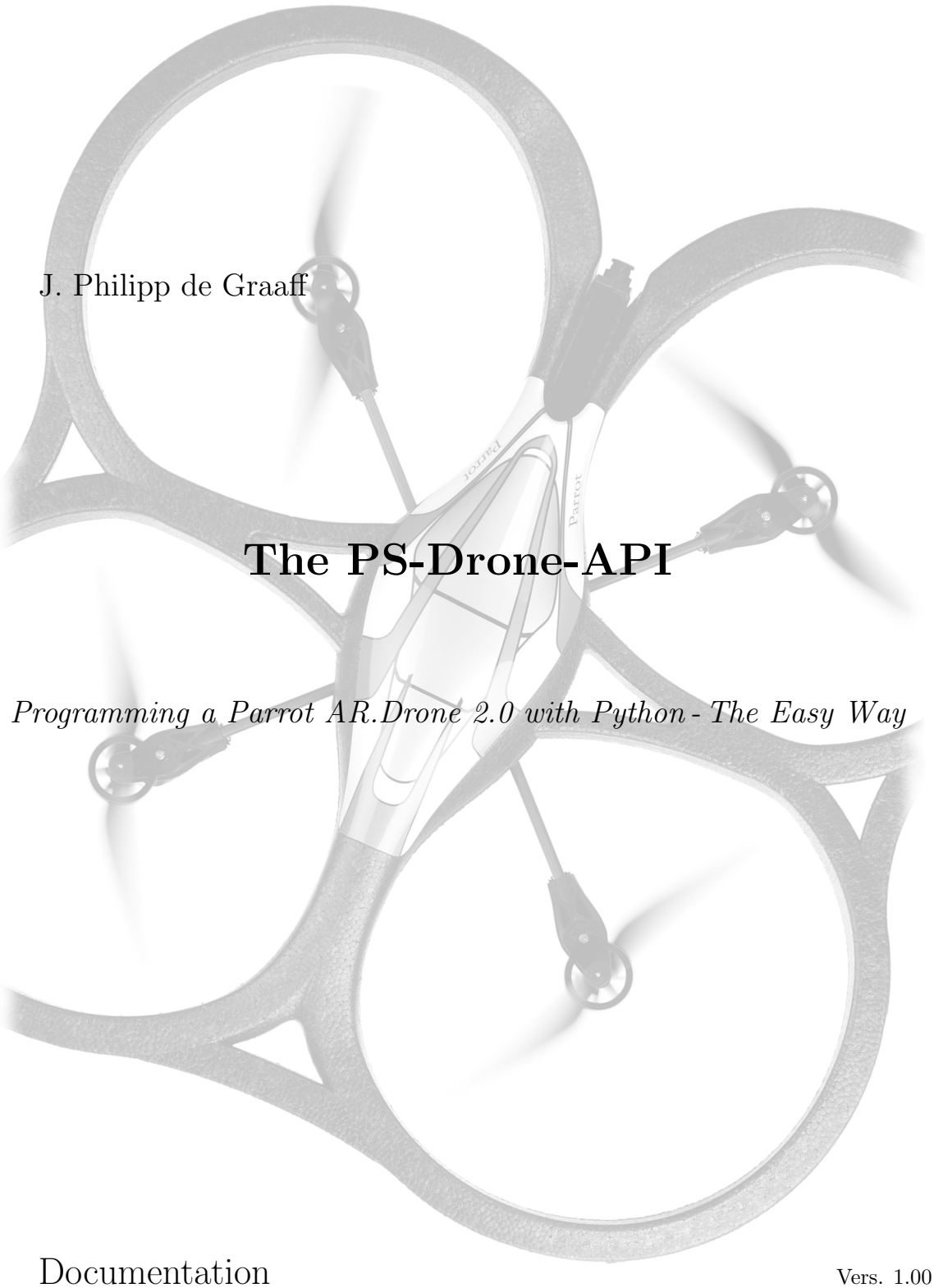
J. Philipp de Graaff




# The PS-Drone-API

*Programming a Parrot AR.Drone 2.0 with Python - The Easy Way*

Documentation

Vers. 1.00

An open source project, available on **www.playsheep.de/drone**

# Contents

# List of Tables

# Listings

# 1 Introduction

PS-Drone is a full featured API, written in and for Python, for Parrot's AR.Drone 2.0. It is based on a part of a master of computer science degree dissertation and was designed to be easy to learn, but it offers the full set of the possibilities of the AR.Drone 2.0, including Sensor-Data (aka NavData), Configuration and full Video-support.

It was also designed to be as fast as Python can be. Unneeded functions can be deactivated which allows your program to run on really slow computers like Intel Atom N270 with 1.6GHz. The video function is not restricted to mere viewing; it is also possible to analyze video images data using OpenCV2.

Obviously, the PS-Drone is perfect for teaching purposes; however, even the requirements for professional purposes can be satisfied. The nomenclature refers to the official definition of parrot and its SDK, so it is easy to find some additional help in newsgroups or elsewhere on the internet by simply typing in the pre-existing keywords.

The examples are easy to understand for people with little programming experience. This Tutorial is a brief introduction for using this API, its most important commands and the most important sensor values sent by the drone. For a full list of the commands and a description of the sensor data, have a look to the PS-Drone documentary, starting in chapter 3, page 24.

PS-Drone is not compliant to Parrots AR.Drone 1.0 ! It could be possible to fly and do simple movements, but it is not possible to get any sensor-values. The PS-Drone-API does not support new models like the Bebop aka AR.Drone 3.0, due significant changes of Parrot's communication-protocol. I will provide full support as soon as I get enough donations to buy a Bebop-drone.

A port to Windows is planned, but compared to Linux, Windows does not support some relevant functions. These functions have to be implemented manually.

I am not a money-grubbing monster, but it took me several months and killed two drones to create PS-Drone and its documentation; so it would be nice to get some donations for further development (e.g. for Parrot's Bebop) and as a appreciation.
Beware, stinginess may lead to bad breath of your dog, sexually transmitted disease and watching reality-shows while listening to the greatest hits of the worst boy-groups for eternity !Think about it, a dollar is not too much. *nudge*

PS-Drone is available for free on **www.playsheep.de/drone**

## 1.1 Disclaimer and License

## 1.2 Requirements

To program the drone you need a computer or virtual machine with a POSIX compliant operating system like Linux and an AR.Drone 2.0 of any edition. OpenCV2 is recommended and necessary for video-support. To use all functions without limitation a CPU equal 1.83GHz Intel Core Solo T1400 is needed, a 2.4GHz E6600 Core 2 Duo CPU or better is recommended for video-analysis.

PS-Drone was programmed and mostly tested on a computer with an Intel Atom N270, running a Linux Mint 13. The used drones were standard AR.Drones 2.0 by Parrot using Firmware 2.2.6 and 2.3.3.

## 1.3 Version

**2.0.0 beta**: Transferred source code from PdG-Drone to PS-Drone, some bug-fixes, changed start-up-sequence, more consistent command names, functions added, source code clean up, enhanced comments in source-code, multiple drone controlling, video initialization recovery. (Nov 2014)

**2.0.0**: Several minor bug-fixes and checked orthography. (Dec 2014)

**2.0.1**: Several major bug-fixes in movement-commands. Only perform the reset-command, if the drone is in fail-mode. Added missed `State`-bit. Error-messages regarding additional NavData for AR.Drones 2.0 with firmware version 2.4.8 are patched. (Dec 2014)

# 2 Tutorial

It is not as complicated as it might look at first sight. First, make sure you are running a POSIX compliant operating system, Linux is suggested, and make sure that python 2.x is installed. In case you want to use the drone's video, is also necessary to install OpenCV2. If you do not have any experience with Linux, try Knoppix (*http://www.knopper.net/knoppix/index-en.html*), the Linux Mint 17 live Image (*http://linuxmint.com*) or install a Linux (Linux Mint 17 MATE Edition is pretty nice) on a virtual machine (f.e. virtualBox (*https://www.virtualbox.org*)).

Parrot's AR.Drone 2.0 is by default a flying WiFi-hotspot. By default, it communicates unencrypted on a channel named *ardrone2_xxxxxx*, using IP *192.168.1.1* and offers a DHCP-Server, which means a client will (regularly) get IP *192.168.1.2*.
As a suggestion, in case you are using a LAN-connected PC, attach a wifi-router in client mode to connect to the drone and add an additional IP-address (e.g. *192.168.1.9*) to your PC.

## 2.1 First flight

When every everything is set up, start directly the PS-Drone-API-program which includes a simple demo program how to control the drone with your keyboard:

*python ps_drone.py*

| *Key* | Movement |
|---|---|
| <*space*> | take-off or land |
| <*0*> | stop moving |
| <*w*> | move forward |
| <*s*> | move backward |
| <*a*> | move left |
| <*d*> | move right |
| <*q*> | turn left |
| <*e*> | turn right |

Try pressing numbers, <*>, <+>, or <->, any not listed key will land the drone and end the demo.

When you are done, take a look at the first example *firstTry.py*.

```
import time
import ps_drone             #Imports the PS-Drone-API

drone = ps_drone.Drone()    #Initials the PS-Drone-API
drone.startup()             #Connects to the drone and starts subprocesses

drone.takeoff()             #Drone starts
time.sleep(7.5)             #Gives the drone time to start

drone.moveForward()         #Drone flies forward...
time.sleep(2)               #...  for two seconds
drone.stop()                #Drone stops...
time.sleep(2)               #...  needs, like a car, time to stop

drone.moveBackward(0.25)    #Drone flies backward with a quarter speed...
time.sleep(1.5)             #...  for one and a half seconds
drone.stop()                #Drone stops
time.sleep(2)

drone.setSpeed(1.0)         #Sets default moving speed to 1.0 (=100%)
print drone.setSpeed()      #Shows the default moving speed

drone.turnLeft()            #Drone moves full speed to the left...
time.sleep(2)               #...  for two seconds
drone.stop()                #Drone stops
time.sleep(2)

drone.land()                #Drone lands
```

Listing 2.1: Sourcecode of sample *firstTry.py*

Before *drone.startup()*, it is possible to configure the PS-Drone, for example, to change the drone's IP. Please take a look at the documentary, in chapter 3, page 24, for all options.

*startup()* connects to the drone, configures the basics and activates the sensor-datastream also known as "NavData". A documentation about the drone's sensors and the returned values will also be in this documentary, in chapter 3.4, page 29.

The command *takeoff()* makes the drone fly. After the drone has reached its hovering state (at around 75cm), it runs some internal calibrations. However, usually the take-off-sequence takes around seven seconds altogether, movement-commands before a clearly completed take-off might cause unexpected (but not critical) behavior.
The commands *land()* makes the drone land.

Note: Only these commands enable the drone to take off, go up into the air, descend again and finally come back to the ground, just using the commands *moveUp()* or *moveDown()* would not bring the expected success.

The basic movement-commands are:

```
moveForward(val)
moveBackward(val)
moveLeft(val)
moveRight(val)
moveUp(val)
moveDown(val)
turnLeft(val)
turnRight(val)
stop()
```

The values for these basic movement-commands determine the speed of the movement and are optional. The values should be between `0.0` and `1.0` as they represent the drone's speed from 0% to 100%. If you do not set the speed-value, the drone will move with its default speed. You can alter this default speed (which is preset by `0.2`) by using the command `setSpeed(val)`.

The command `stop()` stops the drone's movement and makes it hold its position. Setting a movement-command with the speed-value `0.0` will not have the same effect; the drone will glide in its last (horizontal) direction, just getting slower due to air drag (see also next section for further information).

Due to its firmware, it is not possible for the drone to hold its exact position while turning on its own axis; it tends to move sideways. The extent of these unwanted movements depends on the condition of the propellers and the drone in general (the more damaged, the stronger the drifting-effects).

By default, the AR.Drone 2.0 continues to perform the last movement-command forever, until it receives a new movement-command. Therefore, PS-Drone has "self-control" as a security feature. It means that if your program crashes, this API detects it, sends the command to land to the drone and shuts it down properly. In case the PS-Drone has no chance to react (for example because your computer crashed or the drone has flown too far away) you are doomed ! By the way: Houseplants and cats do not like being attacked by an uncontrolled drone on the rampage - believe me !
However, it is not necessary to land and shut down the drone at the end of your program.

## 2.2 Glide vs stop and a better startup-sequence

In this section, the term "thrust" is also used for speed; because, basically, speed is caused by thrust.

The following example `glideNstop.py` shows the difference between active stopping and holding position on the one hand; and reducing the thrust to `0.0` on the other. Active stopping will cause the drone to hold position, while giving no thrust will make the drone to glide on; it will just get slower due to air drag. If you want to experience the difference between the two yourself, start the program, push the flying drone and see what happens.

```
##### Suggested clean drone startup sequence #####
import time, sys
import ps_drone            #Imports the PS-Drone-API

drone = ps_drone.Drone()  #Initials the PS-Drone-API
drone.startup()           #Connects to the drone and starts subprocesses

drone.reset()             #Sets drone's LEDs to green when red
while (drone.getBattery()[0]==-1):  time.sleep(0.1)   #Reset completed ?
print¨Battery:¨+str(drone.getBattery()[0])+¨%¨+str(drone.getBattery()[1])¨
if drone.getBattery()[1]==¨empty¨:  sys.exit()

drone.useDemoMode(True)       #15 basic datasets per second (default)
drone.getNDpackage([¨demo¨])  #Packets, which shall be decoded
time.sleep(0.5)               #Give it some time to fully awake

drone.takeoff()               #Fly, drone, fly !
while drone.NavData[¨demo¨][0][2]:  time.sleep(0.1) #Still in landed-mode?

##### Mainprogram #####
print ¨Drone is flying now, land it with any key but <space>¨

gliding = False
print ¨Drone is holding its position, toggle to glide with <space>-key.¨

end = False
while not end:
    key = drone.getKey()            #Get a pressed key
    if key == ¨ ¨:
        if gliding:
            gliding = False
            drone.stop()            #Stop and hold position
            print ¨Drone is holding its position¨
        else:
            gliding = True
            drone.moveForward(0)    #Do not fly actively in any direction
            print ¨Drone is gliding¨
    elif key:  end = True           #End this loop
    else:      time.sleep(0.1)      #Wait until next looping
```

Listing 2.2: Sourcecode of sample *glideNstop.py*

For a clean start up some additional code has been added, because a just time-based controlling is rarely not a good solution. As a suggestion just copy and paste this start up-sequence to your own code.

The first lines remain more or less the same. Resetting the drone makes it work again, if the drone's firmware blocks flying, indicated by red LEDs. A reset takes a while and the drone does not send data in the meantime. When sending data again (the battery status will be checked here), the drone has almost finished, but still needs a little time.

The commands *useDemoMode(val)* and *getNDpackage(val)* will be explained later, the values *True* and *[¨demo¨]* are standard-settings anyway.
However, it could happen that you require more data and faster. Therefore, a brief explanation beforehand: The drone sends its sensor-/NavData in a large bunch. This includes several packets called *¨demo¨*, *¨time¨*, *¨altitude¨* and so on, with specified data. In demo-mode, there are only the packages *¨demo¨* and *¨vision_detect¨* available,

both 15 times per second; most of the time, this is enough and perfect for learning or debugging. Switching off the demo-mode means all NavData-packages are available, 200 times per second.
Available NavData-values are always stored in an array called *¨NavData¨*. First dimension is the name of the small package the values are stored in. Position *[0][2]* stores the drone's *landed*-status-tag. If not set anymore, the drone is ready for action.

It has already been mentioned, that a take-off takes around seven seconds. However, this example uses the drone's flight-status. There will be a further explanation later.

The command *getKey()* has been added for convenience and has technically nothing to do with the drone. Use it to get the value of a pressed key, without worrying about timing, deadlocks and other annoying effects.
Notice that there is no command to land the drone, PS-Drone does it for you at the end of your program. The example stops when a key was pressed that is not *<space>*.

It is recommended to implement an emergency stop in your code which will land the drone by pressing a key. Also a powerpoint-remote control is suggested in order to be more mobile; a pressed key on that remote control is like pressing a key or a key chain on the keyboard and simple to implement.

## 2.3 Enhanced movement

It is now time for the last important movement and movement-related commands as shown in example *moveTurn.py* in listing 2.3.

The *move()*-command combines all the basic movements that have already been introduced. So you can make the drone fly to the right, backwards, down and turn to the right at the same time; and with different speed in each direction.

The *move()*-command expects four values for the directions *'right'*, *'forward'*, *'up'* and *'turn right'*: `move(right, forward, up, turn right)`

Like the basic movements, the range for the speed-value is *0.0* to *1.0*, representing 0% to 100% of the drone's top speed. By setting a negative value, the drone will reverse its direction; for example, the value *-0.5* for forward-speed lets the drone fly backwards.

The command *turnAngle()* lets the drone turn to the left or the right in the exact given degrees. This command requires values for a certain degree in order to turn to the right and the speed to do so. If you use a negative degree value, the drone rotates to the left. *turnAngle()* blocks which means that following commands are not processed until *turnAngle()* has been executed. So there is no clean way to stop the movement, for example as an emergency stop, until its done.
Optional, you can set a value for accuracy, the allowed mismatch of the given degree. The default value is $\pm 0.005°$ and $\pm 0.1°$, depending on the drone's Demo-Mode-status. The less accurate the turn is, the faster is it done.
The maximum turn is a half rotation; the angle-value can be at most $\pm 180.0°$.

*Known bug*: this command is not able to perform an exact turn of $180.0°$; instead, the drone tries to find its destination angle until the battery is dead and without the possibility to stop the drone neatly. This bug will be fixed soon.

```
##### Suggested clean drone startup sequence #####
import time, sys
import ps_drone            #Imports the PS-Drone-API

drone = ps_drone.Drone()  #Initials the PS-Drone-API
drone.startup()           #Connects to the drone and starts subprocesses

drone.reset()             #Sets drone's LEDs to green when red
while (drone.getBattery()[0]==-1):  time.sleep(0.1)   #Reset completed ?
print¨Battery:¨+str(drone.getBattery()[0])+¨%¨+str(drone.getBattery()[1])¨
if drone.getBattery()[1]==¨empty¨:  sys.exit()

drone.useDemoMode(True)        #15 basic datasets per second (default)
drone.getNDpackage([¨demo¨])  #Packets, which shall be decoded
time.sleep(0.5)                #Give it some time to fully awake

drone.trim()                   #Recalibrate sensors
drone.getSelfRotation(5)       #Auto-alteration-value of gyroscope-sensor
print ¨Auto-alternation:  ¨+str(drone.selfRotation)+¨deg/sec¨

drone.takeoff()
while drone.NavData[¨demo¨][0][2]:  time.sleep(0.1)

##### Mainprogram #####
print ¨Drone is flying now¨

leftRight      =-0.02         #Move with 2% speed to the left
backwardForward=-0.1          #Move with 10% speed backwards
downUp         =0.3           #Move with 30% speed upward
turnLeftRight  =1             #Turn full speed right
drone.move(leftRight, backwardForward, downUp, turnLeftRight)  #Do movement

timeToFlight   =2.5           #Time to move at all
refTime        =time.time()   #Start-time
end            =False
while not end:
    if drone.getKey():  end=True
    if time.time()-refTime>=timeToFlight: end=True

print ¨Drone stopped movement¨
drone.stop()
time.sleep(2)

print ¨Drone turns 120 degree to the left¨
drone.turnAngle(-120,1,1)     #Turn 120° to the left, full speed, 1° accuracy
```

Listing 2.3: Sourcecode of sample *moveTurn.py*

In order to hold position and not drift away slowly in any direction, the drone's sensors for orientation need to be calibrated on horizontal ground. The drone's firmware does a good job recalibrating these sensors in case of inaccuracy, using the sensors for acceleration. If it is necessary to recalibrate the sensors later, use the command *trim()*. Use it before *takeoff()* and give the drone a second to calibrate.

Technically, the accuracy of a sensor's measurements differs from sensor to sensor also depending on external influences like temperature. So the gyroscope-sensor for rotation slowly alters its value, even if the drone is on the ground and not moved.
If it is necessary to be exact, you can get the false measured self-rotation by the command *getSelfRotation(val)* in degrees per second, stored in variable *selfRotation*.

This command expects the time to measure as value. Of cause: the more time you give to measure the more accurate is the result. `turnAngle()` considers the value of `selfRotation` for exact turns.

Note: With the firmware version 2.2.6, the automatic value alteration of the gyroscope-sensor was around 0.0185°/sec. Parrot fixed this auto alteration with the firmware version 2.3.3, but now the entire drone rotates, about 360° in 20 minutes, without any recognition of the gyroscope-sensor. If you yet have not updated, the suggestion is to stay with version 2.2.6. Firmware-version 2.4.8 has not tested, yet.

## 2.4 Using the drone's sensors

Parrot calls the AR.Drone sensor data "NavData", so we do, too.

Starting with some theory:
NavData is sent in one block including the sensor-measurements and some status informations it consists of bunch of 28 packages, each containing a specific set of values. The packages are named in this API as they are named by Parrot in their SDK.

The available packages are:
*¨demo¨*, *¨time¨*, *¨pwm¨*, *¨raw_measures¨*, *¨phys_measures¨*, *¨gyros_offsets¨*, *¨altitude¨*, *¨magneto¨*, *¨euler_angles¨*, *¨references¨*, *¨rc_references¨*, *¨vision_detect¨*, *¨vision¨*, *¨vision_raw¨*, *¨vision_of¨*, *¨vision_perf¨*, *¨trackers_send¨*, *¨video_stream¨*, *¨hdvideo_stream¨*, *¨games¨*, *¨trims¨*, *¨pressure_raw¨*, *¨wind_speed¨*, *¨kalman_pressure¨*, *¨watchdog¨*, *¨wifi¨*, *¨adc_data_frame¨*, *¨zimmu_3000¨* and *¨chksum¨*.

For some reasons from, I am going to speak about *"time-units"* now. A time-unit (TU) is the time from getting a NavData to getting the next. The drone knows two modes for sending its NavData, a "demo"- and a "full"-mode. Depending on its mode, there are 15 or 200 NavData sent per second which means that a second is divided into 15, respectively 200 time-units.

The AR.Drone 2.0 sends its NavData in a UDP-package. This means that the drone keeps on sending without caring about whether its NavData is being received. So there is a possibility of lost or false information. However, it has become apparent that no false NavData is received, but 3% or more gets lost.

There is also a difference between the available NavData-packages depending on whether the demo-mode is enabled or disabled:
With enabled demo-mode, just the packages *¨demo¨*, *¨vision_detect¨* and *¨chksum¨* are (by default) available 15 times a second; however, most of the time, this is enough.
With disabled demo-mode, (by default) all 28 packages are sent 200 times per second.

`State`-tags are also available. They are, like most packages, not very important right now, but are always sent with the NavDatas header.
The drone's state and its NavData are stored in the variable `State` or respectively `NavData`. The variable `NavDataCount` shows the count of the drone's sent NavData.

There are also the variables *NavDataTimeStamp* and *NavDataDecodingTime*, representing the time when the last NavData was received by the PS-Drone-API and how long it took to decode it. Make sure that decoding time is less then a time-unit.

```
##### Suggested clean drone startup sequence #####
import time, sys
import ps_drone            #Imports the PS-Drone-API

drone = ps_drone.Drone()   #Initials the PS-Drone-API
drone.startup()            #Connects to the drone and starts subprocesses

drone.reset()              #Sets drone's LEDs to green when red
while (drone.getBattery()[0]==-1):  time.sleep(0.1)   #Reset completed ?
print¨Battery:¨+str(drone.getBattery()[0])+¨%¨+str(drone.getBattery()[1])¨
drone.useDemoMode(True)        #15 basic datasets per second (default)
drone.getNDpackage([¨demo¨])   #Packets, which shall be decoded
time.sleep(0.5)                #Give it some time to fully awake

##### Mainprogram #####
NDC =   drone.NavDataCount
state = 0
end =   False
while not end:
    while drone.NavDataCount==NDC: time.sleep(0.001) #Wait for NavData
    if drone.getKey():
        state+=1
        if state==1:  drone.addNDpackage([¨demo¨]) #Add ¨demo¨ to list
        if state==2:  drone.getNDpackage([¨all¨])  #Get these packages
        if state==3:  drone.delNDpackage([¨all¨])  #Clear decoder-list
        if state==3:  drone.useDemoMode(False)     #switch to full-mode
        if state==4:  drone.addNDpackage([¨demo¨])
        if state==5:  drone.getNDpackage([¨all¨])
        if state>5:   sys.exit()

    print "-------------------"
    if state==0:  print¨##### Demo Mode: On   NavData-Packages: None¨
    if state==1:  print¨##### Demo Mode: On   NavData-Packages: Demo¨
    if state==2:  print¨##### Demo Mode: On   NavData-Packages: All¨
    if state==3:  print¨##### Demo Mode: Off  NavData-Packages: None¨
    if state==4:  print¨##### Demo Mode: Off  NavData-Packages: Demo¨
    if state==5:  print¨##### Demo Mode: Off  NavData-Packages: All¨

    if drone.NavDataCount-NDC>1:
        print¨Lost ¨+str(drone.NavDataCount-NDC-1)+¨ NavData¨
    print¨Number of package:¨+str(drone.NavDataCount)
    print¨Receifetime:       ¨+str(drone.NavDataTimeStamp)+¨, \t¨\
            +str(time.time()-drone.NavDataTimeStamp)+¨ sec ago¨
    print¨Time to decode:    ¨+str(drone.NavDataDecodingTime)
    print¨Included packages:¨+str(drone.NavData.keys())
    print¨State-data:        ¨+str(drone.State)
    try:    print¨Demo-data:         ¨+str(drone.NavData["demo"])
    except: pass
    NDC=drone.NavDataCount
```

Listing 2.4: Sourcecode of sample *firstNavData.py*

While a program-loop using NavData, it is suggested to wait for the next time-unit by using the NavDataCount as shown in the previous example.

*getNDpackage()* sets the exact list of packages which will be decoded.
With the commands *addNDpackage()* and *delNDpackage()* a couple of packages can be added to, or deleted from the decoding list.

You start, for example, with *getNDpackage([¨demo¨,¨time¨])* and do later *addNDpackage([¨altitude¨])* and *delNDpackage([¨time¨])*, the decoding would as if setting *getNDpackage([¨demo¨,¨altitude¨])*. There is no need to put the entries into a particular order.

Possible entries are the names of the packages, but also *¨all¨* for the complete set of packages:
*¨demo¨*, *¨time¨*, *¨pwm¨*, *¨raw_measures¨*, *¨phys_measures¨*, *¨gyros_offsets¨*, *¨altitude¨*, *¨magneto¨*, *¨euler_angles¨*, *¨references¨*, *¨rc_references¨*, *¨vision_detect¨*, *¨vision¨*, *¨vision_raw¨*, *¨vision_of¨*, *¨vision_perf¨*, *¨trackers_send¨*, *¨video_stream¨*, *¨hdvideo_stream¨*, *¨games¨*, *¨trims¨*, *¨pressure_raw¨*, *¨wind_speed¨*, *¨kalman_pressure¨*, *¨watchdog¨*, *¨wifi¨*, *¨adc_data_frame¨*, *¨zimmu_3000¨*, *¨chksum¨* and *¨all¨*.

Most of the packages and their included values are not very well documented; this tutorial will focus on the most important packages. For further and more comprehensive information, please take a look at chapter 4, starting on page 47, and at the sourcecode of *ps_drone.py*.

Note: Some measurements are not available when the drone is not in flight-mode. If it is on the ground, some values will stay "*0*".

This tutorial will show the most important values for the packages, like *State*, *¨demo¨*, *¨magneto¨*, *¨pressure_raw¨*, *¨altitude¨*, *¨kalman_pressure¨*, *¨wind_speed¨*, *¨pwm¨* and *¨vision_detect¨*.

*State*:

| No | Name | 0: | 1: |
|---|---|---|---|
| [10] | NavData demo | All NavData | NavData demo |
| [12] | Motors status | OK | Problem |
| [15] | Batterystatus | OK | Too low |
| [18] | Magnetometer calib | No calibration needed | Calibration needed |
| [20] | Wind mask | OK | Too much wind |
| [21] | Ultrasonic sensor | OK | Deaf |
| [31] | Emergency landing | No emergency | Emergency |

¨*demo*¨-package:
Shows the most important values at all, like some status-tags and information about slope, acceleration and battery charge.

| Pos | Note |
|---|---|
| [0][2] | Status-tag for ¨landed¨ - Drone is on the ground or not fully flying |
| [0][3] | Status-tag for ¨flying¨ - Drone is flying |
| [0][4] | Status-tag for ¨hovering¨ - Name should be: ¨landing¨, the drone is landing |
| [1] | Battery-status in percent |
| [2][0] | Pitch in degree (front/back) |
| [2][1] | Roll in degree (left/right) |
| [2][2] | Yaw in degree (turn) |
| [3] | Altitude in cm (only when flying) |
| [4][0] | Estimated speed in X in mm/s (forward/backward, only when flying) |
| [4][1] | Estimated speed in Y in mm/s (left/right, only when flying) |
| [4][2] | Estimated speed in Z in mm/s (up/down, only when flying) |

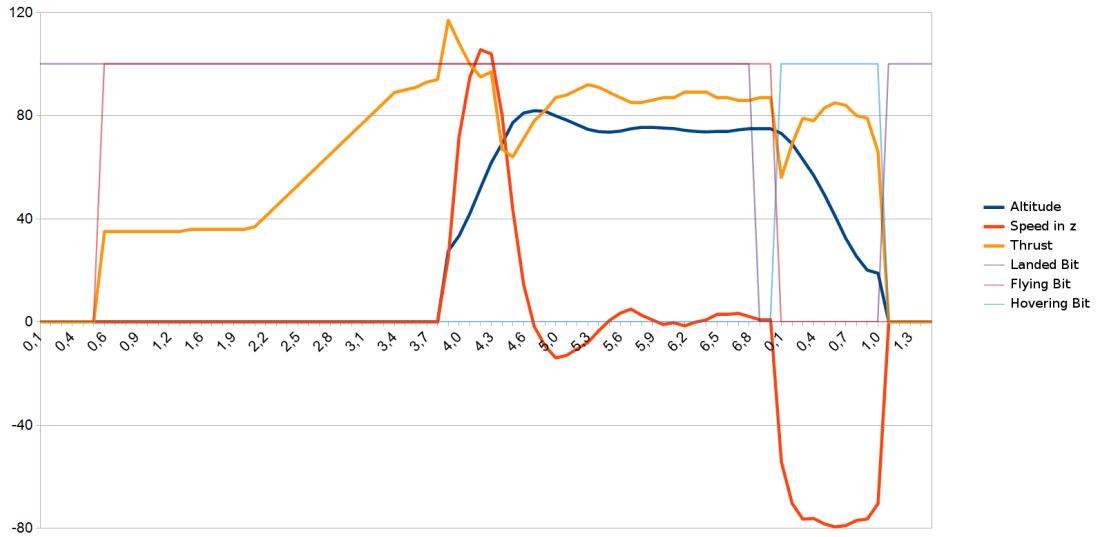



Figure 2.1: Status-tags while taking off and landing of an AR.Drone 2.0

¨*magneto*¨-package:

| Pos | Note |
|---|---|
| [0] | Magnetometer values for X, Y and Z |
| [1] | Raw magnetometer values for X, Y and Z |

¨*pressure_raw*¨-package:

| Pos | Note |
|---|---|
| [0] | A raw value, the height is calculable, the higher the drone flies the lower the value (about 80cm per count) |

¨*altitude*¨-package:

| Pos | Note |
|---|---|
| [3] | Altitude in mm |

¨*kalman_pressure*¨-package:

| Pos | Note |
|---|---|
| [1] | Estimated height (only when flying) |

¨*wind_speed*¨-package:

| Pos | Note |
|---|---|
| [0] | Windspeed (only when flying) |
| [1] | Angle of wind (only when flying) |

¨*pwm*¨-package:
The motors are controlled by pulse-wide modulation. The higher the value, the higher is the rotation velocity of the propellers.

| Pos | Note |
|---|---|
| [0][0] | Shows the actual pwm-value for the motor on the front left |
| [0][1] | Shows the actual pwm-value for the motor on the front right |
| [0][2] | Shows the actual pwm-value for the motor on the back right |
| [0][3] | Shows the actual pwm-value for the motor on the back left |

¨*vision_detect*¨-package:
Gives information on up to four detected markers. In order to detect markers, some configurations needs to be done first.

| Pos | Note |
|---|---|
| [0] | Number of detected marker (max 4) |
| [1] | List of the types of detected markers |
| [2] | List of the X-positions of detected markers |
| [3] | List of the Y-positions of detected markers |
| [4] | List of the dimensional width of detected markers |
| [5] | List of the dimensional depth of detected markers |
| [6] | List of the distances of detected markers |
| [7] | List of the orientation angles of detected markers |
| [9] | List of the rotations of detected markers |

¨*chksum*¨-package:

| Pos | Note |
|---|---|
| [0] | Checksum |
| [1] | Quality: incorrect/correct |

```
##### Suggested clean drone startup sequence #####
import time, sys
import ps_drone              #Imports the PS-Drone-API

drone = ps_drone.Drone()     #Initials the PS-Drone-API
drone.startup()              #Connects to the drone and starts subprocesses

drone.reset()                #Sets drone's LEDs to green when red
while (drone.getBattery()[0]==-1):  time.sleep(0.1)   #Reset completed ?
print¨Battery:¨+str(drone.getBattery()[0])+¨%¨+str(drone.getBattery()[1])¨
drone.useDemoMode(False)    #Give me everything...fast
drone.getNDpackage([¨demo¨,¨pressure_raw¨,¨altitude¨,¨magneto¨,¨wifi¨])
time.sleep(0.5)              #Give it some time to fully awake

##### Mainprogram #####
NDC =   drone.NavDataCount
end =   False
while not end:
    while drone.NavDataCount==NDC: time.sleep(0.001) #Wait for NavData
    if drone.getKey():             end=True
    NDC=drone.NavDataCount

    print¨-----------¨
    print¨Aptitude[X,Y,Z]:           ¨+str(drone.NavData[¨demo¨][2])
    print¨Altitude/sensor/pressure:¨+str(drone.NavData[¨altitude¨][3])\
                                    +¨/¨+str(drone.State[21])+¨/¨\
                                    +str(drone.NavData[¨pressure_raw¨][0])
    print¨Megnetometer[X,Y,Z]:       ¨+str(drone.NavData[¨magneto¨][0])
    print¨Wifi link quality:         ¨+str(drone.NavData[¨wifi¨])
```

Listing 2.5: Sourcecode of sample *getNavData.py*

## 2.5 Configurate the drone

Before working with video, it is necessary to understand some basic configuration issues. Depending on configuration-settings the drone behaves differently, so is e.g. the command *useDemoMode()*, beside others, an alias for a reconfiguration of the drone.

The AR.Drone 2.0 has 104 information and configurable options. Some entries give a status and are only readable, like *¨general:flight_time¨*, some are editable, e.g. *¨general:ardrone_name¨*. These simple text strings, more or less, consist of two parts; the first part is the option's name, containing the category and the option, the second part is the set value.

The PS-Drone supports you by organizing the complex procedures of communication and its handling, so a reconfiguration is not so tricky. You cannot "brick" the drone, but it is always better to know what you are doing.
It is important to know that reconfigurations are set with a delay, depending on the quantity of configurations to set at once. Depending on the drone's firmware (e.g. version 2.3.3) that procedure is optimized by temporarily sending extra NavData in between.
Furthermore, it happens once in a while that the drone's reactions are inconsistent after an alteration: The drone claims that the value has been set, but it maintains its previous

setting. It seems that this concerns mainly video: Sometimes a switch to ground-camera has been acknowledged, but the front-view is still visible.

Here is an example how to get and set the drones configurations:

```
##### Suggested clean drone startup sequence #####
import time, sys
import ps_drone              #Imports the PS-Drone-API

drone = ps_drone.Drone()     #Initials the PS-Drone-API
drone.startup()              #Connects to the drone and starts subprocesses

drone.reset()                #Sets drone's LEDs to green when red
while (drone.getBattery()[0]==-1):  time.sleep(0.1)   #Reset completed ?
print¨Battery:¨+str(drone.getBattery()[0])+¨%¨+str(drone.getBattery()[1])¨
drone.useDemoMode(True)     #15 basic datasets per second (default)
time.sleep(0.5)              #Give it some time to fully awake

##### Mainprogram #####
#Just list the configuration, it is already synchronized
for i in drone.ConfigData:     print i

#Take a closer look at an option...
print¨\nSample:¨,
for i in drone.ConfigData:
    if i[0]==¨control:altitude_max¨:
        print str(i)+¨Count:¨+str(drone.ConfigDataCount)\
                    +¨Timestamp:¨+str(drone.ConfigDataTimeStamp)

#...  and change it
print¨-----¨
print¨Setting \¨control:altitude_max\¨ to \¨2999\¨...¨
CDC =     drone.ConfigDataCount
NDC =     drone.NavDataCount
refTime = time.time()
drone.setConfig(¨control:altitude_max¨,¨2999¨)       #Change of an option
while CDC==drone.ConfigDataCount: time.sleep(0.001) #Wait until it is done
print¨ Finished after ¨+str(time.time()-refTime)+¨seconds, ¨\
          +str(drone.NavDataCount-NDC)+¨NavData where received meanwhile.¨
for i in drone.ConfigData:
    if i[0]==¨control:altitude_max¨:
        print str(i)+¨Count:¨+str(drone.ConfigDataCount)\
                    +¨Timestamp:¨+str(drone.ConfigDataTimeStamp)

#Change an option a couple of times
print¨\n-----¨
print¨Setting \¨control:altitude_max\¨ to \¨2996\¨,then to \¨2997\¨,\
                    \¨2998\¨,\¨2999\¨ and \¨3000\¨...¨
CDC =     drone.ConfigDataCount
NDC =     drone.NavDataCount
refTime = time.time()
drone.setConfig(¨control:altitude_max¨,¨2996¨) #Request change of an option
drone.setConfig(¨control:altitude_max¨,¨2997¨) #  request change again
drone.setConfig(¨control:altitude_max¨,¨2998¨) #  and again.
drone.setConfig(¨control:altitude_max¨,¨2999¨) #PS-Drone detects and deletes
drone.setConfig(¨control:altitude_max¨,¨3000¨) #  doubles and sets only last
while CDC==drone.ConfigDataCount: time.sleep(0.001) #Wait until it is done
print¨ Finished after ¨+str(time.time()-refTime)+¨seconds, ¨\
          +str(drone.NavDataCount-NDC)+¨NavData where received meanwhile.¨
```

```
for i in drone.ConfigData:
    if i[0]==¨control:altitude_max¨:
        print str(i)+¨Count:¨+str(drone.ConfigDataCount)\
                    +¨Timestamp:¨+str(drone.ConfigDataTimeStamp)

print¨\n-----¨
print¨Setting \¨control:altitude_max\¨ to \¨2980\¨,\
              \¨control:altitude_min\¨ to \¨499\¨ and \
              \¨video:video_on_usb\¨ to \¨false\¨...¨
CDC =     drone.ConfigDataCount
NDC =     drone.NavDataCount
refTime = time.time()
drone.setConfig(¨control:altitude_max¨,¨2980¨)#Change of an option
drone.setConfig(¨control:altitude_min¨,¨499¨)  #Change of an other option
drone.setConfig(¨video:video_on_usb¨,¨false¨)  #Change of an other option
while CDC==drone.ConfigDataCount: time.sleep(0.001) #Wait until it is done
print¨ Finished after ¨+str(time.time()-refTime)+¨seconds,¨\
          +str(drone.NavDataCount-NDC)+¨NavData where received meanwhile.¨
for i in drone.ConfigData:
    if i[0]==¨control:altitude_max¨ or i[0]==¨control:altitude_min¨\
                                    or i[0]==¨video:video_on_usb¨:
        print str(i)+¨Count:¨+str(drone.ConfigDataCount)\
                    +¨Timestamp:¨+str(drone.ConfigDataTimeStamp)

print¨\n-----¨
print¨Just resyncing the drones configuration...¨
CDC =     drone.ConfigDataCount
NDC =     drone.NavDataCount
refTime = time.time()
drone.getConfig()
while CDC==drone.ConfigDataCount: time.sleep(0.001) #Wait until it is done
print¨ Finished after ¨+str(time.time()-refTime)+¨seconds,¨\
          +str(drone.NavDataCount-NDC)+¨NavData where received meanwhile.¨
```

Listing 2.6: Sourcecode of sample *firstConfig.py*

For reconfiguration you mainly need two commands: *getConfig()* and *setConfig()*

*getConfig()* will synchronize the local copy of the drone's configuration and stores the list in the variable *ConfigData*, along with the time when it was received in *ConfigDataTimeStamp* and the count of received configurations in *ConfigDataCount*. The fetch of the drone's configuration works not in real-time and may take up to 0.15 seconds.

The command *setConfig(¨option¨,¨value¨)* sets the value for the given option. The name of the option has to be a text-string. As alterations are not in real-time, in order to optimize the configuration-speed and for convenience, the request is added to a queue and sent one after another without you having to care about. PS-Drone tests the queue for double entries of the same option and initiates the latest change to prevent unnecessary delay.

If you try to set an unknown option, the drone will reject the request; however, if the value is unknown, the drone sets an unpredictable value itself. When the queue is done, the local copy of the drone's configuration is automatically updated analogue to command *getConfig()*.

Not all of the options and their possible values are guessable, some are not even documented by Parrot. Take a look at chapter 5, starting on page 57 to find the known descriptions of the entries.

A mode for multi-configuration is also existing. It is mainly interesting for using video and it contains the configurations for the set sessions. It is not really necessary to care about how it works; however, once you have started using it, you must continue to use it until your program has finished or until you have switched back to single-configuration.

```
##### Suggested clean drone startup sequence #####
import time, sys
import ps_drone              #Imports the PS-Drone-API

drone = ps_drone.Drone()     #Initials the PS-Drone-API
drone.startup()              #Connects to the drone and starts subprocesses

drone.reset()                #Sets drone's LEDs to green when red
while (drone.getBattery()[0]==-1):  time.sleep(0.1)   #Reset completed ?
print¨Battery:¨+str(drone.getBattery()[0])+¨%¨+str(drone.getBattery()[1])¨
drone.useDemoMode(True)     #15 basic datasets per second (default)
time.sleep(0.5)              #Give it some time to fully awake

##### Mainprogram #####
print¨default IDs:¨,
for i in drone.ConfigData:
    if i[0].count(¨control:¨)==1 and i[0].count(¨_id¨)==1:  print str(i)

print¨set default multiconfiguration:
drone.setConfigAllID()¨
CDC = drone.ConfigDataCount
while CDC==drone.ConfigDataCount: time.sleep(0.001) #Wait until it is done
for i in drone.ConfigData:
    if i[0].count(¨control:¨)==1 and i[0].count(¨_id¨)==1:  print str(i)

print¨Setting \¨control:altitude_max\¨ to \¨2999\¨...¨
drone.setMConfig(¨control:altitude_max¨,¨2999¨)      #Change of an option
while CDC==drone.ConfigDataCount: time.sleep(0.001) #Wait until it is done
for i in drone.ConfigData:
    if i[0]==¨control:altitude_max¨:  print str(i)
```

Listing 2.7: Sourcecode of sample *multiConfig.py*

As you see, multi-configuration does not really differ from normal configuration.

After having enabled multi-configuration by sending pre-set IDs with the command *setConfigAllID()*, you change configurations by using the command *setMConfig()* which has the same usage and behaviour as *setConfig()*.

## 2.6 Detecting markers

The AR.Drone 2.0 supports detecting certain machine-parsable images using its cameras. These particular images are called “tags” or “markers”. It is an integrated feature of the drone able to detect up to four markers; their position, distance to the drone and, depending on the marker, their orientation at the same time.
The cameras of the drone (tested with a standard AR.Drone 2.0) are not really good, so there is a limited range of luminance in which it is possible to get acceptable results. Also spotlights and reflections of marker-printouts are a serious problem.

The marker that comes with the drone is the “black and white oriented roundel”. It is recommended for the following tests, not least because the drone detects it most certainly. The tag “Roundel” has also been tested, but, because it contains specific colors, it is harder for the drone to detect it.

```
##### Suggested clean drone startup sequence #####
import time, sys
import ps_drone              #Imports the PS-Drone-API

drone = ps_drone.Drone()     #Initials the PS-Drone-API
drone.startup()              #Connects to the drone and starts subprocesses

drone.reset()                #Sets drone's LEDs to green when red
while (drone.getBattery()[0]==-1):  time.sleep(0.1)   #Reset completed ?
print¨Battery:¨+str(drone.getBattery()[0])+¨%¨+str(drone.getBattery()[1])¨
drone.useDemoMode(True)                         #15 basic datasets per second
drone.getNDpackage([¨demo¨,¨vision_detect¨])  #Packets to decoded
time.sleep(0.5)                                 #Gives time to fully awake

##### Mainprogram #####
#Setting up detection...
drone.setConfig(¨detect:detect_type¨,¨3¨)           #Universal detection
drone.setConfig(¨detect:detections_select_h¨,¨128¨) #Front:Oriented Roundel
drone.setConfig(¨detect:detections_select_v¨,¨0¨)   #Ground:None
CDC = drone.ConfigDataCount
while CDC==drone.ConfigDataCount: time.sleep(0.001) #Wait until it is done

#Get detections
end =   False
while not end:
    while drone.NavDataCount==NDC: time.sleep(0.001) #Wait for NavData
    if drone.getKey():              end=True
    tagNum=drone.NavData[¨vision_detect¨][0]   #No of found tags
    tagX  =drone.NavData[¨vision_detect¨][2]   #Horizontal position(s)
    tagY  =drone.NavData[¨vision_detect¨][3]   #Vertical position(s)
    tagZ  =drone.NavData[¨vision_detect¨][6]   #Distance(s)
    tagRot=drone.NavData[¨vision_detect¨][7]   #Orientation(s)

#Show detections
    if tagNum:
        for i in range (0,tagNum):
            print ¨Tag no¨+str(i)+¨:  X=¨+str(tagX[i])+¨ Y=¨+str(tagY[i])\
                  +¨ Dist=¨+str(tagZ[i])+¨ Orientation=¨+str(tagRot[i])
    else:   print "No tag detected"
```

Listing 2.8: Sourcecode of sample *firstTagDetection.py*

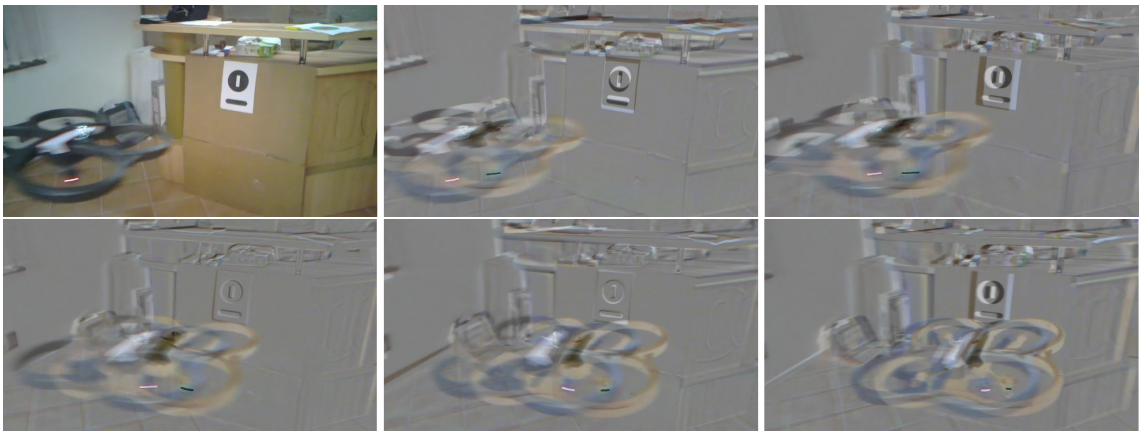
At first, the detection has to be enabled by reconfiguring option *¨detect:detect_type¨* from *¨0¨* to *¨10¨*.
Options *¨detect:detections_select_h¨* and *¨detect:detections_select_v¨* limit the detection to a specific set of available tags for the front or respectively the ground camera. Each available tag has a certain number, e.g. "Black and white oriented roundel" has *¨128¨* and "Roundel" has *¨2¨*. To detect both, set the option's value to *¨130¨*; however, the more tags are selected, the longer the drone might need to detect and analyze the positions of the markers.

The detected coordinates for X (horizontal) and Y (vertical) are given in a range from 0 to 1000, which means a low value on the left respectively on top and a high value on the right respectively bottom. Also the distance has a value range from 0 to 1000, where 0 means very close and 1000 fare away.

## 2.7 Using video

For using the drones video, the package "OpenCV2" is used for decoding, preparing and analyzing the images. Using this API to work with the drone's video-function, it is pretty simple to make yourself familiar with. However, it is still helpful to have some theoretical background knowledge:

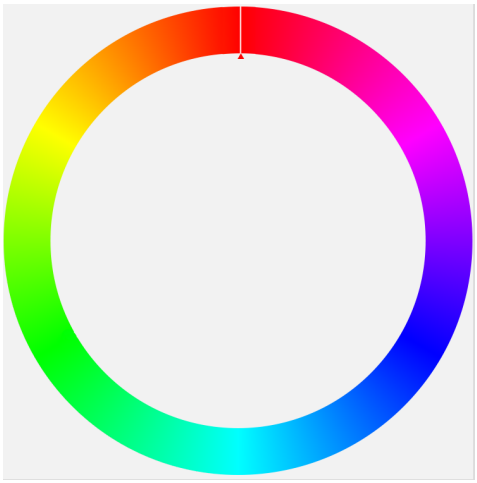
The video-stream of Parrots AR.Drone 2.0 is H.264 or MPEG4 coded (to be exact: MPEG4.10 respectively MPEG4.2).
Both codecs work with "I-frames", also known as "key-frames", which store the whole information of an image like a photo or picture, and "P-Frames", which store just the difference to the preceding frame. I-frames are followed by a certain amount of P-frames. A P-frame always refers to the preceding frame (I-frame or P-frame) and stores only the information that differs from this preceding frame. This might be useful to know as decoding the videostream is pretty complex and needs considerable CPU-usage. For slow(er) computers for which it is impossible to decode every frame in real-time, it is for example not possible to decode every second frame, as P-frames refer to the preceding frame. The result would be a loss of information causing an incorrect and distorted image.



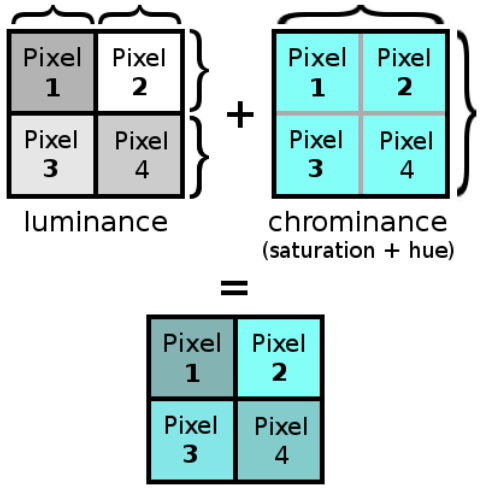
Stylized depiction of a video sequence showing one I-frame, followed by five p-frames

So, decode all or nothing ? Not really, because PS-Drone offers, beside lowering the video-bitrate, the opportunity just to decode the key-frames which are received in a strict period, for up to 95% less CPU-usage.

If you like to examine footage of colored objects, it is also helpful to know that both codecs store their information as progressive 4:2:0 YUV. That means, that every picture is stored as a whole (no two interlaced half pictures) and pixels are represented by luminance, the color's saturation and the color's hue. The color slides from red (0) to green (85) and blue (170) back to (allmost) red (255).
In a square of $2 \times 2$ pixels, every pixel has its own value for brightness, but all four pixels share the same value for color. That is due to the fact, that the human eye cannot notice colour related differences as good as differences in luminance. Also different shades of blue are not as good distinguishable as different shades of green or red, that is why blue is also stripped down by the compression codecs. These are usual techniques to reduce a video's data stream and not a special limitation of the drone or PS-Drone.



Color wheel




Depiction of four 4:2:0 YUV coded pixels. Each pixel has a value for luminance; four pixels share the same value for chrominance.

The H.264-codec is a pretty recent method to reduce and compact a video-stream. The drone sends the pictures for both cameras in resolutions of $1280 \times 720$ and $640 \times 360$ pixels. The MPEG4-codec is older than H.264; it needs more bandwidth, but also less CPU-power to decode and is suggested for slow computers. The drone's resolution for MPEG4-streams is $640 \times 368$ pixels, but the additional 8 lines contain no usable information. The pictures might be sharper, but there is a tendency for "blocking", if the bit-rate is too low for all the information of the videostream.

The drone's default video-settings are a H.264 coded, $640 \times 360$ pixel resolved videostream of the front camera. It is recommended to set the video codec and the resolution before starting the video function and not to change it while it is running.

```
##### Suggested clean drone startup sequence #####
import time, sys
import ps_drone              #Imports the PS-Drone-API

drone = ps_drone.Drone()     #Initials the PS-Drone-API
drone.startup()              #Connects to the drone and starts subprocesses

drone.reset()                #Sets drone's LEDs to green when red
while (drone.getBattery()[0]==-1):  time.sleep(0.1)   #Reset completed ?
print¨Battery:¨+str(drone.getBattery()[0])+¨%¨+str(drone.getBattery()[1])¨
drone.useDemoMode(True)                          #15 basic datasets per second
drone.getNDpackage([¨demo¨,¨vision_detect¨])  #Packets to decoded
time.sleep(0.5)                                  #Gives time to fully awake

##### Mainprogram #####
CDC = drone.ConfigDataCount
drone.setConfigAllID()           #Go to multiconfiguration-mode
drone.sdVideo()                  #Choose lower resolution
drone.frontCam()                 #Choose front view
while CDC==drone.ConfigDataCount: time.sleep(0.001) #Wait until it is done
drone.startVideo()               #Start video-function
drone.showVideo()                #Display the video

print¨<space> to toggle front- and groundcamera, any other key to stop¨
IMC =    drone.VideoImageCount   #Number of encoded videoframes
stop =   False
ground = False                   #To toggle front- and groundcamera
    while drone.VideoImageCount==IMC: time.sleep(0.01) #Wait for next image
    IMC = drone.VideoImageCount #Number of encoded videoframes
    key = drone.getKey()
    if key==¨ ¨:
        if ground:          ground = False
        else:               ground = True
        drone.groundVideo(ground) #Toggle between front- and groundcamera.
    elif key and key!=¨ ¨: stop=True
```

Listing 2.9: Sourcecode of sample *firstVideo.py*

The first step to enable video is to switch the drone to the multi-configuration mode, using *setConfigAllID()* as mentioned before. Remember to use the command *setMConfig()* for changing the drone's configurations.
After the video-resolution and the video-codec have been set, using the commands *sdVideo()/hdVideo()* and *mp4Video()*, give the drone some time to adjust these settings (you can either just wait or check for changes of *ConfigDataCount*). It will take one to three seconds until the video is fully operable.
Again, it is highly recommended not to alter video-resolution and -codec after the video function has been started. Nevertheless, it is always possible to set or alter any other command any time.

As mentioned before, the AR.Drone 2.0 might have a bug when it comes to setting up the video function. Sometimes the request for a change is set and confirmed by the drone (e.g. switching from front to bottom camera), but nothing happens. Unfortunately, there is no workaround for this.

The usage of the video-images is similar to NavData and ConfigData: Every single decoded video picture is stored as an openCV2-image-object in the variable `VideoData`, its timestamp is stored in `VideoImageTimeStamp` and the time it took to decode in `VideoDecodeTime`.
The variable `VideoReady` shows the status of the video-related processes.

The commands `frontCam()` and `groundCam()` allow to change the streamed camera; the commands `showVideo()` and `hideVideo()` open and hide a window which shows the drone's actual view.

If the used computer is too slow to decode the drones videostream in real-time, you can switch to more economic modes by using the commands `slowVideo()` and `midVideo()`. Both commands determine that only keyframes are decoded, which also means that the frame-rate is decreasing; depending on resolution, used camera and codec, it is between 1 to 2 frames per second.

Due to internal buffers, `slowVideo()` has a higher delay, `midVideo()` needs more time to process, but has the same delay as `fullVideo()`. Using `slowVideo()` reduces the CPU usage for decoding by 92% - 96%, using `midVideo()` by 33% - 82%.

Please note: it is possible that the video initialization can fail for various reasons. It tries to detect and recover the initialization, but it is suggested to start the video-function before the take off.

That is all! For further information and all commands, please see chapter 3, beginning on page 24.

# 3 PS-Drone-API commands

## 3.1 Startup

### 3.1.1 Startup settings

***DroneIP***
IP-address of the drone as a string. Manually editable. (Default: *¨192.168.1.1¨*)

***NavDataPort***
Port-number through which the drone sends the NavData-stream, as an integer. Manually editable. (Default: *5554*)

***VideoPort***
Port-number through which the drone sends the video-stream, as an integer. Manually editable. (Default: *5555*)

***CMDPort***
Port through which the drone receives commands, as an integer. Manually editable. (Default: *5556*)

***CTLPort***
Port-number through which the drone sends its configuration, as an integer. Manually editable. (Default: *5559*)

### 3.1.2 *startup()*

Connect to the drone.

**Usage:** startup()

**Return:** None

**Note:** After setting drone's IP and Ports, use this command next.

**Example:** *drone.startup()*

## 3.2 Calibration

### 3.2.1 Calibration variables

***selfRotation***
Contains the self-spinning correction value yaw gyrometer in degrees per second, measured by *getTolerances()* or manually editable. (Default: *0.0185*)

### 3.2.2 *trim()*

Drone sets the reference to the horizontal plane.

**Usage:** trim()

**Return:** None

**Note:** Drone has to be on the ground.

**Example:** `drone.trim()`

### 3.2.3 *mtrim()*

Drone calibrates magnetometer.

**Usage:** mtrim()

**Return:** None

**Note:** Drone has to fly to rotate one time.

**Example:** `drone.mtrim()`

### 3.2.4 *mantrim()*

Manual calibration of drone's gyrometers.

**Usage:** mantrim(theta,phi,yaw)

**Return:** None

| Name: | Type: | Description: |
|---|---|---|
| theta | float | Sets the correction theta-angle (sideward) |
| phi | float | Sets the correction phi-angle (forward/backward) |
| yaw | float | Sets the correction yaw-angle (horizontal turn) |

**Note:** It is a function of the drone, but not officially described or supported.

**Example:** `drone.mantrim(0.321,0.2,0.9113)`

### 3.2.5 *getSelfRotation()*

Drone measures out the yaw gyrometers self spinning.

**Usage:** getSelfRotation(time)

**Return:** Yaw gyro self-spinning correction value in degrees per second.

| Name: | Type: | Description: |
|---|---|---|
| time | integer | Time to measure |

**Note:** Drone has to be on the ground, useful for `turnAngle()` or highly accurate measures. The value is stored in `selfRotation`.

**Example:** `drone.getSelfRotation(10)`

## 3.3 Movement

### 3.3.1 Movement variables

***speed***
The drone's default movement speed, editable by `setSpeed()`. (Default: `0.2`).

### 3.3.2 *takeoff()*

Drone takes off and enters flight mode.

**Usage:** takeoff()

**Return:** None

**Note:** Necessary before any movement command.

**Example:** `drone.takeoff()`

### 3.3.3 *land()*

Drone lands and leaves flight mode.

**Usage:** land()

**Return:** None

**Note:** -

**Example:** `drone.land()`

### 3.3.4 *setSpeed()*

Sets the optional speed of drone's movements.

**Usage:** setSpeed(speed)

**Return:** speed

| Name: | Type: | Description: |
|---|---|---|
| speed | float | Speed value from *0.0* to *1.0*; from not active to full thrust. |

**Note:** Good for basic movement commands.

**Example:** `drone.setSpeed(0.5)`

### 3.3.5 *stop()*

Drone stops all movements and holds position.

**Usage:** stop()

**Return:** None

**Note:** Use this command to stop. Setting the speed of movement to *0.0* makes the drone glide into the same direction as before and not hold its position.

**Example:** `drone.stop()`

### 3.3.6 *Basic movement*

Drone moves or turns to given direction, until it gets the command to change direction.

**Usage:** moveLeft(optional)
moveRight(optional)
moveForward(optional)
moveBackward(optional)
moveUp(optional)
moveDown(optional)
turnLeft(optional)
turnRight(optional)

**Return:** None

| Name: | Type: | Description: |
|---|---|---|
| optional | float | Speed value from *0.0* to *1.0*; from no active movement to full thrust. If empty, the value from *speed* will be used. |

**Note:** Very basic movements for first tests, with only one movement at a time. Setting all values to *0.0* does not make the drone stop, but glide in the same (horizontal) direction as before. To stop movement, use *stop()* instead.

**Example:** *drone.moveLeft(0.4)*

### 3.3.7 *move()*

Drone moves to all given directions in given speed.

**Usage:** move(right,forward,up,turn right)

**Return:** None

| Name: | Type: | Description: |
|---|---|---|
| right | float | Speed value from *-1.0* to *1.0* from full speed left to full speed right movement, *0.0* for passive. |
| forward | float | Speed value from *-1.0* to *1.0* from full speed backward to full speed forward movement, *0.0* for passive. |
| up | float | Speed value from *-1.0* to *1.0* from full speed descent to full speed ascent movement, *0.0* for passive. |
| turn right | float | Speed value from *-1.0* to *1.0* from full speed left to full speed right rotation, *0.0* for passive. |

**Note:** Setting all speed values to *0.0* would not stop movement; use *stop()* instead.

**Example:** *drone.move(0.0,0.8,-0.1,0)*

### 3.3.8 *relMove()*

Drone moves, relative to start position and angle, to all given directions in given speed.

**Usage:** relMove(right,forward,up,turn right)

**Return:** None

| Name: | Type: | Description: |
|---|---|---|
| right | float | Speed value from *-1.0* to *1.0* from full speed left to full speed right movement relative to start position, *0.0* for passive. |
| forward | float | Speed value from *-1.0* to *1.0* from full speed backward to full speed forward movement relative to start position, *0.0* for passive. |
| up | float | Speed value from *-1.0* to *1.0* from full speed descent to full speed ascent movement, *0.0* for passive. |
| turn right | float | Speed value from *-1.0* to *1.0* from full speed left to full speed right rotation, *0.0* for passive. |

**Note:** Use `mtrim()` before using `relMove()`. Setting all speed values to `0.0` would not stop movement; use `stop()` instead.

**Example:** `drone.relMove(0.0,0.8,-0.1,0)`

### 3.3.9 *turnAngle()*

Drone turns exactly by a given angle.

**Usage:** turnAngle(degree,maxspeed,optional)

**Return:** success

| Name: | Type: | Description: |
|---|---|---|
| turnangle | float | degree to turn. negative for left, positive for right rotation. |
| maxspeed | float | Speed value from *0.0* to *1.0* from passive to full speed.<br>If value is *0.0* (passive), value from `speed` will be used instead. |
| optional | float | Accurateness for rotating to given target-angle.<br>Default for demo-mode is ±0.1° for full-mode ±0.005°. |
| sucess | bool | `True`, when finished the turn. |

**Note:** Less accurateness means a faster completion of the turn. A self-running gyrometer interference factor, measured by `getTolerances()`, is included for exact turns. If this is not wished, set `selfRotation=0`.
Known bug: this command is not able to perform an exact turn of 180.0°; instead, the drone tries to find its destination angle until the battery is dead and without the possibility to stop the drone neatly.

**Example:** `drone.turnAngle(-23.811,1)`

## 3.4 NavData

### 3.4.1 NavData variables

***NavData***

Contains the NavData-values as python dictionary. More details in section 4 (starting on page 47) and the source-code of *ps_drone.py*. Not editable.

***State***

Contains the drone's states as a list. More details in section 4.1 (page 48) and the source-code of *ps_drone.py*. Not editable.

***NavDataCount***

The Sequential number of the NavData package. Not editable.

***NavDataTimeStamp***

Time when the NavData package, stored in `NavData`, was decoded. Not editable.

***NavDataDecodingTime***

Time needed to decode the NavData package, stored in `NavData`. Not editable.

***NoNavData***

Success-tag of decoding the drone's NavDatas. Will be `False` if a failure occurs; values of `Navdata` will be not actual. Not editable.

### 3.4.2 *useDemoMode()*

Switch between the NavData demo- and full-mode.

**Usage:** useDemoMode(optional)

**Return:** None

| Name: | Type: | Description: |
|---|---|---|
| optional | boolean | `True`: Use demo-mode or not set. `False`: Use full-mode. |

**Note:** The NavData are sent 200 times per second in full mode and contain, by default, all data. In demo mode, 15 NavData are sent per second and contain, by default, a subset of data.

**Example:** `drone.useDemoMode(True)`

### 3.4.3 *useMDemoMode()*

Switch between the NavData demo and full mode in a multi-configuration environment.

**Usage:** useMDemoMode(optional)

**Return:** None

| Name: | Type: | Description: |
|---|---|---|
| optional | boolean | `True`: Use demo-mode or not set. `False`: Use full-mode. |

**Note:** The NavData are sent 200 times per second in full mode and contain, by default, all data. In demo mode, 15 NavData are sent per second and contain, by default, a subset of data.

**Example:** `drone.useMDemoMode(True)`

### 3.4.4 *getNDpackage()*

Sets the packages of the NavData sensor measurements to decode.

**Usage:** getNDpackage([packages])

**Return:** None

| Name: | Type: | Description: |
|---|---|---|
| packages | list | A list of packages to decode. See name list in 3.1 below. |

**Note:** Use item *"all"* to get all data.

**Example:** *drone.getNDpackage(["demo","pwm","altitude","chksum"])*

### 3.4.5 *addNDpackage()*

Adds packages of the NavData sensor measurements to decode.

**Usage:** addNDpackage([packages])

**Return:** None

| Name: | Type: | Description: |
|---|---|---|
| packages | list | A list of packages to decode. See name list in 3.1 below. |

**Note:** Use item *"all"* to add all data.

**Example:** *drone.addNDpackage(["demo","pwm","altitude","chksum"])*

### 3.4.6 *delNDpackage()*

Ignores packages of the NavData sensor measurements to decode.

**Usage:** delNDpackage([packages])

**Return:** None

| Name: | Type: | Description: |
|---|---|---|
| packages | list | A list of packages to decode. See name list in 3.1 below. |

**Note:** Use item *"all"* to ignore all data.

**Example:** *drone.delNDpackage(["demo","pwm","altitude","chksum"])*

```
"demo", "time", "raw_measures", "phys_measures", "gyros_offsets",
"euler_angles", "references", "trims", "rc_references", "altitude",
"vision_raw", "vision_of", "vision", "vision_perf", "vision_detect",
"games", "trackers_send", "watchdog", "adc_data_frame", "wind_speed",
"pwm", "video_stream", "pressure_raw", "magneto", "kalman_pressure",
"wifi", "hdvideo_stream", "zimmu_3000", "chksum", "all"
```

Listing 3.1: List of available NavData-package-settings

### 3.4.7 *reconnectNavData()*

Reinitializes the NavData communication after a signal loss.

**Usage:** reconnectNavData()

**Return:** None

**Note:** -

**Example:** `drone.reconnectNavData()`

## 3.5 Configuration

### 3.5.1 Configuration variables

***ConfigData***
Contains the configuration of the drone as a list. Will be updated after a set of configuration alterations have been transmitted; it will be also updated after the command `getConfig()`. More details in section 5 (page 57). Not editable.

***ConfigDataCount***
Sequential number of received copies of the drone's configuration which are stored in `ConfigData`. Not editable.

***ConfigDataTimeStamp***
Time when the configuration, stored in `ConfigData`, was received. Not editable.

***ConfigSending***
Shows the status of setting configuration. It is `True` when in process. Not editable.

***ConfigSessionID***
Contains the drone's session ID for multi-configuration-mode as a string.
Editable by `setConfigSessionID()`. (Default: `¨03016321¨`)

***ConfigUserID***
Contains the drone's user ID for multi-configuration-mode as a string.
Editable by `setConfigUserID()`. (Default: `¨0a100407¨`)

***ConfigApplicationID***
Contains the drone's application ID for multi-configuration-mode as a string.
Editable by `setConfigApplicationID()`. (Default: `¨03016321¨`)

***sendConfigSaveMode***
Indicates that the drone's reconfiguration settings will be done in a slower save-mode when `True`. This variable reacts like a command with max 0.1 second delay when changed. Manually editable. (Default: False)

For detailed information about configuration-options , see section 5.1 starting on page 57.

### 3.5.2 *getConfig()*

Requests drone for a copy of its actual configuration.

**Usage:** getConfig()

**Return:** None

**Note:** No update in real time.

**Example:** `drone.getConfig()`

### 3.5.3 *setConfig()*

Changes drone's configuration.

**Usage:** setConfig(parameter,value)

**Return:** None

| Name: | Type: | Description: |
|---|---|---|
| parameter | string | Changes value of parameter as displayed in *ConfigData*. |
| value | string | Value to set |

**Note:** Settings are not in realtime. *ConfigData* will be updated automatically after all undone settings have been processed.

**Example:** `drone.setConfig(¨control:altitude_max¨,¨5000¨)`

### 3.5.4 *setConfigSessionID()*

Sets the session ID for multi-configuration-mode, stores it in *ConfigSessionID* and sends it to the drone.

**Usage:** setConfigSessionID(id)

**Return:** id

| Name: | Type: | Description: |
|---|---|---|
| id | string | A string of eight hexadecimal digits. |

**Note:** Set it before using multi-configuration-mode. Setting is not in real time.

**Example:** `drone.setConfigSessionID(¨03016321¨)`

### 3.5.5 *setConfigUserID()*

Sets the session ID for multi-configuration-mode, stores it in *ConfigUserID* and sends it to the drone.

**Usage:** setConfigUserID(id)

**Return:** id

| Name: | Type: | Description: |
|---|---|---|
| id | string | A string of eight hexadecimal digits. |

**Note:** Set it before using multi-configuration-mode. Setting is not in real time.

**Example:** `drone.setConfigUserID(¨0a100407¨)`

### 3.5.6 *setConfigApplicationID()*

Sets the session ID for multi-configuration-mode, stores it in *ConfigApplicationID* and sends it to the drone.

**Usage:** setConfigApplicationID(id)

**Return:** id

| Name: | Type: | Description: |
|---|---|---|
| id | string | A string of eight hexadecimal digits. |

**Note:** Set it before using multi-configuration-mode. Setting is not in real time.

**Example:** *drone.setConfigApplicationID(¨03016321¨)*

### 3.5.7 *setConfigAllID()*

Sets the session, user and application IDs for multi-configuration-mode of the drone, stored in *ConfigSessionID*, *ConfigUserID* and *ConfigApplicationID*

**Usage:** setConfigAllID()

**Return:** None

**Note:** Settings are not in real time.

**Example:** *drone.setConfigAllID()*

### 3.5.8 *sendConfigIDs()*

Identifies the following configuration by sending the session, user and application IDs for the drone's multi-configuration-mode, as stored in *ConfigSessionID*, *ConfigUserID* and *ConfigApplicationID*.

**Usage:** sendConfigIDs()

**Return:** None

**Note:** *setMConfig()* might do what you want.

**Example:** *drone.sendConfigIDs()*
*drone.setConfigIDs(¨video:video_codec¨,¨131¨)*

### 3.5.9 *setMConfig()*

Changes the drone's configuration to multi-configuration-mode.

**Usage:** setMConfig(parameter,value)

**Return:** None

| Name: | Type: | Description: |
|---|---|---|
| parameter | string | Changes value of parameter as displayed in variable *ConfigData*. |
| value | string | Value of configuration to set. |

**Note:** Settings are not in real time. *ConfigData* will be updated automatically after all settings have been processed. Necessary for most video settings.

**Example:** *drone.setMConfig(¨video:video_codec¨,¨131¨)*

## 3.6 Video

### 3.6.1 Video variables

***VideoImage***
Contains the actual video-image of the drone as an OpenCV2 image-type, when video is activated. Not editable.

***VideoImageCount***
Sequential number of decoded video-images which are stored in *VideoImage*. Not editable.

***VideoDecodeTimeStamp***
Time when the video-image, stored in *VideoImage*, was decoded. Not editable.

***VideoDecodeTime***
Time needed to decode the video-image, stored in *VideoImage*. Not editable.

***VideoReady***
Indicates the online status of the video processes. Not editable.

***SaveVideo***
Indicates, that there is no pre-processing of the drone's video-data-stream. Not settable. (Default: *False*)

### 3.6.2 *startVideo()*

Activates and processes drone's video.

**Usage:** startVideo(optional)

**Return:** None

| Name: | Type: | Description: |
|---|---|---|
| optional | boolean | If not set or *True*: activates and processes drone's video,<br>*False*: stops drone's video. |

**Note:** Pictures of the video are available by *VideoImage*, start may take a second.

**Example:** *drone.startVideo()*

### 3.6.3 *stopVideo()*

Deactivates drone's video.

**Usage:** stopVideo(optional)

**Return:** None

| Name: | Type: | Description: |
|---|---|---|
| optional | boolean | If not set or *True*: stops drone's video,<br>*False*: activates and processes drone's video. |

**Note:** -

**Example:** *drone.stopVideo()*

### 3.6.4 *saveVideo()*

All video pre-processing will be stopped.

**Usage:** saveVideo(optional)

**Return:** None

| Name: | Type: | Description: |
|---|---|---|
| optional | boolean | If not set or *True*: video save-mode,<br>*False*: optimized video mode. |

**Note:** No video pre-processing, useful for unknown non-MPEG-video-codecs. *slowVideo()* or *midVideo()* will not work in this mode.

**Example:** *drone.saveVideo(False)*

### 3.6.5 *fastVideo()*

Processes the drone's whole video-stream, when video is activated.

**Usage:** fastVideo(optional)

**Return:** It is the default video-mode.

| Name: | Type: | Description: |
|---|---|---|
| optional | boolean | If not set or *True*: normal video-mode, *False*: slow video-mode. |

**Note:** Default

**Example:** *drone.fastVideo()*

### 3.6.6 *midVideo()*

Only key-frames of the drone's video-stream will be processed, when video is activated.

**Usage:** midVideo(optional)

**Return:** None

| Name: | Type: | Description: |
|---|---|---|
| optional | boolean | If not set or *True*: slow video-mode with a short delay,<br>*False*: fast video-mode. |

**Note:** Depending on used video-codec, resolution and camera, every 15th to 30th picture (see table 3.1 below) will be processed with no delay, but compared to *slowVideo()* more performance is needed.

**Example:** *drone.midVideo()*

### 3.6.7 *slowVideo()*

Only key-frames of the drone's video-stream will be processed, when video is activated.

**Usage:** slowVideo(optional)

**Return:** None

| Name: | Type: | Description: |
|---|---|---|
| optional | boolean | If not set or *True*: slow video-mode, *False*: fast video-mode. |

**Note:** Depending on used video-codec, resolution and camera, every 15th to 30th picture (see table 3.1 below) will be processed with a delay of 2.5 to 5 seconds, but compared to *midVideo()* less performance is needed.

**Example:** *drone.slowVideo()*

| Mode | Camera | P-Frames |
|---|---|---|
| *sdVideo()* | Front | 14 |
| | Ground | 14 |
| *hdVideo()* | Front | 29 |
| | Ground | 19 |
| *mp4Video()* | Front | 14 |
| | Ground | 14 |

Table 3.1: Number of P-Frames following after an I-Frame, depending on used mode.

### 3.6.8 *frontCam()*

Switch to drone's front camera.

**Usage:** frontCam(optional)

**Return:** None

| Name: | Type: | Description: |
|---|---|---|
| optional | boolean | If not set or *True*: Switch to front camera,<br>*False*: Switch to ground camera. |

**Note:** Default, requires multi-configuration-mode, no real-time-setting.

**Example:** *drone.frontCam()*

### 3.6.9 *groundCam()*

Switch to drone's ground camera.

**Usage:** groundCam(optional)

**Return:** None

| Name: | Type: | Description: |
|---|---|---|
| optional | boolean | If not set or *True*: Switch to ground camera,<br>*False*: Switch to front camera. |

**Note:** Requires multi-configuration-mode, no real-time-setting.

**Example:** *drone.groundCam()*

### 3.6.10 *sdVideo()*

Sets the drone's video stream to H.264 encoded, with an image resolution of 640 × 360.

**Usage:** sdVideo(optional)

**Return:** None

| Name: | Type: | Description: |
|---|---|---|
| optional | boolean | If not set or *True*: Drone sends H.264-video-stream with 640×360, *False*: Drone sends HD H.264-video-stream. |

**Note:** Default, requires multi-configuration-mode, no real-time-setting.

**Example:** `drone.sdVideo()`

### 3.6.11 *hdVideo()*

Sets the drone's video stream to H.264 encoded, with an image resolution of 1280 × 720.

**Usage:** hdVideo(optional)

**Return:** None

| Name: | Type: | Description: |
|---|---|---|
| optional | boolean | If not set or *True*: Drone sends H.264-video with 1280 × 720, *False*: Drone sends SD H.264-video-stream. |

**Note:** Requires multi-configuration-mode, no real-time-setting.

**Example:** `drone.hdVideo()`

### 3.6.12 *mp4Video()*

Sets the drone's video stream to MPEG4.2 encoded, with an image resolution of 640 × 368.

**Usage:** mp4Video(optional)

**Return:** None

| Name: | Type: | Description: |
|---|---|---|
| optional | boolean | If not set or *True*: Drone sends MPEG4-video-stream with 640 × 368, *False*: Drone sends SD H.264-video-stream. |

**Note:** Requires multi-configuration-mode, no real-time-setting.

**Example:** `drone.mp4Video()`

### 3.6.13 *showVideo()*

Displays drone's video in a window.

**Usage:** showVideo(optional)

**Return:** None

| Name: | Type: | Description: |
|---|---|---|
| optional | boolean | If not set or *True*: starts displaying drone's video,<br>*False*: hides drone's video. |

**Note:** Do not change resolution or codec when this function is enabled.

**Example:** `drone.showVideo()`

### 3.6.14 *hideVideo()*

Hides drone's video.

**Usage:** hideVideo(optional)

**Return:** None

| Name: | Type: | Description: |
|---|---|---|
| optional | boolean | If not set or *True*: hides drone's video,<br>*False*: starts displaying drone's video. |

**Note:** -

**Example:** `drone.hideVideo()`

### 3.6.15 *videoFPS()*

Changes the frame rate of the video.

**Usage:** videoFPS(fps)

**Return:** None

| Name: | Type: | Description: |
|---|---|---|
| fps | integer | Frames per second of drone's video. Values: *1 - 60*. |

**Note:** Requires multi-configuration-mode, no real-time-setting.

**Example:** `drone.videoFPS(15)`

### 3.6.16 *videoBitrate()*

Changes the bitrate of the video stream.

**Usage:** videoBitrate(br)

**Return:** None

| Name: | Type: | Description: |
|---|---|---|
| br | integer | Kilobit per second for drone's video encoding.<br>Values: *250 - 20000*. |

**Note:** Requires multi-configuration-mode, no real-time-setting.

**Example:** `drone.videoBitrate(4000)`

## 3.7 Convenient Commands

### 3.7.1 *getBattery()*

Gets battery information of the drone.

**Usage:** getBattery()

**Return:** (batValue,batStatus)

| Name: | Type: | Description: |
|---|---|---|
| batValue | integer | Battery charge level in percent. |
| batValue | string | Drone's battery status, ¨OK¨ or ¨empty¨. |

**Note:** -

**Example:**
```
print ¨Battery:  ¨+str(drone.getBattery()[0])+¨% ¨
+str(drone.getBattery()[1])
```

### 3.7.2 *getKey()*

Gets the character of a pressed key.

**Usage:** getKey()

**Return:** key

| Name: | Type: | Description: |
|---|---|---|
| key | string | Character of a pressed key. |

**Note:** -

**Example:**
```
drone.getKey()
```

### 3.7.3 *angleDiff()*

Gets the difference between two measured angles.

**Usage:** angleDiff(base,degrees)

**Return:** difference

| Name: | Type: | Description: |
|---|---|---|
| base | float | First angle in *-180.0* to *180.0* degrees. |
| degrees | float | Second angle in *-180.0* to *180.0* degrees. |
| difference | float | The smallest difference between first and second angle. |

**Note:** -

**Example:**
```
drone.angleDiff(-170,170)
```

### 3.7.4 *printDefault()*

Prints text in default color.

**Usage:** printDefault(optional)

**Return:** None

| Name: | Type: | Description: |
|---|---|---|
| optional | string | Prints the optional string in default color, otherwise all following text will print in default color. |

**Note:** -

**Example:** `drone.printDefault("All normal")`

### 3.7.5 *printRed()*

Prints text in red color.

**Usage:** printRed(optional)

**Return:** None

| Name: | Type: | Description: |
|---|---|---|
| optional | string | Prints the optional string in red color, otherwise all following. |

**Note:** You may use `printDefault()` to disable red text.

**Example:** `drone.printRed("Error")`

### 3.7.6 *printGreen()*

Prints text in green color.

**Usage:** printGreen(optional)

**Return:** None

| Name: | Type: | Description: |
|---|---|---|
| optional | string | Prints the optional string in green color, otherwise all following. |

**Note:** You may use `printDefault()` to disable green text.

**Example:** `drone.printGreen("OK")`

### 3.7.7 *printYellow()*

Prints text in yellow color.

**Usage:** printYellow(optional)

**Return:** None

| Name: | Type: | Description: |
|---|---|---|
| optional | string | Prints the optional string in yellow color, otherwise all following. |

**Note:** You may use `printDefault()` to disable yellow text.

**Example:** `drone.printYellow("Warning !")`

### 3.7.8 *printBlue()*

Prints text in blue color.

**Usage:** printBlue(optional)

**Return:** None

| Name: | Type: | Description: |
|---|---|---|
| optional | string | Prints the optional string in blue color, otherwise all following. |

**Note:** You may use *printDefault()* to disable blue text.

**Example:** *drone.printBlue(¨Notice¨)*

### 3.7.9 *printPurple()*

Prints text in purple color.

**Usage:** printPurple(optional)

**Return:** None

| Name: | Type: | Description: |
|---|---|---|
| optional | string | Prints the optional string in purple color, otherwise all following. |

**Note:** You may use *printDefault()* to disable purple text.

**Example:** *drone.printPurple(¨Header¨)*

### 3.7.10 *printLineUp()*

Following printout will appear one line up.

**Usage:** printLineUp()

**Return:** None

**Note:** -

**Example:** *drone.printLineUp()*

### 3.7.11 *doggyHop()*

Drone hops like a happy dog.

**Usage:** doggyHop()

**Return:** None

**Note:** -

**Example:** *drone.doggyHop()*

### 3.7.12 *doggyNod()*

Drone nods.

**Usage:** doggyNod()

**Return:** None

**Note:** -

**Example:** `drone.doggyNod()`

### 3.7.13 *doggyWag()*

Drone wags like a happy dog.

**Usage:** doggyWag()

**Return:** None

**Note:** -

**Example:** `drone.doggyWag()`

## 3.8 Misc commands

### 3.8.1 Misc variables

***Version***
Shows the version of the PS-Drone-API. Not editable.

***startTime***
Stores the PS-Drone-API's starting-time. Not editable.

***showCommands***
If set, most commands from PS-Drone-API to the drone and communication between the processes will be displayed. This variable reacts like a command with max 0.1 second delay when changed. Manually editable. (Default: `False`)

***debug***
If set, debug messages will be displayed. This variable reacts like a command with max 0.1 second delay when changed. Manually editable. (Default: `False`)

***valueCorrection***
Enables the correction of most values, when out of range, if set. (Default: `True`)

### 3.8.2 *reset()*

Initiates a soft reset of the drone.

**Usage:** reset()

**Return:** None

**Note:** -

**Example:** `drone.reset()`

### 3.8.3 *thrust()*

Controls the motor thrust directly.

**Usage:** thrust(front left, front right, rear left, rear right)

**Return:** None

| Name: | Type: | Description: |
|---|---|---|
| front_left | int | Value from *0* to *255* for the front left motor thrust. |
| front_right | int | Value from *0* to *255* for the front right motor thrust. |
| rear_left | int | Value from *0* to *255* for the rear left motor thrust. |
| rear_right | int | Value from *0* to *255* for the rear right motor thrust. |

**Note:** Overrides the control of the drone's system.
The order of motors differs from their values in *"pwm"*-NavData-package.

**Example:** `drone.thrust(25,25,25,25)`

### 3.8.4 *pwm()*

Controls the motor thrust directly.

**Usage:** thrust(front left, front right, rear left, rear right)

**Return:** None

| Name: | Type: | Description: |
|---|---|---|
| front_left | int | Value from *0* to *511* for the front left motor thrust. |
| front_right | int | Value from *0* to *511* for the front right motor thrust. |
| rear_left | int | Value from *0* to *511* for the rear left motor thrust. |
| rear_right | int | Value from *0* to *511* for the rear right motor thrust. |

**Note:** Deprecated ! Overrides the control of the drone's system.
The order of motors differs from their values in *"pwm"*-NavData-package.

**Example:** `drone.pwm(50,50,50,50)`

### 3.8.5 *shutdown()*

Lands and disconnects from drone, determines PS-Drone-API.

**Usage:** shutdown()

**Return:** None

**Note:** Will run as last command, when the user's-program ended, as default.

**Example:** `drone.shutdown()`

### 3.8.6 *anim()*

Drone does pre-set movement.

**Usage:** anim(movement, duration)

**Return:** None

| **Name:** | **Type:** | **Description:** |
|---|---|---|
| movement | int | Number of animation, see list below. |
| duration | int | Duration of the animation |

**Note:** -

**Example:** `drone.anim(0,1)`

| anim | discription |
|---|---|
| **0** | Short tipping about 30° to the left |
| **1** | Short tipping about 30° to the right |
| **2** | Short tipping about 30° front down |
| **3** | Short tipping about 30° front up |
| **4** | Slide to the left |
| **5** | Slide to the right |
| **6** | Spin clockwise |
| **7** | Spin clockwise and goes down |
| **8** | Yaw shake |
| **9** | Dance by rotating |
| **10** | Dance left/right |
| **11** | Dance forward/backward |
| **12** | Dance upward/downward |
| **13** | Tipping right then left. Tipping front up then down |
| **14** | Tipping left then right. Tipping front down then up |
| **15** | Double tipping left, then right. Double tipping front down, then up |
| **16** | Flip ahead |
| **17** | Flip backward |
| **18** | Flip left |
| **19** | Flip right |

Table 3.2: Possible options for *anim()* and drone's behaviour.

### 3.8.7 *led()*

Drone shows pre-set sequences with the LEDs at the end of the arms.

**Usage:** led(animation, frequency, duration)

**Return:** None

| **Name:** | **Type:** | **Description:** |
|---|---|---|
| animation | integer | Number of animation, see list below. |
| frequency | float | Speed of animation |
| duration | integer | Duration of animation |

**Note:** -

**Example:** *drone.led(0,1.5,1)*

| anim. | description |
|---|---|
| **0** | All LEDs are red, then all LEDs are green |
| **1** | All LEDs are green, then all LEDs are off |
| **2** | All LEDs are red, then all LEDs are off |
| **3** | All LEDs are orange, then all LEDs are off |
| **4** | All LEDs flash fast green/red/off |
| **5** | Front LEDs alternate fast yellow/off, rear LEDs are red |
| **6** | Front LEDs are green, read LEDs are red |
| **7** | All LEDs are red |
| **8** | All LEDs are green |
| **9** | Front left LED is red, all other are off. The light moves clockwise. |
| **10** | All LEDs are off |
| **11** | Left LEDs are off, right: rear LED is red, then rear LED is off and front LED is yellow. Then all LEDs are off |
| **12** | Right LEDs are off, left: rear LED is red, then rear LED is off and front LED is yellow. Then all LEDs are off |
| **13** | Rear LEDs are red, then are off and front LEDs are yellow. All LEDs are off |
| **14** | Front left LED is green, all other LEDs are red |
| **15** | Front right LED is green, all other LEDs are red |
| **16** | Rear right LED is green, all other LEDs are red |
| **17** | Rear left LED is green, all other LEDs are red |
| **18** | Left LEDs are green, right LEDs are red |
| **19** | Left LEDs are red, right LEDs are green |
| **20** | Front LEDs are green, rear LEDs are red, then all LEDs are off |

Table 3.3: Possible options for *led()* and drone's behaviour.

### 3.8.8 *aflight()*

Makes the drone fly around and follow 2D tags the camera can detect.

**Usage:** aflight(flag)

**Return:** None

| Name: | Type: | Description: |
|---|---|---|
| flag | boolean | *True*: start flight, *True*: stop flight |

**Note:** Drone function, might be obsolete.

**Example:** `drone.aflight(True)`

### 3.8.9 *at()*

Sends a command and its parameters to the drone.

**Usage:** at(command,[parameters])

**Return:** None

| Name: | Type: | Description: |
|---|---|---|
| command | string | Drone command. |
| parameters | various | Command's parameter(s) |

**Note:** For valid commands and parameters check AR.Drone 2.0 developer guide.

**Example:** `drone.at(¨PCMD¨, [0,0.0,0.0,0.0,0.0])`

# 4 NavData packages

NavData are sent as blocks, including the sensor-measurements and status information; each block is divided into a bunch of 28 packages which contain a specific set of values. The names of the packages in this API remain the same as in Parrot's SDK.

```
"demo", "time", "raw_measures", "phys_measures", "gyros_offsets",
"euler_angles", "references", "trims", "rc_references", "pwm",
"altitude", "vision_raw", "vision_of", "vision", "vision_perf",
"trackers_send", "vision_detect", "watchdog", "adc_data_frame",
"video_stream", "games", "pressure_raw", "magneto", "wind_speed",
"kalman_pressure", "hdvideo_stream", "wifi", "zimmu_3000", "chksum"
```

Listing 4.1: List of available NavData-packages

The drone provides two modes for sending its NavData, a ¨demo¨- and a ¨full¨-mode. Depending on the mode, 15 or 200 NavData are sent per second.

The availability of the NavData-packages also differs depending on whether the demo-mode is enabled or disabled.
With enabled demo-mode, by default, only the packages *"demo"*, *"vision_detect"* and *"chksum"* are available 15 times a second, but most of the time that is enough. With disabled demo-mode, by default, all 28 packages are sent 200 times per second. There are also state-tags available; however, like most other packages, they are not very important and are always sent with each of NavData's header.

## 4.1 State

The following entries can be found at the APIs State-variable.

| No | Name | 0: | 1: |
|---|---|---|---|
| [0] | Fly mask | Drone landed | Drone is flying |
| [1] | Video mask | Video disabled | Video enabled |
| [2] | Vision mask | Vision disabled | Vision enabled |
| [3] | Control algo | Euler angles control | Angular speed control |
| [4] | Altitude control | Altitude control inactive | Altitude control active |
| [5] | User feedback | Start button not pressed | Start button pressed |
| [6] | Control command ACK | None | received |
| [7] | Camera mask | Camera not ready | Camera ready |
| [8] | Travelling mask | disabled | enabled |
| [9] | USB key | USB flash drive is not ready | USB flash drive is ready |
| [10] | NavData demo | All NavData | NavData demo |
| [11] | Navdata bootstrap | Options sent | No NavData options sent |
| [12] | Motors status | OK | Problem |
| [13] | Communication lost | Com is ok | Com problem |
| [14] | Software fault | OK | Software fault detected |
| [15] | Batterystatus | OK | too low |
| [16] | User emergency landing | User EL is OFF | User EL is ON |
| [17] | Timer elapsed | Not elapsed | Elapsed |
| [18] | Magnetometer calib | No calibration needed | Calibration needed |
| [19] | Angles | OK | Out of range |
| [20] | Wind mask | OK | Too much wind |
| [21] | Ultrasonic sensor | OK | Deaf |
| [22] | Cutout system | Not detected | Detected |
| [23] | PIC Version number | Bad version number | Version number is OK |
| [24] | ATCodec thread | OFF | ON |
| [25] | Navdata thread | OFF | ON |
| [26] | Video thread | OFF | ON |
| [27] | Acquisition thread | OFF | ON |
| [28] | CTRL watchdog | Well scheduled | Delay ($>5$ms) |
| [29] | ADC watchdog | Uart2 is good | Delay ($>5$ms) |
| [30] | Com Watchdog | Com is ok | Com problem |
| [31] | Emergency landing | No emergency | Emergency |

Table 4.1: State-entries and their meanings

## 4.2 NavData

Most of the packages are not well documented. The following sections contain all information found, mostly in crude form. The entries' names remain the same as in Parrot's AR.Drone 2.0 SDK. Parrot's AR.Drone 2.0 SDK packages' names are the these names, covert in ¨NAVDATA_¨ and ¨_TAG¨, e.g. ¨NAVDATA_DEMO_TAG¨.

### 4.2.1 ¨demo¨

| Pos | Datatype | Name | Note |
|---|---|---|---|
| [0][0] | bool | default | |
| [0][1] | bool | init | |
| [0][2] | bool | landed | Landed, or flight-mode not initialled |
| [0][3] | bool | flying | Initialled flight-mode |
| [0][4] | bool | hovering | Better name would be ¨landing¨ |
| [0][5] | bool | test | |
| [0][6] | bool | trans_takeoff | |
| [0][7] | bool | trans_gofix | |
| [0][8] | bool | trans_landing | |
| [0][9] | bool | trans_looping | |
| [0][10] | bool | trans_no_vision | |
| [0][11] | bool | num_state | |
| [1] | uint32 | vbat_flying_percentage | Charge of battery in percent |
| [2][0] | float | theta | Pitch in degrees |
| [2][1] | float | phi | Roll in degrees |
| [2][2] | float | psi | Yaw in degrees |
| [3] | float | altitude | Altitude in centimetres |
| [4][0..2] | float | vx/vy/vz | Estimated speed in X/Y/Z in mm/s |
| [5] | uint32 | num_frames | Streamed frame index, don't use ! |
| [6][0..8] | matrix33 | detection_camera_rot | Camera params computed by detection |
| [7][0..2] | vector3 | detection_camera_trans | Deprecated ! Don't use ! |
| [8] | uint32 | detection_tag_index | Deprecated ! Don't use ! |
| [9] | uint32 | detection_camera_type | Type of tag |
| [10][0..8] | matrix33 | drone_camera_rot | Camera parameters computed by drone |
| [11][0..2] | vector3 | drone_camera_trans | Deprecated ! Don't use ! |

### 4.2.2 ¨time¨

| Pos | Datatype | Name | Note |
|---|---|---|---|
| - | float | time | Time in seconds |

### 4.2.3 ¨wifi¨

| Pos | Datatype | Name | Note |
|---|---|---|---|
| - | uint32 | link_quality | |

### 4.2.4 ¨magneto¨

| Pos | Datatype | Name | Note |
|---|---|---|---|
| [0][0..2] | int16 | mx/my/mz | Magnetometer measurements [xyz] |
| [1][0..2] | float | magneto_raw | Magneto in the body frame [mG] |
| [2][0..2] | float | magneto_rectified | |
| [3][0..2] | float | magneto_offset | |
| [4] | float | heading_unwrapped | |
| [5] | float | heading_gyro_unwrapped | |
| [6] | float | heading_fusion_unwrapped | |
| [7] | uint8 | magneto_calibration_ok | |
| [8] | uint32 | magneto_state | |
| [9] | float | magneto_radius | |
| [10] | float | error_mean | |
| [11] | float | error_var | |

### 4.2.5 ¨altitude¨

| Pos | Datatype | Name | Note |
|---|---|---|---|
| [0] | int32 | altitude_vision | [mm] |
| [1] | float | altitude_vz | [mm/2] |
| [2] | int32 | altitude_ref | [mm] |
| [3] | int32 | altitude_raw | [mm] |
| [4] | float | obs_accZ | Observer AccZ [m/s$^2$] |
| [5] | float | obs_alt | Observer altitude US [m] |
| [6][0..2] | vector3 | obs_x | |
| [7] | uint32 | obs_state | Observer state |
| [8][0..1] | Vector2 | est_vb | |
| [9] | uint32 | est_state | Observer flight state |

### 4.2.6 ¨pressure_raw¨

| Pos | Datatype | Name | Note |
|---|---|---|---|
| [0] | int32 | up | |
| [1] | int16 | ut | |
| [2] | int32 | temperature_meas | |
| [3] | int32 | pression_meas | |

### 4.2.7 ¨wind_speed¨

| Pos | Datatype | Name | Note |
|---|---|---|---|
| [0] | float | wind_speed | |
| [1] | float | wind_angle | |
| [2][0] | float | wind_compensation_theta | |
| [2][1] | float | wind_compensation_phi | |
| [3][0..5] | float | state_x[6] | |
| [4][0..2] | float | magneto_debug[3] | |

### 4.2.8 ¨kalman_pressure¨

| Pos | Datatype | Name | Note |
|---|---|---|---|
| [0] | float | offset_pressure | |
| [1] | float | est_z | Estimated hight |
| [2] | float | est_zdot | |
| [3] | float | est_bias_PWM | |
| [4] | float | est_biais_pression | |
| [5] | float | offset_US | |
| [6] | float | prediction_US | |
| [7] | float | cov_alt | |
| [8] | float | cov_PWM | |
| [9] | float | cov_vitesse | |
| [10] | bool | bool_effet_sol | |
| [11] | float | somme_inno | |
| [12] | bool | flag_rejet_US | |
| [13] | float | u_multisinus | |
| [14] | float | gaz_altitude | |
| [15] | bool | flag_multisinus | |
| [16] | bool | flag_multisinus_debut | |

### 4.2.9 ¨zimmu_3000¨

| Pos | Datatype | Name | Note |
|---|---|---|---|
| [0] | int32 | vzimmu | |
| [1] | float | vzfind | |

### 4.2.10 ¨raw_measures¨

| Pos | Datatype | Name | Note |
|---|---|---|---|
| [0][0..2] | uint16 | raw_accs[xyz] | filtered accelerometer-data [LSB] |
| [1][0..2] | int16 | raw_gyros[xyz] | filtered gyrometer-data [LSB] |
| [2][0..2] | int16 | raw_gyros_110[xyz] | gyrometers x/y 110 deg/s [LSB] |
| [3] | uint32 | vbat_raw | battery voltage [mV] |
| [4] | uint16 | us_debut_echo | |
| [5] | uint16 | us_fin_echo | |
| [6] | uint16 | us_association_echo | |
| [7] | uint16 | us_distance_echo | |
| [8] | uint16 | us_courbe_temps | |
| [9] | uint16 | us_courbe_valeur | |
| [10] | uint16 | us_courbe_ref | |
| [11] | uint16 | flag_echo_ini | |
| [12] | uint16 | nb_echo | |
| [13] | uint32 | sum_echo | (just lower 16Bit (higher 16Bit=tags?)) |
| [14] | uint32 | alt_temp_raw | in millimetre (just lower 16Bit) |
| [15] | uint16 | gradient | |

### 4.2.11 "phys_measures"

| Pos | Datatype | Name | Note |
|---|---|---|---|
| [0] | float | accs_temp | |
| [1] | uint16 | gyro_temp | |
| [2][0..2] | float | phys_accs[xyz] | |
| [3][0..2] | float | phys_gyros[xyz] | |
| [4] | uint32 | alim3V3 | 3.3volt alim |
| [5] | uint32 | vrefEpson | Ref volt Epson gyro |
| [6] | uint32 | vrefIDG | Ref volt IDG gyro |

### 4.2.12 "references"

| Pos | Datatype | Name | Note |
|---|---|---|---|
| [0][0] | int32 | ref_theta | Theta ref embedded [milli-deg] |
| [0][1] | int32 | ref_phi | Phi ref embedded [milli-deg] |
| [0][2] | int32 | ref_psi | Psi ref embedded [milli-deg] |
| [1][0] | int32 | ref_theta_I | Theta ref int [milli-deg] |
| [1][1] | int32 | ref_phi_I | Phi ref int [milli-deg]] |
| [2][0] | int32 | ref_pitch | Pitch ref embedded [milli-deg] |
| [2][1] | int32 | ref_roll | Roll ref embedded [milli-deg] |
| [2][2] | int32 | ref_yaw | Yaw ref embedded [milli-deg/s] |
| [3][0] | float | vx_ref | Vx Ref [mm / s] |
| [3][1] | float | vy_ref | Vy Ref [mm / s] |
| [4][0] | float | theta_mod | Theta modele [radian] |
| [4][1] | float | phi_mod | Phi modele [radian] |
| [5][0] | float | k_v_x | |
| [5][1] | float | k_v_y | |
| [6] | uint32 | k_mode | |
| [7][0] | float | ui_time | |
| [7][1] | float | ui_theta | |
| [7][2] | float | ui_phi | |
| [7][3] | float | ui_psi | |
| [7][4] | float | ui_psi_accuracy | |
| [7][5] | int32 | ui_seq | |

### 4.2.13 "rc_references"

| Pos | Datatype | Name | Note |
|---|---|---|---|
| [0] | int32 | rc_ref_pitch | Pitch rc embedded |
| [1] | int32 | rc_ref_roll | Roll rc embedded |
| [2] | int32 | rc_ref_yaw | Yaw rc embedded |
| [3] | int32 | rc_ref_gaz | Gaz rc embedded |
| [4] | int32 | rc_ref_ag | Ag rc embedded |

### 4.2.14 ¨gyros_offsets¨

| Pos | Datatype | Name | Note |
|---|---|---|---|
| [0][0..2] | float | offset_g[xyz] | [deg/s] |

### 4.2.15 ¨euler_angles¨

| Pos | Datatype | Name | Note |
|---|---|---|---|
| [0] | float | theta_a | head / back |
| [1] | float | phi_a | sides |

### 4.2.16 ¨watchdog¨

| Pos | Datatype | Name | Note |
|---|---|---|---|
| - | uint32 | watchdog | Watchdog controll |

### 4.2.17 ¨trims¨

| Pos | Datatype | Name | Note |
|---|---|---|---|
| [0] | float | angular_rates_trim | |
| [1] | float | euler_angles_trim_theta | [milli-deg] |
| [2] | float | euler_angles_trim_phi | [milli-deg] |

### 4.2.18 ¨pwm¨

| Pos | Datatype | Name | Note |
|---|---|---|---|
| [0][0..3] | uint8 | motor[1234] | [Pulse-width mod] |
| [1][0..3] | uint8 | sat_motor[1234] | [Pulse-width mod] |
| [2] | float | gaz_feed_forward | [Pulse-width mod] |
| [3] | float | gaz_altitud | [mm/s] |
| [4] | float | altitude_integral | [mm/s] |
| [5] | float | vz_ref | [Pulse-width mod] |
| [6][0] | int32 | u_pitch | [Pulse-width mod] |
| [6][1] | int32 | u_roll | [Pulse-width mod] |
| [6][2] | int32 | u_yaw | [Pulse-width mod] |
| [7] | float | yaw_u_I | [Pulse-width mod] |
| [8][0] | int32 | u_pitch_planif | [Pulse-width mod] |
| [8][1] | int32 | u_roll_planif | [Pulse-width mod] |
| [8][2] | int32 | u_yaw_planif | [Pulse-width mod] |
| [8][3] | float | u_gaz_planif | [Pulse-width mod] |
| [9][0..3] | uint16 | current_motor[1234] | [mA] |
| [10] | float | altitude_prop | [Pulse-width mod] |
| [11] | float | altitude_der | [Pulse-width mod] |

### 4.2.19 "vision"

| Pos | Datatype | Name | Note |
|---|---|---|---|
| [0] | uint32 | vision_state | |
| [1] | int32 | vision_misc | |
| [2] | float | vision_phi_trim | |
| [3] | float | vision_phi_ref_prop | |
| [4] | float | vision_theta_trim | |
| [5] | float | vision_theta_ref_prop | |
| [6] | int32 | new_raw_picture | |
| [7][0..2] | float | theta/phi/psi_capture | |
| [8] | int32 | altitude_capture | |
| [9] | float | time_capture | |
| [10][0..2] | float | velocities[xyz] | X-/Y-/Z-velocity |
| [11][0..2] | float | delta_phi/theta/psi | |
| [12] | uint32 | gold_defined | |
| [13] | uint32 | gold_reset | |
| [14][0] | float | gold_x | |
| [14][1] | float | gold_y | |

### 4.2.20 "vision_stream"

| Pos | Datatype | Name | Note |
|---|---|---|---|
| [0] | uint8 | quant | Quantizer reference used to encode |
| [1] | uint32 | frame_size | Frame size in bytes |
| [2] | uint32 | frame_number | Frame index |
| [3] | uint32 | atcmd_ref_seq | Atmcd ref sequence number |
| [4] | uint32 | atcmd_mean_ref_gap | Mean time between two consecutive atcmd_ref [ms] |
| [5] | float | atcmd_var_ref_gap | |
| [6] | uint32 | atcmd_ref_quality | Estimator of atcmd link quality |
| [7] | uint32 | out_bitrate | Measured out throughput from the video TCP-socket |
| [8] | uint32 | desired_bitrate | Size of last frame generated by the video encoder |
| [9][0..4] | int32 | data | Misc temporary data |
| [10] | uint32 | tcp_queue_level | Queue usage |
| [11] | uint32 | fifo_queue_level | Queue usage |

### 4.2.21 "vision_of"

| Pos | Datatype | Name | Note |
|---|---|---|---|
| [0][0..4] | float | of_dx | |
| [1][0..4] | float | of_dy | |

### 4.2.22 "vision_raw"

| Pos | Datatype | Name | Note |
|---|---|---|---|
| [0..2] | float | vision_tx_raw[xyz] | |

### 4.2.23 "vision_detect"

| Pos | Datatype | Name | Note |
|---|---|---|---|
| [0] | uint32 | nb_detected | Number of found markers |
| [1][0..3] | uint32 | type[4] | Type of detected marker |
| [2][0..3] | uint32 | xc[4] | X-pos of marker ( 0 - 1000 ), 0 = left |
| [3][0..3] | uint32 | yc[4] | Y-pos of marker ( 0 - 1000 ), 0 = top |
| [4][0..3] | uint32 | width[4] | Optical width of marker |
| [5][0..3] | uint32 | height[4] | Optical height of marker |
| [6][0..3] | uint32 | dist[4] | Distance of marker ( 0 - 1000 ), 0 = near |
| [7][0..3] | float | orientation_angle[4] | Orientation of marker [deg] |
| [8][0..3] | matrix33 | rotation[4] | |
| [9][0..3] | vector3 | rotation[4] | |
| [10][0..3] | uint32 | camera_source[4] | |

### 4.2.24 "vision_perf"

| Pos | Datatype | Name | Note |
|---|---|---|---|
| [0] | float | time_szo | |
| [1] | float | time_corners | |
| [2] | float | time_compute | |
| [3] | float | time_tracking | |
| [4] | float | time_trans | |
| [5] | float | time_update | |
| [6][0..19] | float | time_custom[20] | |

### 4.2.25 "hdvideo_stream"

| Pos | Datatype | Name | Note |
|---|---|---|---|
| [0] | float | hdvideo_state | |
| [1] | float | storage_fifo_nb_packets | |
| [2] | float | storage_fifo_size | |
| [3] | float | usbkey_size | USB flash drive in kb (no key=0) |
| [4] | float | usbkey_freespace | USB flash drive in kb (no key=0) |
| [5] | float | frame_number | PaVE field of the frame encoder started |
| [6] | float | usbkey_remaining_time | [sec] |

### 4.2.26 "games"

| Pos | Datatype | Name | Note |
|---|---|---|---|
| [0] | uint32 | double_tap_counter | |
| [1] | uint32 | finish_line_counter | |

### 4.2.27 ¨trackers_send¨

| Pos | Datatype | Name | Note |
|---|---|---|---|
| [0..29] | int32 | locked[30] | |
| [0..29][0..1] | int32 | point[x[30],y[30]] | |

### 4.2.28 ¨adc_data_frame¨

| Pos | Datatype | Name | Note |
|---|---|---|---|
| [0] | uint32 | version | |
| [0][0..31] | uint8 | data_frame[32] | |

### 4.2.29 ¨chksum¨

| Pos | Datatype | Name | Note |
|---|---|---|---|
| [0] | uint32 | chksum | Transmitted checksum |
| [1] | uint32 | | Checksums match |

# 5 Configuration entries

## 5.1 General

**general:num_version_config** *Std: read only*

Configuration subsystem's version.

**general:num_version_mb** *Std: read only*

Drone's mainboard hardware-version.

**general:num_version_soft** *Std: read only*

Drone's firmware-version.

**general:drone_serial** *Std: read only*

Drone's serial number.

**general:soft_build_date** *Std: read only*

Drone's firmware compilations date.

**general:motor1_soft** *Std: read only*

Software version of drone's motor subsystem 1.
Also available for motor subsystem 2, 3 and 4.

**general:motor1_hard** *Std: read only*

Hardware version of drone's motor subsystem 1.
Also available for motor subsystem 2, 3 and 4.

**general:motor1_supplier** *Std: read only*

Supplier code of drone's motor subsystem 1.
Also available for motor subsystem 2, 3 and 4.

**general:ardrone_name** *Std: read/write*

Drone's name for AR-game development.
Note: The name is not related to WiFi-SSID.

**general:flying_time** *Std: read only*

Time (in seconds) the drone spent in a flying state in its whole lifetime.

**general:navdata_demo** *Std: read/write*

Tag that indicates the drone's send mode for NavData.
`TRUE` means that NavData are sent in demo mode, `FALSE` stands for full send mode. See section 4 on page 47 for further details.
This setting can also be done by the command `useDemoMode()` and `useMDemoMode()`, described in section 3.4.1 on page 29.

**general:navdata_options** *App: read/write*

Enabled NavData-packages sent by the drone. The basic demo-mode-packages `state`, `demo`, `vision_detect` and `chksum` are always sent and cannot be disabled.

| | | | |
|---|---|---|---|
| `demo` : | 1 | `vision_perf` : | 16384 |
| `time` : | 2 | `trackers_send` : | 32768 |
| `raw_measures` : | 4 | `vision_detect` : | 65536 |
| `phys_measures` : | 8 | `watchdog` : | 131072 |
| `gyros_offsets` : | 16 | `adc_data_frame` : | 262144 |
| `euler_angles` : | 32 | `video_stream` : | 524288 |
| `references` : | 64 | `games` : | 1048576 |
| `trims` : | 128 | `pressure_raw` : | 2097152 |
| `rc_references` : | 256 | `magneto` : | 4194304 |
| `pwm` : | 512 | `wind_speed` : | 8388608 |
| `altitude` : | 1024 | `kalman_pressure` : | 16777216 |
| `vision_raw` : | 2048 | `hdvideo_stream` : | 33554432 |
| `vision_of` : | 4096 | `wifi` : | 67108864 |
| `vision` : | 8192 | `zimmu_3000` : | 134217728 |

Table 5.1: Values to set NavData-package to be sent

Add values for packages you want to get, e.g.:
`demo` + `time` + `trims` + `vision_detect` = 65667.
Default packages in demo-mode and in full-mode: 65537 and 268435455.

**general:vbat_min** *Std: read only*

Minimum battery charge before drone lands automatically.

**general:vision_enable** *Std: read/write*

*Reserved for future use, value has to be `TRUE` to prevent unexpected behaviour.*
Note: Not related to tag detection.

**general:localtime** *Std: read only*

*No description available*

**general:video_enable** *Std: read/write*

*Reserved for future use, value has to be `TRUE` to prevent unexpected behaviour.*

**general:com_watchdog** *Std: read/write*

Shows the time (in seconds) the drone accepts not to receive any command from a client before it will cut the connection. Default: `2`
Officially, the drone should also stop its moving and hover on its position.
Note: Only important for connection-losses as the PS-Drone-API sends keep-alive commands to prevent the drone becoming passive.

## 5.2 Control

**control:altitude_max** *Std: read/write*

Drone's maximum altitude in millimetres. The altitude is measured by the pressure-sensor and has no limitation.

**control:altitude_min** *Std: read/write*

Drone's minimum altitude in millimetres. Might cause problems regarding stability when modified.

**control:flight_without_shell** *Std: read/write*

Indicates the shell the drone expects to be covered. When set to flying with outdoor shell by change to `True`, several internal optimizations regarding flight behaviour will be done.
Default: `False` / indoor-shell.

**control:autonomous_flight** *Std: read/write*

Indicates autonomous flight-mode. By default disabled (controlled by client), can cause unexpected behaviour when set `True`.
*Deprecated and not longer maintained by Parrot.*

**control:accs_offset** *Std: read only*

Accelerometers offset. *For Parrot's internal debugging.*

**control:accs_gains** *Std: read only*

Accelerometers gains. *For Parrot's internal debugging.*

**control:gyros_offset** *Std: read only*

Gyrometers offset. *For Parrot's internal debugging.*

**control:gyros_gains** *Std: read only*

Gyrometers gains. *For Parrot's internal debugging.*

**control:gyros110_offset** *Std: read only*

*For Parrot's internal debugging.*

**control:gyros110_gains** *Std: read only*

*For Parrot's internal debugging.*

**control:magneto_offset** *Std: read only*

*For Parrot's internal debugging.*

**control:magneto_radius** *Std: read only*

*For Parrot's internal debugging.*

**control:gyro_offset_thr_x** *Std: read only*

*For Parrot's internal debugging.* Also available for Y- and Z-axis.

**control:pwm_ref_gyros** *Std: read only*

*For Parrot's internal debugging.*

**control:osctun_value** *Std: read only*

*For Parrot's internal debugging.*

**control:osctun_test** *Std: read only*

*For Parrot's internal debugging.*

**control:flying_mode** *Sess: read/write*

Displays the current flight mode. There are three flight modes available by setting a value:

| | |
|---|---|
| 0 | Legacy mode, the drone is controlled by a client (Default) |
| 1 | Semi-autonomous mode, the drone hovers above a roundel-tag (page 75) |
| 2 | Semi-autonomous mode, the drone hovers above an oriented-black-and-white-roundel-tag and faces in tags direction (page 77). |

Table 5.2: Settings for different flying-modes.

Note: You have to choose the tag you use by editing `detect:detect_type`, described on page 64.

**control:hovering_range** *Sess: read/write*

Drone's maximum allowed distance to an oriented-roundel in millimetres. Only in use when the drone is in a semi-autonomous-flight-mode, set by `control:flying_mode`.

**control:manual_trim** *Usr: read/write*

Tag that indicates that the drone uses manual trims.
`FALSE` stands for automated trims, `TRUE` means activated manual trims.
Note: Parrot suggests not to use manual trims.

**control:flight_anim** *Std: read/write*

When set, drone flies a pre-set movement. First value sets the type of movement, second value sets the duration.

| anim | discription |
|---|---|
| **0** | Short tipping about 30° to the left |
| **1** | Short tipping about 30° to the right |
| **2** | Short tipping about 30° front down |
| **3** | Short tipping about 30° front up |
| **4** | Slide to the left |
| **5** | Slide to the right |
| **6** | Spin clockwise |
| **7** | Spin clockwise and goes down |
| **8** | Yaw shake |
| **9** | Dance by rotating |
| **10** | Dance left/right |
| **11** | Dance forward/backward |
| **12** | Dance upward/downward |
| **13** | Tipping right then left. Tipping front up then down |
| **14** | Tipping left then right. Tipping front down then up |
| **15** | Double tipping left, then right. Double tipping front down, then up |
| **16** | Flip ahead |
| **17** | Flip backward |
| **18** | Flip left |
| **19** | Flip right |

Table 5.3: Possible first values for *control:flight_anim* and drone's behaviour.

The setting is also implemented by PS-Drone-API-commands *anim()*, described in section 3.8.5 on page 43.

**control:control_level** *App: read/write*

Shows how the drone will react to user's movement-commands. Normally, the drone turns on a horizontal axis and is always in a stable condition. In race-mode, the drone moves more dynamically and may "lean into the bend" while rotating.

| | |
|---|---|
| 1 | Normal movement, only turning |
| 3 | Race-mode, lean into the bend |

Table 5.4: Settings for movement behaviour control.

Other values could cause unexpected behaviour of the drone.

**control:outdoor** *Std: read/write*

Indicates which surrounding the movement settings are optimized for. When set to outdoor-mode by change to `True`, wind estimation will be enabled. Furthermore, values for the drone's behaviour will be taken from `control:outdoor_control_yaw`, `control:outdoor_euler_angle_max` and `control:outdoor_control_vz_max`, instead of the values from `control:indoor_control_yaw`, `control:indoor_euler_angle_max` and `control:indoor_control_vz_max`. Default: `False` / indoor.

**control:euler_angle_max** *Usr: read/write*

Maximum slope for pitch and roll angles in radians (1.0 rad ≈ 57.295°). This effects directly the maximum speed of the drone meaning that 100% speed (respectively the value `1.0`) of the movement-commands (section 3.3 beginning on page 26) will be faster or slower, depending on this setting. The value should be between `0.0` and `0.52` (≈30°). Higher values might work, but the drone might have problems to hold its altitude and nosedive.
By editing this parameter, the suitable values and the drone's behaviour indoors respectively outdoors (depending on the `control:outdoor`-status) will be changed.

**control:control_vz_max** *Usr: read/write*

Drone's maximum vertical speed in millimetre per second. The value should be between `200` and `2000`. Beyond these values the drone may become unstable.
By editing this parameter, the suitable values and the drone's behaviour indoors respectively outdoors (depending on the `control:outdoor`-status) will be changed.

**control:control_yaw** *Usr: read/write*

Drone's maximum speed for yaw angles in radians per second. The value should be between `0.7` and `6.11` (≈ 40°/s to 350°/s). Beyond these values the drone may become unstable.
By editing this parameter, the suitable values and the drone's behaviour indoors respectively outdoors (depending on the `control:outdoor`-status) will be changed.

**control:indoor_euler_angle_max** *Usr: read/write*

Maximum slope for pitch and roll angles in radians (1.0 rad ≈ 57.295°). This effects directly the maximum speed of the drone meaning that 100% speed (respectively the value `1.0`) of the movement-commands (section 3.3 beginning on page 26), will be faster or slower, depending on this setting. The value should be between `0.0` and `0.52` (≈30°). Higher values might work, but the drone might have problems to hold its altitude and nosedive.
Used when `control:outdoor` is `False` and may already be set by editing `control:euler_angle_max`.

**control:indoor_control_vz_max** *Usr: read/write*

Drone's maximum vertical speed in millimetre per second. This value should be between *200* and *2000*. Beyond these values the drone may become unstable. Used when *control:outdoor* is *False* and may already be set by editing *control:control_vz_max*.

**control:indoor_control_yaw** *Usr: read/write*

Drone's maximum yaw-speed angles in radians per second. The value should be between *0.7* and *6.11* ($\approx 40°$/s to $350°$/s). Beyond these values the drone may become unstable.
Used when *control:outdoor* is *False* and may already be set by editing *control:control_yaw*.

**control:outdoor_euler_angle_max** *Usr: read/write*

Maximum slope for pitch and roll angles in radians (1.0 rad $\approx 57.295°$). This effects directly the maximum speed of the drone meaning that 100% speed (respectively the value *1.0*) of the movement-commands (section 3.3 beginning on page 26), will be faster or slower, depending on this setting. The value should be between *0.0* and *0.52* ($\approx 30°$). Higher values might work, but the drone might have problems to hold its altitude and nosedive.
Used when *control:outdoor* is *True* and may already be set by editing *control:euler_angle_max*.

**control:outdoor_control_vz_max** *Usr: read/write*

Drone's maximum vertical speed in millimetre per second. This value should be between *200* and *2000*. Beyond these values the drone may become unstable.
Used when *control:outdoor* is *True* and may already be set by editing *control:control_vz_max*.

**control:outdoor_control_yaw** *Usr: read/write*

Drone's maximum yaw-speed in radians per second. The value should be between *0.7* and *6.11* ($\approx 40°$/s to $350°$/s). Beyond these values the drone may become unstable.
Used when *control:outdoor* is *True* and may already be set by editing *control:control_yaw*.

**control:control_iphone_tilt** *Usr: read/write*

Correction angle in radians (1.0 rad $\approx 57.295°$) when drone is controlled by accelerometers of an iPhone. A smart phone is normally held in a tilted position and not flat (meaning that the drone would constantly fly backwards): in order to accelerate and decelerate the drone by tilting the phone, the correction angle is necessary. Default: 0.34906584 ($\approx 20°$).

## 5.3 Detect

**detect:detect_type** *Sess: read/write*

Shows what type of tag/marker is detectable.
Modifying this seems useless as the detection works always anyway (tested with firmware version 2.3.3) as soon as a roundel-/marker-/tag-type has been set in *detect:detections_select_h* or *detect:detections_select_v*.

| Val | Type of detection | Note |
|---|---|---|
| *2* | 2D horizontal tags on drone shells | - |
| *3* | Detection disabled | - |
| *4* | Roundel on ground | - |
| *5* | Oriented roundel on ground | - |
| *6* | Uniform stripe on ground | - |
| *7* | Roundel in front | - |
| *8* | Oriented roundel in front | - |
| *9* | Vertical Stripes | - |
| *10* | Several detections at the same time | suggested |
| *11* | Orange and green cap in front | - |
| *12* | Black and white roundel | - |
| *13* | 2nd version of shell/tag in front | - |
| *14* | Tower-side in front | - |

Table 5.5: Official values to choose tag detection methods.

*detect:detections_select_h* and *detect:detections_select_v* will be reset to *0*, if *detect:detect_type* is modified.
The roundel and a modified roundel can be found on pages 75 and 76, the black and white oriented roundel is on page 77.

**detect:detections_select_h** *Sess: read/rwite*

Enables tags for detection by front camera.

| | | |
|---|---|---|
| Shell-Tag : | 1 | |
| Roundel : | 2 | find them on page 75 & 76 |
| Black Roundel : | 4 | |
| Stripe : | 8 | |
| Cap : | 16 | |
| Shell-Tag V2 : | 32 | |
| Tower Side : | 64 | |
| Oriented Roundel : | 128 | find it on page 77 |

Table 5.6: Tag detection enabling values.

Add the values to detect more than one tag. Value will be set to *0* when *detect:detect_type* is modified.

**detect:detections_select_v_hsync** *Sess: read only*

Enables tags for detection by ground camera, synchronized with the front camera. See table 5.6 within description of *detect:detections_select_h* above for the values to enable tags.
Value will be set to *0* when *detect:detect_type* is modified.
It is recommended not to use *detect:detections_select_v* and this tag on ground detection at the same time.

**detect:detections_select_v** *Sess: read/write*

Enables tags for detection by ground camera. See table 5.6 within description of *detect:detections_select_h* above for the values to enable tags.
Value will be set to *0* when *detect:detect_type* is modified.
It is recommended not to use *detect:detections_select_v_hsync* and this tag on ground detection at the same time.

**detect:enemy_colors** *Std: read/write*

Type of game objects and other drone detection. Use following values to modify:

| | |
|---|---|
| 1 | Drones with green and orange coverage |
| 2 | Drones with yellow and orange coverage |
| 3 | Drones with blue and orange coverage |
| 16 | AR-race finish line (not official) |
| 17 | AR-race donut (not official) |

Table 5.7: Settings for different flying-modes.

**detect:enemy_without_shell** *Std: read/write*

Type of detecting enemy drones coverage.

| | |
|---|---|
| 0 | Detection of drones with indoor coverage |
| 1 | Detection of drones with outdoor coverage |

Table 5.8: Settings to change coverage detection of enemies.

## 5.4 Video

**video:camif_fps** *Std: read only*

Actual frame rate of the video interface.

**video:camif_buffers** *Std: read only*

Shows the buffer-size of the video interface.

**video:video_slices** *Sess: read/write*

*For Parrot's internal debugging, do not modify.*

**video:num_trackers** *Std: read only*

Used number of tracking-points for optical speed estimation.

**video:video_live_socket** *Sess: read/write*

*For Parrot's internal debugging, do not modify.*

**video:video_codec** *Sess: read only* *MConf: read/write*

The actual used videocodec, the resolution of video and usage of a recording-video-stream.

| Value | Codec | Resolution | Recording-stream | Note |
|---|---|---|---|---|
| *128* | MPEG4 | 640 × 368 | disabled | *mp4Video()* (3.6.11, P. 37) |
| *129* | H.264 | 640 × 360 | disabled | *sdVideo()* (3.6.9, P. 37) |
| *130* | MPEG4 | 640 × 368 | H.264 1280 × 720 | - |
| *131* | H.264 | 1280 × 720 | disabled | *hdVideo()* (3.6.10, P. 37) |
| *132* | MPEG4 | 640 × 368 | disabled | SLR-coded stream |
| *133* | H.264 | 640 × 360 | disabled | SLR-coded stream |
| *134* | H.264 | 1280 × 720 | disabled | SLR-coded stream |
| *135* | H.264 | dynamic | disabled | resolution according to bitrate |
| *136* | MPEG4 | 640 × 368 | H.264 640 × 360 | - |

Table 5.9: Values for the video-codec, -resolution and recording-stream usage.

MPEG4 is also known as MPEG4.2, H264 is also known as MPEG4.10. The term SLR is not clear, could stand for "Simple LR parser". Some values can be set by PS-Drone-API-commands. The video-recording-stream has always 30 fps and will, when enabled, be stored on an USB flash drive or sent to the client.

**video:video_channel** *Sess: read/write*

Index of the drone's actively used camera:

| | | |
|---|---|---|
| 0 : | Frontcamera (Default) | implemented by *frontCam()*, 3.6.8, P. 36 |
| 1: | Groundcamera | implemented by *groundCam()*, 3.6.9, P. 36 |

Table 5.10: Settings to switch used camera.

**video:codec_fps** *Sess: read/write*

Used frame rate of the video-stream. The frame rate can be set from 1 to 30 fps.
This setting can also be done by the command *videoBitrate()*, described in section 3.6.15 on page 38.

**video:bitrate_ctrl_mode** *Std: read only* *MConf: read/write*

Status of the drone's video-stream bitrate-control. Altering the bitrate-control-mode may reduce the video-stream's bandwidth.

| | |
|---|---|
| 0 : | Constant bitrate as set in `video:max_bitrate` (Default) |
| 1 : | Bitrate variates from 250kBit to the value set in `video:max_bitrate` |
| 2 : | Constant bitrate as set in `video:bitrate` |

Table 5.11: Values to change the bitrate-control-mode.

**video:max_bitrate** *Std: read only*

Bitrate of the drone's video-stream in kBit. If `video:bitrate_ctrl_mode` is set to `0`, this value will be used as constant bitrate and will be used as the maximum of a variable bitrate, if `video:bitrate_ctrl_mode` was set to `1`. Parrot recommends the range from 500 to 4000 as typical values, depending on what bitrate the client is able to decode.
This setting can also be done by the command `videoBitrate()`, described in section 3.6.15 on page 38.

**video:bitrate** *Std: read only* *MConf: read/write*

Bitrate of the drone's video-stream in kBit.
This value is used, if `video:bitrate_ctrl_mode` is set to `2`. Parrot recommends the range from 500 to 4000 as typical values.
This setting can also be done by the command `videoBitrate()`, described in section 3.6.15 on page 38.

**video:video_file_index** *Std: read/write*

The number of the last recorded video on the USB flash drive.
Parrot suggests not to modify this value.

**video:bitrate_storage** *App: read/write*

Bitrate of the video-recording stream.

**video:video_storage_space** *Std: read only*

Used size of the WiFi-video-recording-buffer.

**video:video_on_usb** *Std: read/write*

Usage of an USB-Stick as video-recording-device.
If an USB flash drive is connected and this value is set to `True`, the drone's video-recording-stream will be stored on the device.
If the value is set to `False`, no USB flash drive is connected or the available storage space is less then 100MB; the video-recording-stream will be available per network stream.
The video-recording-stream is always H.264-encoded, with 30 frames per second and a resolution of $640 \times 360$ or $1280 \times 720$. If not recorded on an USB flash drive, it will be available at TCP-port 5553.

## 5.5 LEDs, Syslog and Pic

**leds:leds_anim** *Std: read/write*

Drone shows pre-set animation with the LEDs on the end of its arms. The parameter string to set an animation contains three parameters, divided by commas: the number of wished animation, its frequency in hertz and its duration in seconds. The frequency has to be a floating point value, represented as 32bit integer, e.g. `0.2` is represented by `-1102263091`.

| anim. | description |
|---|---|
| **0** | All LEDs are red, then all LEDs are green |
| **1** | All LEDs are green, then all LEDs are off |
| **2** | All LEDs are red, then all LEDs are off |
| **3** | All LEDs are orange, then all LEDs are off |
| **4** | All LEDs flash fast green/red/off |
| **5** | Front LEDs alternate fast yellow/off, rear LEDs are red |
| **6** | Front LEDs are green, read LEDs are red |
| **7** | All LEDs are red |
| **8** | All LEDs are green |
| **9** | Front left LED is red, all other are off. The light moves clockwise. |
| **10** | All LEDs are off |
| **11** | Left LEDs are off, right: rear LED is red, then rear LED is off and front LED is yellow. Then all LEDs are off |
| **12** | Right LEDs are off, left: rear LED is red, then rear LED is off and front LED is yellow. Then all LEDs are off |
| **13** | Rear LEDs are red, then are off and front LEDs are yellow. All LEDs are off |
| **14** | Front left LED is green, all other LEDs are red |
| **15** | Front right LED is green, all other LEDs are red |
| **16** | Rear right LED is green, all other LEDs are red |
| **17** | Rear left LED is green, all other LEDs are red |
| **18** | Left LEDs are green, right LEDs are red |
| **19** | Left LEDs are red, right LEDs are green |
| **20** | Front LEDs are green, rear LEDs are red, then all LEDs are off |

Table 5.12: Possible values for `leds:leds_anim` and drone's behaviour.

Example: With parameter string `"3,-1102263091,3"` all LEDs flash orange with 0.2 Hz for three seconds.
This setting can also be done by the command `led()`, described in section 3.8.7 on page 45.

**syslog:output** | **syslog:max_size** | **syslog:nb_files** *Std: read only*

*For Parrot's internal debugging.*

**pic:ultrasound_freq** *Std: read/write*

Frequency of the ultrasound for altitude measurement.

| | |
|---|---|
| 7 : | 22.22 kHz |
| 8 : | 25.00 kHz (Default) |

Table 5.13: Values to change ultrasound frequency.

**pic:ultrasound_watchdog** *Std: read/write*

*For Parrot's internal debugging, do not modify.*

**pic:pic_version** *Std: read only*

Software-version of the drone's Nav-board.

## 5.6 Custom

**custom:application_id** *Std: read/write*

Shows the current application ID in eight hexadecimal digits, stores application settings or creates them if they do not exist. To delete a set of settings, negate the ID by putting a minus (-) in front. By ID ¨-all¨ you delete all saved entries. This setting can also be done by command `setConfigApplicationID()`, described in section 3.5.6 on page 33; or set pre-set application-, profile- and session-IDs by command `setConfigAllID()` at once, described in section 3.5.7 on page 33.

**custom:application_desc** *App: read/write*

The description of the current application.

**custom:profile_id** *Ses: read/write*

Shows the current profile/user ID in eight hexadecimal digits, stores application settings or creates them if they do not exist. To delete a set of settings, negate the ID by putting a minus (-) in front. By ID ¨-all¨ you delete all saved entries. This setting can also be done by command `setConfigUserID()`, described in section 3.5.6 on page 32; or set pre-set application-, profile- and session-IDs by command `setConfigAllID()` at once, described in section 3.5.7 on page 33.

**custom:profile_desc** *Usr: read/write*

The description of the current profile/user.

**custom:session_id** *Ses: read only*

Shows the current session ID in eight hexadecimal digits, stores application settings or creates them if they do not exist. To delete a set of settings, negate the ID by putting a minus (-) in front. By ID ¨-all¨ you delete all saved entries. This setting can also be done by command `setConfigSessionID()`, described in section 3.5.6 on page 32; or set pre-set application-, profile- and session-IDs by command `setConfigAllID()` at once, described in section 3.5.7 on page 33.

**custom:session_desc** *Ses: read/write*

The description of the current session.

## 5.7 GPS, Network and Userbox

**gps:latitude** *Std: read/write*

The drone's latitude (north/south), set by GPS controlling device and used for media tagging and userbox recording. The value of latitude is a double precision floating point number represented as 64bit integer, also when set. So the value `50.130862` has to be represented by `4632252108916466947`.
Note: The GPS receiver is an extra device, connected to the drone's USB-port.

**gps:longitude** *Std: read/write*

The drone's longitude (east/west), set by GPS controlling device, used for media tagging and userbox recording. The value of longitude is a double precision floating point number represented as 64bit integer, also when set. So the value `8.455938` has to be represented by `4620949887957991902`.
Note: The GPS receiver is an extra device, connected to the drone's USB-port.

**gps:altitude** *Std: read/write*

The drone's altitude, set by GPS controlling device and used for media tagging and userbox recording. The value of altitude is a double precision floating point number represented as 64bit, also when set. So the value `189.8` has to be represented by `4640882010386700698`.
Note: The GPS receiver is an extra device, connected to the drone's USB-port.

**network:ssid_single_player** *Std: read/write*

The drone's WiFi-SSID. A change will apply after the drone's next reboot.

**network:ssid_multi_player** *Std: read/write*

*Reserved for further usage.*

**network:owner_mac** *Std: read/write*

Shows the MAC-address of the client connected to the drone.
Set value to `00:00:00:00:00:00` to unpair.

**network:wifi_rate** *Std: read/write*

*For Parrot's internal debugging, do not modify.*

**network:wifi_mode** *Std: read/write*

Represents the connection mode of the drone's WiFi-subsystem.

| | |
|---|---|
| 0 : | The drone is connectable as a WiFi-access-point (Default) |
| 1 : | The drone is connectable in Ad-Hoc-modus |
| 2 : | WiFi is in client-mode and the drone connects to an existing access point |

Table 5.14: Values to change the drone's WiFi-mode.

Changes are not suggested for multi-configurations.

**userbox:userbox_cmd** *Ses: read/write*

This Option gives the possibility to save the drone's GPS.

| | | |
|---|---|---|
| 0 | Stop | |
| 1 | Cancel | |
| 2 | Start | current date [date] |
| 3 | Photo | delay in sec [uint], number of pictures [uint] and current date [date] |

Table 5.15: Tag detection enabling values.

Extra parameters should be seperated by a comma, e.g. `2,20141222_130600`.
By *starting* a recording, the drone creates the <tempdir>-folder and a binary-file named `userbox_`<timestamp> which contains the GPS-positions. When taking a *photo* the shots are made by the front camera and stored in <tempdir>, named ¨`picture_`<date>`.jpg`¨.
If a recording got *canceled*, all recordings stop and <tempdir>-folder is deleted.
By *stopping* a recording the <tempdir>-folder is renamed to ¨`/data/video/boxes/flight_`<date>`/`¨.
Use FTP to download the recorded positions and pictures from the drone.

<timespamp> represents the time since the last boot of the drone.

<date> has (<...>) to have the format: YYYYMMDD_hhmmss

<tempdir> represents following directory in the drone's filesystem:
¨`/data/video/boxes/tmp_flight_`<date>`/`¨

Note: The GPS receiver is an extra device and is connected to the drone's USB-port.

# A List of configuration and Tags

Example listing of Ar.Drone 2.0 configuration

| Name | Value |
|---|---|
| general:num_version_config | 1 |
| general:num_version_mb | 33 |
| general:num_version_soft | 2.3.3 |
| general:drone_serial | PS721xxxxxxxxxxxx |
| general:soft_build_date | 2012-11-26 12:16 |
| general:motor1_soft | 1.43 |
| general:motor1_hard | 5.0 |
| general:motor1_supplier | 1.1 |
| general:motor2_soft | 1.43 |
| general:motor2_hard | 5.0 |
| general:motor2_supplier | 1.1 |
| general:motor3_soft | 1.43 |
| general:motor3_hard | 5.0 |
| general:motor3_supplier | 1.1 |
| general:motor4_soft | 1.43 |
| general:motor4_hard | 5.0 |
| general:motor4_supplier | 1.1 |
| general:ardrone_name | My ARDrone |
| general:flying_time | 28109 |
| general:navdata_demo | TRUE |
| general:com_watchdog | 2 |
| general:video_enable | FALSE |
| general:vision_enable | TRUE |
| general:vbat_min | 9000 |
| control:accs_offset | -3.9962664e+03 4.1537573e+03 4.1192974e+03 |
| control:accs_gains | 1.9310879e+00 6.4174724e-03 -1.7682407e-02 -1.5522186e-03 -1.9930630e+00 -2.9528014e-02 -4.4175964e-02 7.7207056e-03 -1.9653542e+00 |
| control:gyros_offset | 3.7773750e+01 -9.2787504e+00 1.1687500e+00 |
| control:gyros_gains | 1.0552111e-03 -1.0667081e-03 -1.0715150e-03 |
| control:gyros110_offset | 1.6625000e+03 1.6625000e+03 |
| control:gyros110_gains | 1.5271631e-03 -1.5271631e-03 |
| control:magneto_offset | 2.9589743e+02 -1.1218007e+02 -6.8020508e+02 |
| control:magneto_radius | 1.7466907e+02 |

...

...

| Name | Value |
|---|---|
| control:gyro_offset_thr_x | 4.0000000e+00 |
| control:gyro_offset_thr_y | 4.0000000e+00 |
| control:gyro_offset_thr_z | 5.0000000e-01 |
| control:pwm_ref_gyros | 500 |
| control:osctun_value | 63 |
| control:osctun_test | TRUE |
| control:altitude_max | 20000 |
| control:altitude_min | 50 |
| control:outdoor | FALSE |
| control:flight_without_shell | FALSE |
| control:autonomous_flight | TRUE |
| control:flight_anim | 0,0 |
| network:ssid_single_player | ardrone2_xxxxxx |
| network:ssid_multi_player | ardrone2_xxxxxx |
| network:wifi_mode | 0 |
| network:wifi_rate | 0 |
| network:owner_mac | 00:00:00:00:00:00 |
| pic:ultrasound_freq | 8 |
| pic:ultrasound_watchdog | 3 |
| pic:pic_version | 184877090 |
| video:camif_fps | 30 |
| video:camif_buffers | 2 |
| video:num_trackers | 12 |
| video:video_storage_space | 15360 |
| video:video_on_usb | FALSE |
| video:video_file_index | 2 |
| leds:leds_anim | 0,0,0 |
| detect:enemy_colors | 1 |
| detect:enemy_without_shell | 1 |
| syslog:output | 7 |
| syslog:max_size | 102400 |
| syslog:nb_files | 5 |
| general:localtime | 0 |
| general:navdata_options | 65537 |
| control:control_level | 0 |
| video:bitrate | 1000 |
| video:bitrate_ctrl_mode | 0 |
| video:bitrate_storage | 4000 |
| custom:application_desc | Default application configuration |
| control:euler_angle_max | 2.0943999e-01 |
| control:control_iphone_tilt | 3.4906584e-01 |
| control:control_vz_max | 7.0000000e+02 |
| control:control_yaw | 1.7453290e+00 |

...

...

| Name | Value |
|---|---|
| control:manual_trim | FALSE |
| control:indoor_euler_angle_max | 2.0943999e-01 |
| control:indoor_control_vz_max | 7.0000000e+02 |
| control:indoor_control_yaw | 1.7453290e+00 |
| control:outdoor_euler_angle_max | 3.4906584e-01 |
| control:outdoor_control_vz_max | 1.0000000e+03 |
| control:outdoor_control_yaw | 3.4906585e+00 |
| custom:profile_desc | Default profile configuration |
| control:flying_mode | 0 |
| control:hovering_range | 1000 |
| video:codec_fps | 30 |
| video:video_codec | 129 |
| video:video_slices | 0 |
| video:video_live_socket | 0 |
| video:max_bitrate | 1000 |
| video:video_channel | 0 |
| detect:groundstripe_colors | 16 |
| detect:detect_type | 3 |
| detect:detections_select_h | 0 |
| detect:detections_select_v_hsync | 0 |
| detect:detections_select_v | 0 |
| userbox:userbox_cmd | 0 |
| gps:latitude | 5.0000000000000000e+02 |
| gps:longitude | 5.0000000000000000e+02 |
| gps:altitude | 0.0000000000000000e+00 |
| custom:application_id | 00000000 |
| custom:profile_id | 00000000 |
| custom:session_id | 00000000 |
| custom:session_desc | 00000000 |

Listing A.1: Example listing of Ar.Drone 2.0 configuration

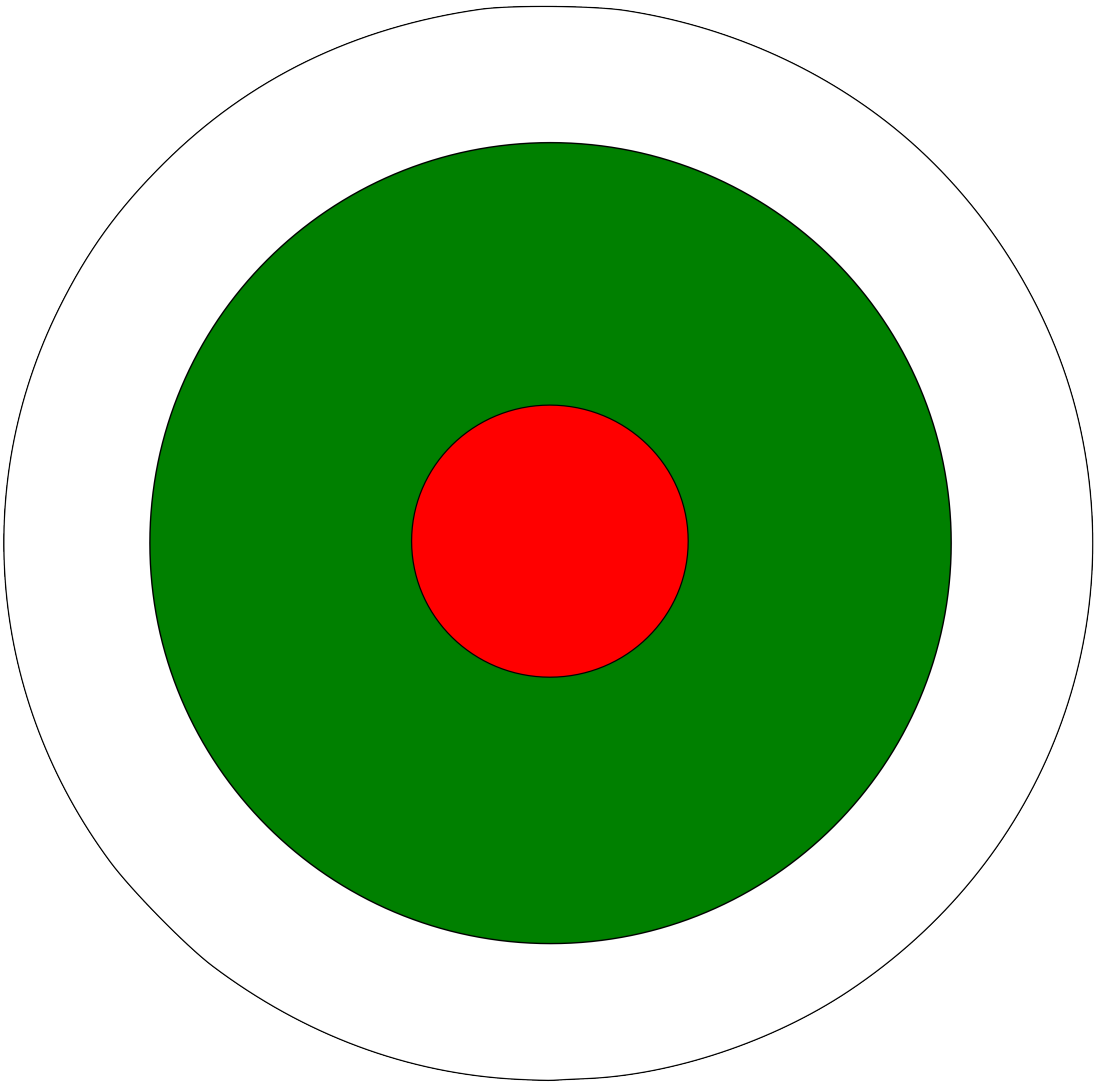


Figure A.1: Roundel

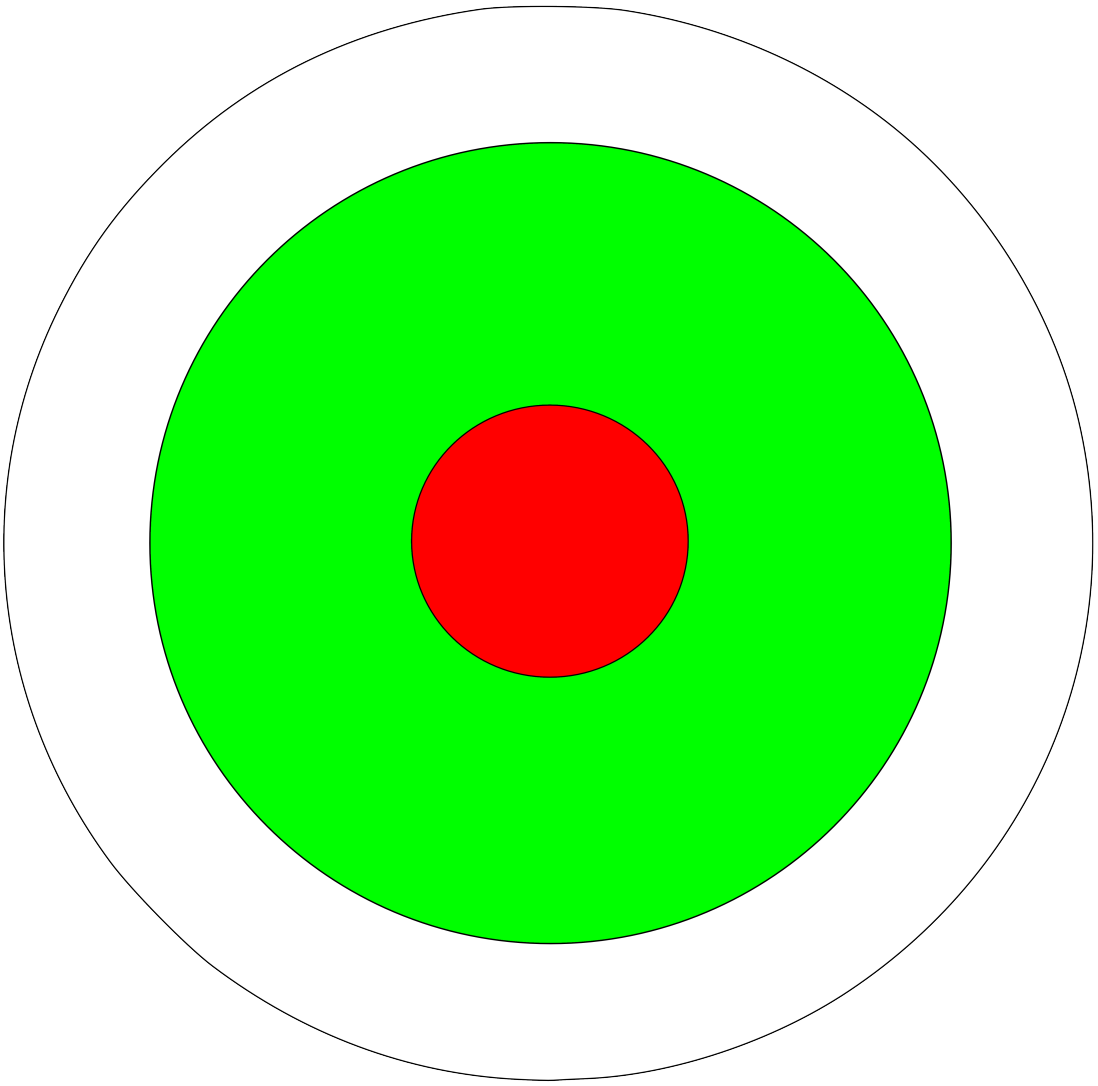


Figure A.2: Modified Roundel for a better detection

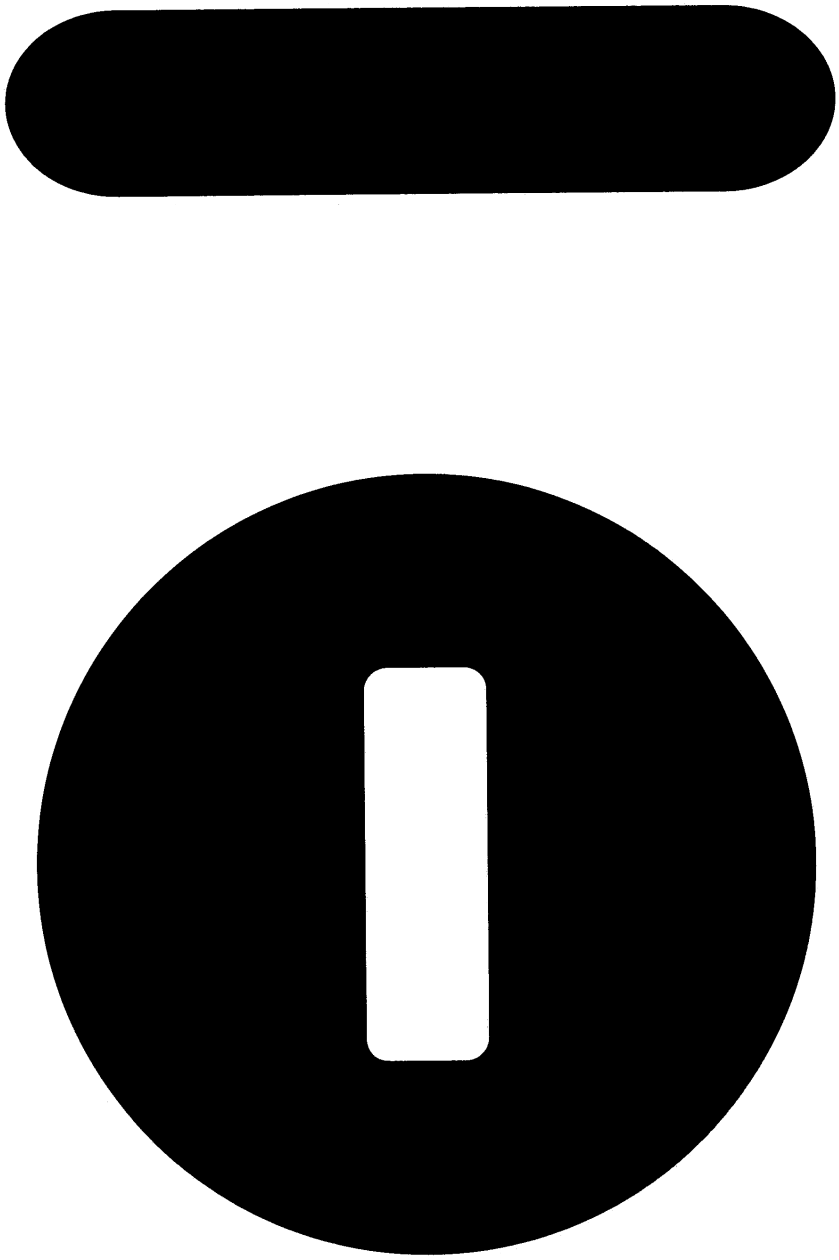


Figure A.3: Black and white Oriented Roundel