 0 0 >/dev/null) &

INT_MEM_DIRECTORY=/data/ftp/internal_000/
BOP_MEDIA_DIRECTORY=/data/ftp/internal_000/Bebop_Drone/media/
USB_OTG_DIRECTORY=

# search for USB OTG directory using pattern *_nnn
for directory in $(find /data/ftp/ -maxdepth 1 -type d -regex ".*_[0-9][0-9]*"); do
       if [ "$directory/" != "$INT_MEM_DIRECTORY"Bebop_Drone/media/ ]; then
       USB_OTG_DIRECTORY="$directory"/Bebop_Drone/media/
       break
       fi
```

```
done
# copy the files if we have USB OTG drive mounted
if [ -d "$USB_OTG_DIRECTORY" ]; then
       echo "Moving media to USB OTG drive ($USB_OTG_DIRECTORY)..."
       # Orange LED during copy
(BLDC_Test_Bench -G 1 1 0 >/dev/null) &

       cp -f "$BOP_MEDIA_DIRECTORY"* "$USB_OTG_DIRECTORY"
       # Green LED = copy finished
       (BLDC_Test_Bench -G 0 1 0 >/dev/null) &
       #creates copy_ok file when the copy process is finished
       touch "$INT_MEM_DIRECTORY"Bebop_Drone/copy_ok
else
       echo "USB OTG drive not mounted!"
       # Red LED flashes for 3 seconds to notify the error
(BLDC_Test_Bench -G 1 0 1 >/dev/null; sleep 3; BLDC_Test_Bench -G 0 1 0 >/dev/null)
&
fi
```

Since the name of the USB stick directory in `/data/ftp` may change, the first part of the script searches for the corresponding file name pattern (an underscore followed by three numbers). File copy starts only if such a directory is found. The power button LED will give the following feedback on the script execution: red light on when script starts; orange light on during copy; green light on when copy is completed successfully; alternatively, red light flashing if copy was not successful (the USB stick was not mounted).

Script 2 (delete): `/bin/onoffbutton/shortpress_6.sh`

```
#! /bin/sh
# This script deletes your media files if they have previously been copied to a USB stick
# Red LED = script starts
(BLDC_Test_Bench -G 1 0 0 >/dev/null) &

INT_BOP_DIR=/data/ftp/internal_000/Bebop_Drone/

if [ -e "$INT_BOP_DIR"copy_ok ]; then
   echo "Previous copy was successful"
   # delete the files
   echo "Deleting contents of media directory"
   # Orange LED flashing during deletion
   (BLDC_Test_Bench -G 1 1 1 >/dev/null) &
   # delete all media files
   rm -R "$INT_BOP_DIR"thumb/*
   # delete all thumb files
   rm -R "$INT_BOP_DIR"media/*
   # delete file copy_ok
   rm -f "$INT_BOP_DIR"copy_ok
   # Green LED flashes= delete OK
   (BLDC_Test_Bench -G 0 1 1 >/dev/null; sleep 3; BLDC_Test_Bench -G 0 1 0 >/dev/null) &
else
   echo "Media were not previously copied! Nothing has been deleted"
   # Red LED flashes for 3 seconds
   (BLDC_Test_Bench -G 1 0 1 >/dev/null; sleep 3; BLDC_Test_Bench -G 0 1 0 >/dev/null) &

fi
```

Note that the delete script checks for the existence of a file called `copy_ok` in the directory `/data/ftp/internal_000/Bebop_Drone` before deleting videos and photos from the internal

memory. Since `copy_ok` is created by the copy script at the end of successful copy process and deleted by the delete script at the end of the delete process, this check should ensure that you do not accidentally erase your media directory.
During the delete script execution, the power button LED gives the following feedback: red light on when script starts; orange light flashing during media file deletion; green light flashing when all files have been deleted; alternatively, red light flashing if the `copy_ok` file was not found.

USB transfer is also the fastest way to get data out of your Bebop. As a reference, transferring 4.04gb of data from the Bebop memory takes:

Connection	time	speed
WiFi 5 GHz	8min 50sec	7.63 MBps
WiFi 2.4 GHz	10min 46sec	6.25 MBps
USB Transfer	5min 30sec	12.24 MBps

4.4.4. The lazy shortpress 4

As you already know, under ARDrone3 v. 3.x, the *telnet* daemon is launched by four presses of the power button. This is achieved through the script `/bin/onoffbutton/shortpress_4.sh`

WARNING: by editing `shortpress_4.sh` you are fiddling with the lock of your main door to the Bebop filesystem: if you introduce an error here, you might permanently lose the ability to connect via *telnet*!
Safety precautions are described at the end of this paragraph:
read them carefully and ponder your decision!!!

You should also know that before you can edit or move files, you must activate read/write permissions to the whole filesystem by entering the folowing command in a *telnet* session:

```
mount -o remount,rw /
```

Repeating this sequence at every connection appears rather annoying to some of us, and we propose here a mod that includes the *mount* command within the `shortpress_4.sh` script, as follows:

`/bin/onoffbutton/shortpress_4.sh`

```
#!/bin/sh

echo "Button activating..." | ulogger -t "ShortPressDebug" -p I
/bin/usbnetwork.sh
echo "Button activated" | ulogger -t "ShortPressDebug" -p I
echo "activating read-write permissions"
mount -o remount,rw /
echo "write permissions active"
```

The mod is highlighted in bold. Replacing the original script with this edited version, makes sure that by pressing four times the power button, both telnet and write permissions will be activated immediately.

Before you jump on the train of happy lazy hackers, anyway consider very well the following points:

- Read only permissions on the file system have been introduced by Parrot as a safety measure that prtevents unwanted damage (possibly after they have realized how common it used to be hacking into the original Bebop system). If you decide to force that at every connection, never forget you did or you migh get in big trouble.
- We have already highlighted the importance of checking exec permissions and line-termination characters in the scripts that you modify. Since we are now dealing with modifying the only script that opens your way into the file system, such aspects deserve your full attention.

Since, anyway, one can never be 100% safe, here is a strategy that will save your telnet access in case you make any mistake while editing `shortpress_4`:
Before doing anything else to the script, create a backup copy with the following command (you must activate write permissions first, of course):

```
cp /bin/onoffbutton/shortpress_4.sh /bin/onoffbutton/shortpress_12.sh
```

Followed by:

```
chmod 755 /bin/onoffbutton/shortpress_12.sh
```

It is not over yet: turn your Bebop off and on again, then verify that the telnet daemon starts by pressing the button 12 times. If everything is ok, you can go on and edit *shortpress_4.sh*.
After the edit, in case the new script does not run and the telnet connection is not open, you can press the button 12 times to launch the backup script and you are ready to go again.

4.4.5. Custom camera settings

The camera options of *dragon-prog* are powerful tools to change the aspect of the streamed as well as the recorded video. We present here a few scripts that can be used to relaunch dragon-prog with alternative camera settings, but a number of other combinations can be used to suit specific needs.

Wide-angle video

While the Bebop camera can shoot beautiful circular 180° pictures, videos are restricted to a more standard form factor. By default, in fact, the camera is set to frame a small portion of the 180° field of viev (FOV) of the lens, which is also involved in the software process of image stabilization (see par. 1.4).

It is anyway possible to open the camera field of view until it eventually embraces the whole lens viewing angle. This can be done by restarting *dragon-prog* (see section 3) with the appropriate options.
For example:

```
/usr/bin/dragon-prog -c 1280x2560 -s 1024x600 -V 115
```

Will set a 115° FOV and adjust camera settings (-c and -s) to cover the new viewing angle.

The same (and alternative) settings can be written in a shell script that can be run on demand, for example by using the power button press system (see par. 4.4). Let us explore a couple of possibilities:

`/bin/onoffbutton/shortpress_7.sh`

```
#! /bin/sh
# This script restarts dragon-prog with camera settings for full frame fisheye video
# Red LED for 2 seconds
(BLDC_Test_Bench -G 1 0 0 >/dev/null; sleep 3; BLDC_Test_Bench -G 0 1 0 >/dev/null)
&

kill -9 `ps | grep dragon | grep '/' | awk '{print $1}'`

/usr/bin/dragon-prog -c 1280x3072 -s 1024x600 -V 140 &
```

Once you have placed this script ìn the `/bin/onoffbutton/` directory and changed its permissions to make it executable, 7 presses of the power button will relaunch *dragon-prog* and generate stunning 140° videos.
You may want to consider adding an options to disable noise filtering (resulting in sharper images). To do so, replace the last line in the script with:

```
/usr/bin/dragon-prog -c 1280x3072 -s 1024x600 -V 140 -I off &
```

First Person View

Wide angle lenses (not as extreme as the one we have on our Bebop) are of common use in racing drones and analogous FPV setups. In fact, wider viewing angles allow the pilot to better appreciate which objects surround the drone and how far they are, reducing the occurrence of unavoidable crashes. If you want to go this way with your Bebop, the following script can be a starting point:

`/bin/onoffbutton/shortpress_7.sh`

```
#! /bin/sh
# This script restarts dragon-prog with FPV-friendly camera settings
# Red LED for 2 seconds
(BLDC_Test_Bench -G 1 0 0 >/dev/null; sleep 3; BLDC_Test_Bench -G 0 1 0 >/dev/null)
&

kill -9 `ps | grep dragon | grep '/' | awk '{print $1}'`

/usr/bin/dragon-prog -c 1280x3072 -s 1024x600 -V 120 -S 0 -I off &
```

This sets a FOV of 120° (quite common for FPV) and disables camera stabilization, to make your piloting experience more realistic and immersive. It also disables noise filtering (`-I`) to reduce CPU load and help streaming fluidity.
Also consider the use of the `-q` option (par. 3.3.2) to further improve video streaming.
Remember to change permissions with `chmod 755 /bin/onoffbutton/shortpress_7.sh`

Disabling lens correction

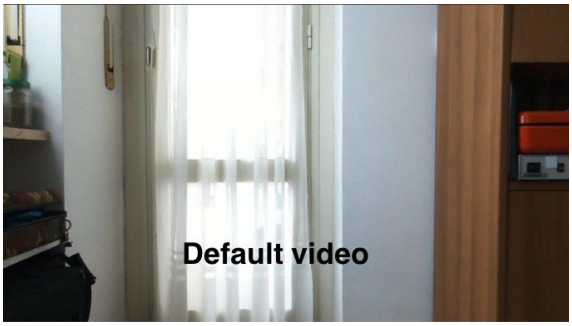
As mentioned above, the onboard software performs a *reprojection* of the images coming from the fisheye camera. This process corrects the image, removing the fish-eye deformation and also turns the image on its side by 90°, to correct for the camera orientation. By combining the use of different dragon-prog options, it is possible to record a video of the original uncorrected frame, even if this will inevitably appear turned by 90° and further video editing will be needed to reorient it correctly. The following script does the job:

`/bin/onoffbutton/shortpress_7.sh`

```
#! /bin/sh
# This script restarts dragon-prog fisheye camera settings
# Red LED for 2 seconds
(BLDC_Test_Bench -G 1 0 0 >/dev/null; sleep 3; BLDC_Test_Bench -G 0 1 0 >/dev/null)
&

kill -9 `ps | grep dragon | grep '/' | awk '{print $1}'`

/usr/bin/dragon-prog -s 1280x1400 -c 3744x3744 -R off -r 1280x1400 -I off &
```



As you can appreciate from the images above, the non-reprojected image does not have a wider angle than the original image, but horizontal and vertical lines appear bent as for the lens properties. In theory, by increasing the -r resolution, it should be possible to embrace a wider field of view, but in our hands the above setting was the widest that did not cause a dragon-prog crash due to data overflow. It is anyway possible to experiment with different image formats, such as 1024x2048, which generates a wider (but shorter) image, where the horizontal field of view is indeed broader than with the default settings. As a side comment, disabling reprojection

with -R also makes -V ineffective, so it is not possible to widen the field of view with this approach.

4.5. Scripts for saving and retrieving GPS data

On the Bebop(s), data from the GPS hardware are accessible via a Linux FIFO. You can think of the FIFO as a live access point located at `/tmp/gps_nmea_out`. Data is not recorded from here by default, but a script can be written to capture the NMEA GPS data into a file in a directory that can later be accessed via FTP. How we do this depends on which Bebop we are working with.

4.5.1. ARDrone3 v 2.0.57

The Original Bebop uses a *Furuno GN-8715* GPS receiver. This receiver is configured to send standard NMEA sentences plus several proprietary sentences, all of which are ASCII text. The Furuno terminates all sentences with a CR/LF, so viewing the data will display correctly in a DOS environment and other machines will require conversion with the *dos2unix utility* or equivalent (see par. 4.2.2).

The following method has been tested on Bebop Drone running ARDrone3 v 2.0.29 and 2.0.57. It involves a one time change to a Bebop system startup file and an external trigger file that allows convenient switching of GPS data capture on/off through a special power button sequence.

Here's an outline of the data capture process.

1) With the Bebop powered on, pressing the power button eight times will launch a script that creates the trigger file.
2) To start GPS data capture you now need to reboot your Bebop and during next startup, your edited *rcS_gps* script will find the trigger file and activate data capture.
3) At the end of your flight, press the power button eight times: the *shortpress_8.sh* script will delete the trigger file disabling GPS data capture as of the next power-up cycle.
4) You can now use FTP to retrieve the `furuno-nmea-out.txt` file from `/data/ftp/internal_000/log/`

To make the process easier to follow, we will deal with the power button script first.
The power button script creates or deletes the trigger file on demand. This is achieved by creating the following script in `/bin/onoffbutton/`

`/bin/onoffbutton/shortpress 8.sh`

```
#! /bin/sh
# This script creates or deletes an empty file called furuno_data in /data/ftp
directory
# Flashing red LED = script starts
(BLDC_Test_Bench -G 1 0 1 >/dev/null; sleep 3) &

INT_BOP_DIR=/data/ftp/
if [ -e "$INT_BOP_DIR"furuno_data ]; then
  echo "Trigger file found – toggle capture off"
  # delete the file
  echo "Deleting trigger file"
  rm -R "$INT_BOP_DIR"furuno_data
  Echo "Trigger file deleted"
```

```
  # Orange LED flashes= delete OK
  (BLDC_Test_Bench -G 1 1 1 >/dev/null; sleep 3; BLDC_Test_Bench -G 0 1 0
>/dev/null) &
else
  echo "Trigger file not found"
  # create trigger file – toggle capture on
  echo "Creating trigger file"
  touch "$INT_MEM_DIRECTORY"furuno_data
  Echo "Trigger file created"
  # Green LED flashes= create OK
  (BLDC_Test_Bench -G 0 1 1 >/dev/null; sleep 3; BLDC_Test_Bench -G 0 1 0
>/dev/null) &
fi
```

Remember to change permissions with:

```
chmod 755 /bin/onoffbutton/shortpress_8.sh
```

And ensure that line termination characters are Linux. If there is any doubt, run *dos2unix utility* on *shortpress_8.sh*.

Now for the second part, editing the `rcS_gps` file (which is run during startup) so that it will check for the trigger file created by the *shortpress_8.sh* script and activate GPS data capture. Edit `/etc/init.d/rcS_gps` and, after the lines:

```
eRide_aiding /data/ftp/internal_000/gps_data/eRide_data.bin &
fi
```

add the following block of code:

```
# furuno data capture if trigger file exists
if [ -f /data/ftp/furuno_data ]
then
 cat /tmp/gps_nmea_out >> /data/ftp/internal_000/log/furuno-nmea-out.txt &
 echo "${name} logging gps nmea out" | logger -s -t ${name} -p user.info
fi
```

The new code, checks for the existence of the 'trigger file' *furuno_data* in the `/data/ftp/` directory. If the trigger file is found at startup, *rcS_gps* will use the *cat utility*, run as a background task, to append the GPS FIFO data to file:

```
/data/ftp/internal_000/log/furuno-nmea-out.txt
```

which can directly be retrieved via FTP.
If the trigger file does not exist, *rcS_gps* will skip the highlighted block of code and the Bebop boot-up will proceed as usual.

Note that this capture strategy creates files that will all have the same name, which implies that data from a new flight will overwrite older data. You should therefore retrieve your file after each flight.
We can anyway introduce an extra mod to add a timestamp to the data file name, so that data will be protected from overwrite. To achieve this, we have to edit 2 more files.
First of all, create the following script and save it in `/bin`:

/bin/extract rmc date.sh

```
#!/bin/sh
#
#      extract the GNRMC date & UTC time from NMEA file
#      supplied via cmdline or from
#      /data/ftp/internal_000/Debug/current/myfileNmea.txt if it exists or file
passed in as a script parameter
#
#      we want the date/time from the first & last GNRMC sentence
#      output as ddmmyy-hhmm-hhmm
#
if [ $# -eq 0 ]
then
 file=/data/ftp/internal_000/Debug/current/myfileNmea.txt
else
 file=$1      # using cmdline file
fi

# find first GNRMC in file
first=`awk -F, '$1=="$GNRMC"{print $10 "-" substr($2,1,4) "-";exit;}' ${file}`
#echo ${first}

# find last GNRMC in file
last=`awk '/GNRMC/{k=$0}END{print k}' ${file} | awk -F ',' '{print
substr($2,1,4)}'`
#echo ${last}
echo ${first}${last}
```

Remember to set permissions with `chmod 755` and ensure that text line termination is Linux. This script pulls the date/time from the first and last GNRMC sentence that it finds in the *myfileNmea.txt* GPS debug file.

The last file to edit is the Bebop shutdown script:

```
/bin/ardrone3_stop.sh
```

After the lines:

```
# Shut down the GPS
echo '!' > /tmp/gps_debug &
```

Insert:

```
gps_file_path=/data/ftp/internal_000/log/
gps_file=furuno-nmea-out.txt

#check for furuno-nmea-out.txt, if file exists, use the extract_rmc_date.sh
script to insert a time stamp in the file name

if [ -e ${gps_file_path}${gps_file}  ]
then
  new_gps_filename="furuno-nmea-out-`/bin/extract_rmc_date.sh
${gps_file_path}${gps_file}`.txt"
  #echo ${gps_file_path}${new_gps_filename}
  mv ${gps_file_path}${gps_file} ${gps_file_path}${new_gps_filename}
fi
```

4.5.2. ARDrone3 v 3.1.0

The following method has only been tested on the Bebop 2 running ARDrone3 v 3.1.0. More tests are needed with v 3.2.0 and Bebop 1. Bebop 2 uses a *u-blox Neo M8N* GPS receiver. This receiver is configured to send a number of standard NMEA sentences plus a number of binary data packets. Although the GPS data can be captured in the same manner as the Bebop, handling the mix of ASCII and binary is a challenge. A better data capture method is provided by the Bebop 2 developers, as of FW 3.1.0. In using this method, the developers have converted the binary data to a hex ASCII format (with Linux line terminations), so that all available data is ASCII and readable in common viewers.
Before editing any file in the bebop 2, remember to enable read/write permissions to the whole file system, as described in par 4.2.1.

In Bebop 2, a GPS capture parameter has been added to the `/etc/debug.conf` file. This NMEA_DUMP parameter defaults to 0, which disables GPS data capture. By editing *debug.conf* and changing NMEA_DUMP=1, GPS data capture is enabled:

```
# Record nmea gps data into emmc
NMEA_DUMP=1
```

This records a copy of the NMEA GPS data in:

```
/data/ftp/internal_000/Debug/current/myfileNmea.txt
```

In order to prevent this from being overwritten at next startup, we need to copy *myfileNmea.txt* into a safe directory for later FTP retrieval. The file will be renamed and timestamped to make it unique and prevent overwriting.
We will deal with the required changes in reverse order. First the time stamping script (same as the one desribed above for ARDrone3 v 2.0.57).
The following script must first be created and saved in `/bin`:

`/bin/extract rmc date.sh`

```
#!/bin/sh
#
#      extract the GNRMC date & UTC time from NMEA file
#      supplied via cmdline or from
#      /data/ftp/internal_000/Debug/current/myfileNmea.txt if it exists or file
passed in as a script parameter
#
#      we want the date/time from the first & last GNRMC sentence
#      output as ddmmyy-hhmm-hhmm
#
if [ $# -eq 0 ]
then
 file=/data/ftp/internal_000/Debug/current/myfileNmea.txt
else
 file=$1      # using cmdline file
fi

# find first GNRMC in file
first=`awk -F, '$1=="$GNRMC"{print $10 "-" substr($2,1,4) "-";exit;}' ${file}`
#echo ${first}
```

```
# find last GNRMC in file
last=`awk '/GNRMC/{k=$0}END{print k}' ${file} | awk -F ',' '{print
substr($2,1,4)}'`
#echo ${last}
echo ${first}${last}
```

Remember to set permissions with `chmod 755` and ensure that text line termination is Linux. This script pulls the date/time from the first and last GNRMC sentence that it finds in the *myfileNmea.txt* GPS debug file.

You should then edit the Bebop shutdown script:

```
/bin/ardrone3_stop.sh
```

After the lines:

```
# Shut down the GPS
echo '!' > /tmp/gps_debug &
```

Insert:

```
# start of gord's mods...
# copy out nmea debug file if it's active
echo "copying Gord's debug files to eMMC internal_000/log/..." | ulogger -t
"shutdown" -p I
if [ -s /data/ftp/internal_000/Debug/current/myfileNmea.txt ]
then
 cp /data/ftp/internal_000/Debug/current/myfileNmea.txt
/data/ftp/internal_000/log/myfileNmea-`/bin/extract_rmc_date.sh`.txt
 sleep 3
 sync        # sync below comes after emmc release...
fi
# end of gord's mods...
```

When you shut down your drone, this code checks for non-zero length *myfileNmea.txt* file and if found, copies the file to

```
/data/ftp/internal_000/log/myfileNmea-ddmmyy-hhmm-hhmm.txt
```

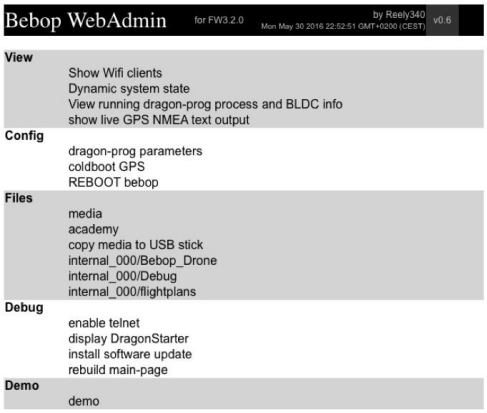
to make it unique. This file will not be deleted by the Bebop software, unless you do a reset (see par. 5.1) and can be retrieved at a later time via a FTP connection.

4.6. Dragon-prog replacement

Expert software developers have tried to edit the *dragon-prog* file itself to replace the original with a revised version that includes extra functions. Since this kind of editing is not accessible to the target users of this guide and openly violates software copyright, we have chosen not to present such modifications here. Those interested can easily find the necessary information on the web, including the official Bebop forum.

4.7. Bebop Webconfig

A very useful web interface has been created by Reely340 that can be installed on the Bebop and grants access to key information, files and *dragon-prog* options via any web browser. Please search the official Bebop forum for its latest release and detailed installation instructions. Among other functions, the current Bebop Webconfig release allows you to launch the *telnet* daemon, access internal directories (media, academy, debub), copy media to a USB drive, cold-boot the GPS ans display several diagnostic parameters.



5. In case of emergency

Hopefully you are reading this section to follow your curiosity while downloading the latest videos from your Bebop after an amazing flight session. If this is not the case, anyway, and you think you have messed up something, or the Bebop simply does not behave as expected, you can consider trying one of the following escape strategies.

5.1. How to hard reset the Bebop

One relatively frequent problem with Bebop is the lack of wifi connectivity after powerup. This may happen because of some problems during last shutdown (like unplugging the battery before the drone was completely shut down), or other oddities that sometimes happen even in the best operative systems.
Wifi connection problems are major issues, because you not only lose the ability to pilot the drone (either via Skycontroller or smartphone/tablet), but also cannot establish a telnet or FTP session to try and sort things out the geek way.
The Bebop user guide instructs you to reset your Bebop to factory settings by pressing the power button for 10 seconds. This activates the *verylongpress_0.sh* script which restarts the drone software by using factory copies of the configuration files.

As a visible feedback, the power button led goes through a series of flashes in different colors and the Bebop eventually shuts down (you can hear the fan stop) and reboots. This procedure has saved the life of several Bebops according to forum reports, and even restored malfunctioning files that caused odd behavior, so use it with confidence.
Remember anyway that a hard reset will erase your media files from the internal memory.

5.2. Using a failsafe script

A brilliant modification can save your day in case the Bebop wifi connection is unavailable as a consequence of some changes you have introduced. The idea is to save a copy of all the original configuration and service files that you intend to edit, in a dedicated directory in the Bebop filesystem; a power button-operated script (see par. 4.4) can then be used to replace all the files you have edited (likely causing the system failure) with the pristine version and reboot the Bebop. If the lack of wifi connectivity was due to your changes, you should now have fully recovered your Bebop functionality.
First of all create a directory to store a copy of the original files. We suggest to create it in the `/data/ftp/internal_000` directory (see par. 1.3), so that you can access it from your device or personal computer. We will call our directory *emergency*.
Open a telnet session, navigate to `/data/ftp/internal_000` and type:

```
mkdir emergency
```

Now copy all the files you want to backup with commands such as the following:

```
cp /usr/bin/DragonStarter.sh
/data/ftp/internal_000/emergency/DragonStarter.sh
```

Repeat for each file you plan to edit.

Now check that all your backup copies are in position with:

```
ls /data/ftp/internal_000/emergency
```

If we have created a backup of these files:

```
dragon_shell.sh
dragon.conf
system.conf
DragonStarter.sh
```

We will now enter their respective paths in the following script:

```
#!/bin/sh

echo "Restoring pristine files" | logger -s -t "Shortpress_9" -p user.info

# Red-Orange-Green LED
(BLDC_Test_Bench -G 1 0 0 >/dev/null; sleep 2; BLDC_Test_Bench -G 1 1 0 >/dev/null;
sleep 2; BLDC_Test_Bench -G 0 1 0 >/dev/null) &

# Remount filesystem with wrirte permissions
mount -o remount,rw /

# Copy saved files over edited/damaged files
### adjust this list to your own needs ###
cp /data/ftp/internal_000/emergency/dragon_shell.sh /bin/dragon_shell.sh
cp /data/ftp/internal_000/emergency/dragon.conf /data/dragon.conf
cp /data/ftp/internal_000/emergency/system.conf /data/system.conf
cp /data/ftp/internal_000/emergency/DragonStarter.sh /usr/bin/DragonStarter.sh

# Red-Orange-Red LED
(BLDC_Test_Bench -G 1 0 0 >/dev/null; sleep 2; BLDC_Test_Bench -G 1 1 0 >/dev/null;
sleep 2; BLDC_Test_Bench -G 1 0 0 >/dev/null; sleep 2) &

# reboot
./bin/reboot.sh &
```

The final step is to save this script in the `/bin/onoffbutton` directory with an appropriate file name (par 4.4.1). For example, `/bin/onoffbutton/shortpress_9.sh`
After restart, 9 short presses on the power button will launch the failsafe script and restore the included files from the backup you made in the emergency directory.

Unfortunately this strategy is preemptive: you must set everything up when the Bebop is fully functional for the script to be ready in case of necessity. But if you go this way you should be able to reset your Bebop to its pristine conditions with a simple action on the power button.

5.4. USB networking

USB networking creates a TCP/IP network through the UART port (see par. 5.4.1). When the wifi connection is in place, USB networking can be activated by launching the script:

```
/bin/usbnetwork.sh
```

The same script is run by short pressing 13 times the Bebop power button, which is what you should do in case you have definitely lost wifi connectivity. The Bebop's IP address on the USB/UART network is 192.168.43.1. This action is also reported to force-start the telnet daemon, enabling telnet connectivity through the wifi network in case it was lost due to previous issues.

In Bebop 2 running ARDrone3 v 3.0.6, USB networking has been reported to be activated over the normal micro-USB port, but this remains to be verified for the Bebop Drone.

5.4.1. How to connect via UART

An extreme measure in case you are still unable to connect to your Bebop is to exploit a hardware hijack that implies plugging cables into ***unused contacts*** on the the Bebop motherboard (UART port). This port has direct access to the motherboard and can be used to connect the Bebop to a USB cable when no other method can establish a connection via wifi. The whole procedure is explained in full detail here (scroll down to the end of the page).

5.5. Forcing firmware reinstall

Possibly the most powerful action to restore a damaged filesystem, e.g. in case you are not able to connect via wifi any more, is to reinstall the whole software. It is normally impossible to reinstall ARDrone once you already have the latest release. Nevertheless, a very smart user has found a workaround that will actually let you even reinstall older firmware versions. Proceed as follows:
Download the firmware update file from the Parrot site and save it to your computer hard drive. Now use a hex editor like UltraEdit or any other equivalent software (there are several free to download) to open and edit the `.plf` file containing the firmware.
Once you have opened the file, search for the offset 36 and set it to a higher value. For example, in ARDrone3 v 3.x, offset 36 reads 03 and is followed by three 00 (highlighted in red below).

```
50 4c 46 21 0d 00 00 00 38 00 00 00 14 00 00 00
02 00 00 00 00 00 00 00 06 00 00 00 68 00 00 00
5c 64 55 2a 03 00 00 00 02 00 00 00 00 00 00 00
00 00 00 00 58 7a 1c 01 0b 00 00 00 58 02 00 00
74 50 0e c8 00 00 00 00 00 00 00 00 00 00 00 00
```

Changing 03 to 04 will make this file appear to the Bebop system as a more recent update (to version 4.x).
You can now choose two different strategies:

1) copy the `.plf` file to a USB stick (must be in FAT32 format); plug it into the micro-USB port of your drone using a suitable cable, then turn the drone on. During boot-up, a check is made on the USB bus and if a suitable `.plf` file is found, the install procedure is launched. Bingo!

2) Alternatively, if FTP connectivity over wifi is still available, you can open a terminal window, then type:

   `ftp 192.168.42.1 51` (note the space before 51)

   This will open a telnet session over port 51, granting access to a directory that is used for software updates. Copy your edited file to your computer home directory, then type:

   `put filename.plf` (use the name of the file you edited)

   This will copy your file to the Bebop memory.
   You can now close your FTP connection and reboot the drone. You will see from blinking LED colours that the firmware is being installed. Bingo, again!

WARNING: it is not recommended to switch back and forth between firmware updates. Apparently some files may get corrupted and cause dramatic and critical misbehavior in your Bebop. Refer to the official Bebop forum for further details

---

5.6. Crash debugging

As a flying computer, your bebop is not immune to bugs and crashes that can affect its hardware and software. While your desktop PC will at worst freeze or shut off, when such problems arise during flight, the consequences can be varied, but always stressful.
Assuming you are abe to recover your drone and access its internal memory, you may want to investigate on what cused the problem.
In both models, `.zip` files are produced during each power-on session and contain a lot of useful information about the use of sensor inputs and software status before, during and after flights. Such files are cyclically replaced and the bebop Drone will always keep the last 4 files, whereas this number was raised to 10 in Bebop 2. They are stored in:

```
/data/ftp/internal_000/Debug/archive/
```

It is therefore easy to access and download them via *FTP* to your personal computer for processing.
Files must be decompressed (*unzipped*) and extracted (*tar*) to obtain a binary file named

```
ckcm.bin
```

The binary contains several readable ascii strings, which give useful hints for your investigation.

If you are using Linux or MacOS, running the *strings* command on *ckcm.bin* from a shell window extracts and prints out all the ascii strings contained in the file (20-30,000 lines). A search (*grep*) on this output can bring the useful information in sight.

5.6. Disconnections

The unfortunate case of a wifi disconnection during flight is a frustrating experience that most pilots have experienced at least once. So, even if this guide is devoted to hacks and modifications, a short discussion on how to prevent and respond to disconnections may be of use to everyone.

5.6.1. Types of disconnections

The most feared type of disconnection is the so-called flyaway. Needless to say, the Bebop banks or drifts out of your control and into an obstacle, the sea, or beyond your line of sight. This type of problem may be caused by radio interference, GPS problems, local geo-magnetic conditions, or a malfunction in the software that manages such sensors. Your best chance in such cases is that the drone bumps into a nearby soft object. In most cases video connection is still active, so you can 'see' where your drone is going and - hopefully - where it fell.
Occasional reports can be found on the internet of Bebops suddenly falling from the sky. What causes this is not clear: a software crash could be held responsible, but hardware problems, such as a stuck motor, a broken blade or the collision with a bird, have also been envisaged. In such cases your problem is not really the disconnection, but the cost of spare parts and possible damage to others.
A more common - and far less dramatic - type of disconnection leaves some hope, anyway. Symptoms are crystal clear: loss of video feedback; loss of control from the Skycontroller sticks; if you are using a direct connection from your smartphone or tablet, you will also get a connection lost error message. Eventually, a message 'connecting to Bebop…' will make your nightmare come true. What happens then in most cases is that the Bebop will hang in the air for a few eternal minutes, until the battery drains out and the drone lands elegantly and undamaged, assuming it was hovering above a clear area; if this happens over water… well, YouTube is a great source of videos with desperate pilots that plunge into cold waters trying to catch their drones before the fatal splash (this is not limited to Bebops). What else can you do? In fact there are a few hints that can be useful in such situations.

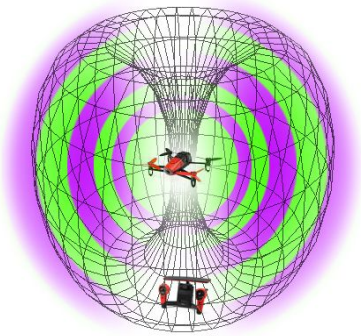
Keep calm and think

The Bebop extraordinary self-control will keep it from drifting with the wind. The combined input by the GPS chip, vertical camera and accelerometers will make your drone stick to its position quite impressively. So, unless it is hovering a few feet from an obstacle or in wild winds, you can take your time and use the remaining battery life for something more practical than just stamping your feet in despair.
Disconnections happen because of various causes. Wifi congestion and radio interference appear to be the most frequent. The Bebop and the controller are set to consider the respective wifi network as the strongest signal in the area. Anyway, they both scan the wifi spectrum in search for less congested channels to switch to, and so on. As soon as the drone-controller signal is weaker than another wifi signal present in the area, both the drone and the controller will start

pondering whether the stronger signal is the network they should actually be connected to, send over a handshaking to the router,etc. All such activity engages the wifi band, further weakening the true connection. In fact, if you fly in an isolated place, the chances to get disconnected are very close to zero - think about this when you choose your next flight site.
In all such disconnection cases, the Bebop hovers on site, waiting for a new connection. The problem is, your device may not 'want' to reconnect, even if onscreen messages say the opposite. This may be caused by errors in the wifi settings files that took place during the loss of signal.

What to do
One easy - albeit counterintuitive - possibility is to ***power off your Skycontroller and switch your device's wifi off***. Remember the Bebop has a return-to-home function, and 60 seconds after disconnection (you can adjust this timer in the FreeFlight3 preferences), it will raise to 20m (or the highest altitude reached before disconnection) and fly back to you autonomously, coming to a stable hover 2m off the ground. The reason you should switch your devices off line is that the drone might 'sense' their attempts to reconnect and stop the RTH timer each time.

In some cases, though, the disconnection takes place just a few meters from your position. RTH does not engage in such short ranges. In fact, short-range disconnections have been related to the infamous ***wifi black hole***, rather than radio congestion or interference. In short, the antennas that lay in the feet of your Bebop generate a wifi field that has the shape of a tall donut, with a 'hole' more or less placed underneath the drone (and wobbling around depending on the drone pitch and roll); consequently, if your controller falls within the wifi black hole, it will just lose its connection. In this case it is recommended to ***walk several meters away*** from your Bebop, in order to get out of the hole and be in condition to try and reconnect.



Assuming you have your drone sitting in the sky several meters away, not returning to home on its own aven after you have switched off your controller and device, how can you get back in control? The most successful strategy appear to be the following (assuming you are using a Skycontroller):

- Go to your ***device wifi settings*** and remove the drone and Skycontroller wifi IDs from your list of preferred networks, or whatever they are called in your operative system. You must have your device forget it has ever connected to them.

- Then reconnect to the Bebop wifi, if you see it, directly from your device. If not, connect to the Skycontroller.

- If you are lucky, the Skycontroller will beep its connection to the Bebop as usual.

- If this does not happen, close and relaunch FreeFlight3. Since it cannot connect, you will see the wifi manager window in the top left frame. Open it. ***Delete your drone wifi ID*** from the list of known networks.

- Close and relaunch FreeFlight3. If it does not reconnect on its own, open the wifi manager again and ***manually select your Bebop ID***.

If all this proves unsuccessful, be conscious that you have done your best and you have been able to keep your temper and act reasonably in a situation of emergency. You have become a better person. This will undoubtedly comfort you from seeing your drone diving in the ocean.

6. SkyController

The SkyController (SC) is unfortunately inaccessible to hacks and mods, since it has no telnet running by default. We anyway present here a bunch of information that can turn out useful to advanced users.

6.1. SkyController software

Similarly to the Bebop user guide, also the SC user guide from Parrot is pretty poor of information. One thing you should know is that also the SC is an android-operated computer. As such it has its own OS (based on Android 4.2.2), which is booted every time you activate the power button (the flashing LED dance is the equivalent of a progress bar).
In fact, the SC can run apps, and is set to run FreeFlight at startup. As a matter of fact, when you use your mobile device with the SC, your device FreeFlight app is communicating to the SC FreeFlight app through a short-range wi-fi network (*wlan1*); in turn, the SC FreeFlight app is in control of the Bebop through a second, long-range wi-fi network (*wlan0*); the video from the Bebop is streamed through the long-distance network, then passed through to your tablet or smartphone. Device-to-SC communication also includes the transmission of touch controls operated on the screen.
You should already know that the HDMI port can be used to plug FPV goggles or an HDMI monitor/TV. If you do so, the screen will show the SC FreeFlight interface, which is similar but clearly distinct from the android or iOS version. One of the SC knobs can be used for moving a pointer on the screen and navigate through the app menus and launch a flight session. The use of HDMI goggles or monitors strongly reduces lags in live video because - by plugging directly into the SC - the second step of wi-fi video broadcast is avoided.

6.1.1. The SkyController filesystem

Parrot ARDrone3 SDK is a free software package that is meant for Android app developers and grants access to the SC file system via an adb connection over USB. The file system is extensive and there could be hidden treasures (e.g. setting files abilitating extra features).
In particular, the `/system/xbin` directory shown below appears to contain interesting files, although we have no indication that this kind of access is of any use to non-developers.

`/system/xbin`:

```
dd-property-tag      cpustats       flock     kill        mount                renice           stat        uptime
ppWatchdog           cut            free      killall     mpstat               reset            strace      usleep
sh                   date           fsck      klogd       mv                   resize           strings     uudecode
th6kl_reload         dc             fsync     latencytop  nanddump             rev              stty        uuencode
th6kl_usb_loader     dd             ftpd      led_bootanim nandwrite           rm               su          vi
tmegCheckUpdate      deallocvt      ftpget    less        nbd-client           rmdir            sulogin     vlock
tmeg_boot_check.sh   devmem         getopt    librank     netstat              rmmod            switch_root volname
wk                   dexdump        grep      linuxrc     omxregister-bellagio route            sync        watch
ase64                df             groups    ln          opcontrol            run-parts        tail        wc
asename              dfu-programmer gunzip    logcol      openvt               rx               tar         wget
lockdev              dhcprelay      gzip      logkeeper   oprofiled            sane_schedstat   tee         which
ootchartd            diff           halt      ls          passwd               scriptreplay     telnet      whoami
tool                 dirname        head      lsattr      pidof                sed              telnetd     whois
usybox               dmesg          hexdump   lsmod       ping                 setWifiBitrate   test        wifidriverloaded_hook.sh
at                   dos2unix       hwclock   lsof        pinst_trigger        setWifiParameters time       wpa_supplicant_launcher
hattr                du             i2ctool   lspci       pinst_version        set_reg_domain   top         xargs
heck-lost+found      dumpleases     id        lzop        pmap                 setleds          touch       xz
heckData             echo           ifconfig  lzopcat     poweroff             setserial        traceroute  xzcat
heckProperties       ed             init      md5sum      powertop             setsid           true        yes
```

```
heckplf             egrep         initAccessPoint       mdev                procmem           sh          tty
hgrp                env           insmod    micro_bench procrank            sha1sum           udhcpc      zcat
hmod                expr          install   mkdir       ps                  showmap           udhcpd
hown                false         iostat    mke2fs      pstree              showslab          umount
hroot               fbset         ip        mkfs.ext2   pwd                 sii5293_tools     uname
hvt                 fdformat      ipaddr    mknod       pwdx                sleep             uniq
lear                fdisk         iplink    mktemp      rawbu               smemcap           unix2dos
mp                  fgconsole     iproute   modinfo     rawjsread           sort              unlzop
ountryCodeConverter fgrep         iprule    modprobe    readlink            sqlite3           unxz
p                   find          iptunnel  more        reboot              start-stop-daemon updater_root.sh
```

Since the SC is our ‘hand in the sky’ guiding the Bebop, we strongly discourage fiddling with SC internal files unless you know very well what you are doing.

6.2. SkyController hardware

The SC sports its own sensors, including a magnetometer, accelerometer, gyroscope. A combination of sensor output is used to calculate the SC orientation in space, which is used by Parrot as well as third party apps.

6.2.1. USB mouse

You can plug a USB mouse in the SC USB port and use it to navigate the Freeflight app, which - for bench work such as software updates etc. - is far more practical than using the little black knob.

6.3 Resetting the SkyController

Some forum posts report that different combinations of button presses are supposed to factory reset the SkyController. None of them seemed to be doing anything in our tests, and it is possible that this function has been removed in recent firmware updates.
Nonetheless, a post reports the reply obtained from Parrot support, which is interesting and worth mentioning here. The problem reported by the user concerned an abnormal behavior of the joysticks, leading to drone drifting to one direction when the right stick was centered. Here is Parrot’s reply:

> *Joystick calibration offset reset is now possible (see procedure below)*
> *--- Do not touch the Skycontroller's joysticks during the whole procedure ---*
>
> *1) Update your SkyController to 1.3.8 version* (this was the latest version available at the time)
> *2) Create an empty text file "calibrate_me.txt" on a USB key*
> *3) Plug the USB key into your SkyController*
> *4) Turn on the SkyController and wait until FreeFlight is launched: do not touch the SkyController joysticks*
> *5) Unplug the USB key*
> *6) Reboot the SkyController*

This procedure has successfully been used also on 1.6.6 firmware and is likely to fix your joysticks in case you are experiencing similar issues but, most importantly, it tells us that an empty text file with the right name is able to access the SkyController system. It will be interesting to experiment with alternative file names that could for example open a telnet connection over the wifi network and avoid the SDK approach.

7. Acknowledgements

We herewith acknowledge the whole community of Parrot Bebop users who published the information that we assembled in this guide in several official and unofficial forums, websites and blogs.
The Unofficial Bebop hacking Guide includes contributions from:

AO
coolys
enderffx
Eric78
Foomanchu
Foundobjectwind
Island_Bob
kfj4100
Matthewlloyd
mbarnes
mountkidd
nicknack
Parrot_Axel
Parrot developers blog
RAR69
Reely340
rodsantos
ShellDude
SilentException
videokahuna
Volodiaja
zeroprobe