AutoFlight Documentation

Release dev-preview

Lukas Lao Beyer

August 23, 2015

Contents


AutoFlight
lbpclabs.com/autoflight
alpha

CHAPTER 1

Overview

AutoFlight's GUI is divided into a main panel used to display the video stream and two sidebars containing buttons (to connect to the AR.Drone, launch the scripting environment, etc.) and indicators to show the real time sensor data.

Note: If you have any questions, do not hesistate to contact me at `lukas (at) lbpclabs (dot) com`.

1.1 Basic Usage

Warning: By using AutoFlight you agree that I'm not responsible for any damage you might cause to your AR.Drone while using this program. This is an alpha version!

1.1.1 Connecting to the AR.Drone

To start flying your AR.Drone, make sure you are already connected to it via WiFi, just like you would connect to any other wireless router. Then, just click the button in the upper right corner ("Connect to AR.Drone"). In about one second the real time video stream and sensor data should appear.

Note: If it doesn't work, check you are not connected to any other wired or wireless networks. If it still doesn't work, contact me.

1.1.2 Head-Up Display

You can switch to a head-up display mode by pressing F5 or going into *View* → *Head-Up Display* option. Now the sensor data should be presented as an overlay on top of the video stream. Press F5 again to exit this mode.

1.1.3 Flying with a Joystick / Gamepad / Keyboard

Flying with a Joystick / Gamepad

You can configure your joystick over the *Edit* → *Controller Configuration* menu - it should be pretty straightforward. If you don't have a controller, you can fly with your keyboard, too. The next section shows how.

Note: My gamepad and joysticks work flawlessy, but if your do not, I've heard from users that MotioninJoy can solve

the problems.

Flying with the Keyboard

No configuration is needed. Just use the commands described in the table below.

AR.Drone Commands		AutoFlight Commands	
Take Off / Land	T	Take Picture	P
Switch camera (Front/Bottom)	V	Start/Stop recording video	R
Flip	2x F	Toggle HUD	F5
Emergency	2x Y	Start/Stop recording sensor data	N
Up | Rotate left | Down | Rotate right	I J K L		
Forward | Left | Backward | Right	W A S D		

1.2 Important warnings and known issues

This program is still in alpha, which means that it is not yet stable and complete enough to be considered production-quality software. Also, you should keep in mind that I can not take responsability for broken AR.Drones and you should use this program at your own risk. (However, should AutoFlight crash while flying, the AR.Drone would hover and descend to an altitude of 1m.)

The controller configuration is not checked automatically (yet), so you should confirm that you haven't assigned the same button/axis to multiple actions.

Some features like the image processor are not implemented yet but may be shown in the menus.

A few AutoScript functions are not implemented yet (see in-program AutoScript documentation).

The WiFi indicator is shown in the GUI but does not work.

There seem to be problems with the 3D map view not adjusting the view correctly (the virtual camera does not follow the drone indicator as it should).

If the main panel doesn't show the AutoFlight logo and you are unable to see the live video stream or the head-up display, you should make sure that you have at least OpenGL version 2. When running the program in VirtualBox (or other virtual environments) this might be a problem.

1.3 Miscellaneous

1.3.1 Photos/Video

Photos and recorded videos are saved in your home folder, under a new folder called AutoFlightSaves (e.g. in `C:\Users\your_username\AutoFlightSaves` on Windows 7).

To change the resolution of the video stream from 360P to 720P, you need to use the `--stream-resolution` command-line argument when starting the program:

```
/path/to/AutoFlight.exe --stream-resolution 720P
```

For this you will need to start the program from the command prompt/terminal.

1.3.2 AR.Drone configuration

Go into the *AR.Drone → Flight Settings* menu to change the on-board flight parameters of the drone (max. roll/pitch angles, max. height, etc.).

1.3.3 Hardware extensions

Hardware extensions to the AR.Drone 2.0 are not yet available.

1.4 Something does not work / I found a bug

Please create an issue on my JIRA project tracker. You just will have to sign up. Thank you!

CHAPTER 2

Welcome to AutoScript's documentation!

Contents:

2.1 Introduction to AutoScript

AutoScript is AutoFlight's scripting environment. You can write Python scripts to automatise the AR.Drone's behaviour using predefined functions to control movements, read sensors, etc. Here you will find some information on how to get started writing scripts and detailed descriptions of every available function.

Use the sidebar on the left to continue to the next topic, search, etc.

2.2 Tutorial

This tutorial will teach you some very basic Python and AutoScript functions so you can get started writing scripts right away. For a full Python tutorial visit http://docs.python.org/3/tutorial/.

2.2.1 AutoScript Scripting Environment

First, take a look at the user interface for writing scripts. In the center there is a text area with line numbering and syntax highliting. On the top there's a main menu containing options to save/open scripts and execute them as well as to open the documentation. There are shortcuts for these actions on the bottom bar. Note that on this bar there's also a toggle button to enable the "land on error" behavior, which will automatically make the AR.Drone land when an error in the script is occurrs. Also, apart from executing the scripts you can also simulate them. When simulating a script no commands will be sent to the drone. Instead, what would happen gets printed in the script output and the program needs the user to enter simulated sensor values. This feature is useful for testing your scripts and making sure everything works as expected.

Note: When editing a script you can always press `F1` to open the documentation. If there's an AutoScript command in the current line, the description of that command will be opened automatically. (Try it!)

2.2.2 Take Off and Land

Let's start with something very simple. Send a take off command to the drone, keep it in the air for some seconds, and land. To make the drone take off, the function

```
control.takeOff()
```

has to be called. It's that simple. This function, however, does only send the take off command without waiting for the drone to actually take off, so we'll have to wait a few seconds. Python has a function called `sleep()` which does exactly this. It can be found in the `time` module. To use functions in the time module, you have to import it. This is done at the beginning of the Python script, using the `import` command followed by the module you want to import, in this case

```
import time
```

Then you can use the sleep function with the time you want to sleep as parameter, in parentheses.

```
time.sleep(6)
```

The above function waits 6 seconds before letting the script contine. This will be enough time for the AR.Drone to take off. The rest of the time it will simply stay in the air. Now it's time to land the drone. The function for doing that is

```
control.land()
```

Finally, we'll show a message saying that the script worked. Python has a command that shows a message called `print`. Let's print our message

```
print("I just made the drone automatically take off and land!")
```

The finished script would look like this:

```
import time

control.takeOff()
time.sleep(6)
control.land()
print("I just made the drone automatically take off and land!")
```

Now it's time to run that script. Click run (The 4th button on the bottom bar, counting from the left). If there's a typo and the drone doesn't land because an error ocurred, you can just land the drone manually (switch to the main window and press `T`), check your code for typos and try to run it again.

If everything worked, congratulations! You ran your first AR.Drone-controlling-Python-script!

2.2.3 More coming soon

You can take a look at the official Python tutorial and all the available functions to experiment a bit!

2.3 Available AutoScript Functions

Here you will find a description of every available function you can use to control the AR.Drone, divided into the main modules (`control`, `sensors`, `util`).

2.3.1 AR.Drone control

`control.`**`backward_distance`**(*speed*, *centimeters*)

Moves the drone backward, and stops it after the specified distance has been traveled. Shortcut for `move_distance(0, speed, 0, 0, centimeters)`

Parameters

- **speed** (*float*) – The speed at which the drone should be flying. See `move()`.
- **centimeters** (*float*) – The distance to be traveled, in centimeters.

Returns `True` if the command could be send successfully, `False` otherwise.

`control.`**`backward_time`**(*speed*, *millis*)
Moves the drone backward, and stops it after the specified amount of time has elapsed. Shortcut for `move_time(0, speed, 0, 0, millis)`

Parameters

- **speed** (*float*) – The speed at which the drone should be flying. See `move()`.
- **millis** (*int*) – The time to wait before stoping the drone, in milliseconds.

Returns `True` if the command could be send successfully, `False` otherwise.

`control.`**`backward`**(*speed*)
Moves the drone backward. Shortcut for `move(0, speed, 0, 0)`. To stop it, call the `hover()` command.

Parameters **speed** (*float*) – The speed at which the drone should be flying. See `move()`.

Returns `True` if the command could be send successfully, `False` otherwise.

`control.`**`down_distance`**(*speed*, *centimeters*)
Moves the drone down, and stops it after the specified distance has been traveled. Shortcut for `move_distance(0, 0, -speed, 0, centimeters)`

Warning: Not implemented yet! Do not use.

Parameters

- **speed** (*float*) – The speed at which the drone should be flying. See `move()`.
- **centimeters** (*float*) – The distance to be traveled, in centimeters.

Returns `True` if the command could be send successfully, `False` otherwise.

`control.`**`down_time`**(*speed*, *millis*)
Moves the drone down, and stops it after the specified amount of time has elapsed. Shortcut for `move_time(0, 0, 0, -speed, millis)`

Parameters

- **speed** (*float*) – The speed at which the drone should be flying. See `move()`.
- **millis** (*int*) – The time to wait before stoping the drone, in milliseconds.

Returns `True` if the command could be send successfully, `False` otherwise.

`control.`**`down`**(*speed*)
Moves the drone backward. Shortcut for `move(0, 0, 0, -speed)`. To stop it, call the `hover()` command.

Parameters **speed** (*float*) – The speed at which the drone should be flying. See `move()`.

Returns `True` if the command could be send successfully, `False` otherwise.

`control.`**`flip`**(*direction*)
Sends the flip command to the AR.Drone. Only works with 2.0 drones.

Returns `True` if the command could be send successfully, `False` otherwise.

control.**forward_distance**(*speed*, *centimeters*)
Moves the drone forward, and stops it after the specified distance has been traveled. Shortcut for `move_distance(0, -speed, 0, 0, centimeters)`

Parameters

- **speed** (*float*) – The speed at which the drone should be flying. See `move()`.
- **centimeters** (*float*) – The distance to be traveled, in centimeters.

Returns `True` if the command could be send successfully, `False` otherwise.

control.**forward_time**(*speed*, *millis*)
Moves the drone forward, and stops it after the specified amount of time has elapsed. Shortcut for `move_time(0, -speed, 0, 0, millis)`

Parameters

- **speed** (*float*) – The speed at which the drone should be flying. See `move()`.
- **millis** (*int*) – The time to wait before stoping the drone, in milliseconds.

Returns `True` if the command could be send successfully, `False` otherwise.

control.**forward**(*speed*)
Moves the drone forward. Shortcut for `move(0, -speed, 0, 0)`. To stop it, call the `hover()` command.

Parameters **speed** (*float*) – The speed at which the drone should be flying. See `move()`.

Returns `True` if the command could be send successfully, `False` otherwise.

control.**hover**()
Hovers the drone, so it tries to stay at a fixed position. Equivalent to calling `move(0, 0, 0, 0)`.

Returns `True` if the command could be send successfully, `False` otherwise.

control.**land**()
Sends a land command to the drone.

Returns `True` if the command could be send successfully, `False` otherwise.

control.**left_distance**(*speed*, *centimeters*)
Moves the drone left, and stops it after the specified distance has been traveled. Shortcut for `move_distance(-speed, 0, 0, 0, centimeters)`

Parameters

- **speed** (*float*) – The speed at which the drone should be flying. See `move()`.
- **centimeters** (*float*) – The distance to be traveled, in centimeters.

Returns `True` if the command could be send successfully, `False` otherwise.

control.**left_time**(*speed*, *millis*)
Moves the drone left, and stops it after the specified amount of time has elapsed. Shortcut for `move_distance(-speed, 0, 0, 0, millis)`

Parameters

- **speed** (*float*) – The speed at which the drone should be flying. See `move()`.
- **millis** (*int*) – The time to wait before stoping the drone, in milliseconds.

Returns `True` if the command could be send successfully, `False` otherwise.

`control.`**`left`**(*speed*)
Moves the drone left. Shortcut for `move_distance(-speed, 0, 0)`. To stop it, call the `hover()` command.

Parameters **`speed`** (*float*) – The speed at which the drone should be flying. See `move()`.

Returns `True` if the command could be send successfully, `False` otherwise.

`control.`**`move_distance`**(*phi*, *theta*, *gaz*, *yaw*, *centimeters*)
Moves the drone as in `move()`, but stops it automatically after the specified distance has been traveled.

Warning: If the vertical camera of your AR.Drone does not work or data is somehow not sent back correctly, this can be dangerous. This function uses speed to calculate the traveled distance, and this speed data is computed by analyzing the vertical cameras pictures. **Will** be problematic if what the drone is flying over has no distinguishable features!

Parameters

- **`phi`** (*float*) – See `move()`.
- **`theta`** (*float*) – See `move()`.
- **`gaz`** (*float*) – See `move()`.
- **`yaw`** (*float*) – See `move()`.
- **`centimeters`** (*float*) – The distance to be traveled, in centimeters.

Returns `True` if the command could be send successfully, `False` otherwise.

`control.`**`move_time`**(*phi*, *theta*, *gaz*, *yaw*, *millis*)
Moves the drone as in `move()`, but stops it automatically after the specified amount of time.

Parameters

- **`phi`** (*float*) – See `move()`.
- **`theta`** (*float*) – See `move()`.
- **`gaz`** (*float*) – See `move()`.
- **`yaw`** (*float*) – See `move()`.
- **`millis`** (*int*) – The time to wait before stoping the drone, in milliseconds.

Returns `True` if the command could be send successfully, `False` otherwise.

`control.`**`move`**(*phi*, *theta*, *gaz*, *yaw*)
Moves the drone. The parameters are fractions of the maximum allowed angle, and have to be in the range from -1.0 (corresponding to the maximum tilt into one direction) to 1.0 (corresponding to the maximum tilt into the other direction).

Warning: This function will cause the drone to move with the specified parameters for an infinite amount of time. You will need to call the `hover()` command to stop it.

Parameters

- **`phi`** (*float*) – Roll angle (-1.0: full angle to left hand side; 1.0: full angle to right hand side)
- **`theta`** (*float*) – Pitch angle (**-1.0**: full speed in **forward** direction; **1.0**: full speed in **backward** direction)
- **`gaz`** (*float*) – Vertical speed (Exception as it is no angle. -1.0 would then be full speed down, 1.0 full speed up)

- **yaw** (*float*) – Yaw speed (Also no angle. -1.0 would be full speed in counterclockwise direction, 1.0 full speed in clockwise direction)

Returns `True` if the command could be send successfully, `False` otherwise.

`control.`**`right_distance`**(*speed*, *centimeters*)

Moves the drone right, and stops it after the specified distance has been traveled. Shortcut for `move_distance(speed, 0, 0, 0, centimeters)`

Parameters

- **speed** (*float*) – The speed at which the drone should be flying. See `move()`.
- **centimeters** (*float*) – The distance to be traveled, in centimeters.

Returns `True` if the command could be send successfully, `False` otherwise.

`control.`**`right_time`**(*speed*, *millis*)

Moves the drone right, and stops it after the specified amount of time has elapsed. Shortcut for `move_distance(speed, 0, 0, 0, millis)`

Parameters

- **speed** (*float*) – The speed at which the drone should be flying. See `move()`.
- **millis** (*int*) – The time to wait before stoping the drone, in milliseconds.

Returns `True` if the command could be send successfully, `False` otherwise.

`control.`**`right`**(*speed*)

Moves the drone right. Shortcut for `move_distance(speed, 0, 0, 0)`. To stop it, call the `hover()` command.

Parameters **speed** (*float*) – The speed at which the drone should be flying. See `move()`.

Returns `True` if the command could be send successfully, `False` otherwise.

`control.`**`rotate`**(*speed*, *degs*, *clockwise*)

Rotates the drone by the specified angle at the specified speed in the specified direction. Yes, it does exactly what you specified. And no, in the next sentence there will not be the word *specified* again.

> **Warning:** Somehow, this does not always work as expected. I don't yet know why, but I hope I'll find the error soon.

Parameters

- **speed** (*float*) – The rotation speed
- **degs** (*float*) – How many degrees the drone should rotate (>=0, please)
- **clockwise** – `True` for clockwise rotation, `False` for counterclockwise rotation

Returns `True` if the command could be send successfully, `False` otherwise.

`control.`**`takeOff`**`()`

Sends a take off command to the drone. This will only send the command and continue immediately, so you'll probably want to wait 4-6 seconds before calling any other functions.

Returns `True` if the command could be send successfully, `False` otherwise.

`control.`**`up_distance`**(*speed*, *centimeters*)

Moves the drone up, and stops it after the specified distance has been traveled. Shortcut for `move_distance(0, 0, speed, 0, centimeters)`

> **Warning:** Not implemented yet! Do not use.

Parameters

- **speed** (*float*) – The speed at which the drone should be flying. See `move()`.
- **centimeters** (*float*) – The distance to be traveled, in centimeters.

Returns `True` if the command could be send successfully, `False` otherwise.

`control.`**`up_time`**(*speed*, *millis*)
Moves the drone up, and stops it after the specified amount of time has elapsed. Shortcut for `move_time(0, 0, 0, speed, millis)`

Parameters

- **speed** (*float*) – The speed at which the drone should be flying. See `move()`.
- **millis** (*int*) – The time to wait before stoping the drone, in milliseconds.

Returns `True` if the command could be send successfully, `False` otherwise.

`control.`**`up`**(*speed*)
Moves the drone up. Shortcut for `move(0, 0, 0, speed)`. To stop it, call the `hover()` command.

Parameters **speed** (*float*) – The speed at which the drone should be flying. See `move()`.

Returns `True` if the command could be send successfully, `False` otherwise.

2.3.2 Sensor data retrieving

`sensors.`**`getAcceleration`**(*axis*)
Reads the accelerometer's value on the specified axis.

Parameters **axis** (*string*) – "X", "Y" or "Z"

Returns The acceleration on the specified axis, in g.

`sensors.`**`getAltitude`**()
Reads the drone's altitude.

Returns The altitude in m.

`sensors.`**`getBatteryLevel`**()
Reads the battery's level.

Returns The battery level in %.

`sensors.`**`getOrientation`**(*axis*)
Reads the gyroscope's value on the specified axis.

Parameters **axis** (*string*) – "YAW", "PITCH" or "ROLL"

Returns The angle on the specified axis, in degrees, as a value between -180 and +180.

`sensors.`**`getOrientation360`**(*axis*, *clockwise*)
Reads the gyroscope's value on the specified axis.

Parameters

- **axis** (*string*) – "YAW", "PITCH" or "ROLL"
- **clockwise** (*boolean*) – The direction in which to count, if `True` then in clockwise direction.

Returns The angle on the specified axis, in degrees, as a value between 0 and 360. E.g. if the drone is tilted 10 degrees to the right, the value would not be 10 but 100 degrees, if clockwise is `True`. Useful for measuring yaw angles.

sensors.**getLinearVelocity**(*axis*)
Reads the drone's speed on the specified axis.

Parameters **axis** (*string*) – "X", "Y", or "Z"

Returns The drone's speed on the specified axis, in m/s.

util.**isConnected**()
Checks if AutoFlight is receiving data from the drone.

Returns `True` if connected, `False` otherwise

util.**isFlying**()
Checks if the drone is flying.

Returns `True` if flying, `False` otherwise

2.3.3 Utilities and other

util.**flatTrim**()
Sends the "flat trim" command to the AR.Drone. This command calibrates the drone's inertial measurement unit, so it probably is nonsense if you call this while not on a flat surface.

Returns `True` if the command could be send successfully, `False` otherwise.

util.**calibrateMagnetometer**()
Sends the "magnetometer calibration" command to the AR.Drone. This command will make the drone rotate on its yaw axis a couple of times, so be careful when calling this.

Returns `True` if the command could be send successfully, `False` otherwise.

util.**startRecording**()
Starts recording video to the default location.

Returns `True` if command completed successfully, `False` otherwise.

util.**stopRecording**()
Stops recording video.

Returns `True` if command completed successfully, `False` otherwise.

util.**toggleRecording**()
Toggles video recording. (Starts recording if it's currently not recording, stops recording if it's currently recording.)

Returns `True` if command completed successfully, `False` otherwise.

util.**savePicture**(*path*)
Takes a picture and saves it as JPEG to the specified file.

Parameters **path** (*string*) – The filename of the picture.

Returns `True` if command completed successfully, `False` otherwise.

CHAPTER 3

Image Processing with AutoFlight

Contents:

3.1 Getting Started using OpenCV in AutoFlight

Note: To be able to use OpenCV from AutoScript, you will first have to install the image processing add-on. Get it here. Also, please make sure you are using at least AutoFlight version 0.2.

3.1.1 Installation

Extract the downloaded `AF_ImageProcessing.zip` file into the `Lib/site-packages/` folder in AutoFlight's root directory, which usually will be in the AppData folder. (To find it, you can simply open Explorer and type `%appdata%/AutoFlight/Lib/site-packages/`.) Now you are ready to use OpenCV from the AutoScript environment. You can test this by importing the `cv2` module and running the help function for it. To do this, open AutoScript and type:

```
import cv2
help(cv2)
```

If the image processing add-on was correctly installed, running the script should output a long list with available classes, functions, etc. You are now ready to use OpenCV.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

C

S

U