Activity 3

# Folder: algorithm\_3

## Quickstart - How it works

Line Detection

Algorithm03 is has a much better line detection algorithm. It finds two points on the line and so can calculate the slope of the line in the image (Recall y=mx+c). This is then used to calculate the actual direction of the line on the ground.

As with the previous algorithms, we also use the offset, but this time it is to measure how well placed the vehicle is over the line.

There are now two regions of interest, top and bottom of the image. With some trigonometry, we can work out the position of these two points **on the ground** and so work out the bearing of the line compared to the direction faced by the vehicle.

*Note the effect of perspective and the camera angle of 45 degrees*. The line is sloping to the *left* on the image, but the bearing is actually +9 degrees on the ground. We need to turn *right* to face the same direction as the line!

The offset is taken from the bottom ROI to indicate the vehicle position over the line.

Control

The code in detectline3.py returns the angle of the line on the camera image and the offset.

Firstly, the vehicle turns to face the same direction as the line. This means it will be following a path along the line, rather than just flying towards a point on the line.

Secondly, the vehicle moves sideways to be better positioned over the line.

In the example above, the vehicle needs to yaw 9 degrees clockwise and translate to the right (along the +y axis).

## What to Do

**IMPORTANT:**

**Test your code on the simulator every time before real flight.**

**Set the flightHeight variable to -1.0 for real flight, or a different value given to you. DO NOT leave it at   
-5.0 m used for the simulator!**

There are several variables you can change to improve performance further from algoithm02.

1. vMax. The maximum forward velocity along the x axis. As before, this is a fixed value you can set.
2. yawP. As before, the proportional constant used to correct the yaw. Remember this time the vehicle is trying to align itself directly with the line, not just fly towards it.
3. vyMax. The maximum ‘sideways’ velocity along the y axis. This assists in positioning the vehicle more effectively over the line, correcting for the offset.
4. vyP. The proportional constant used to correct the offset. As with yaw control, a proportional control is used for vy to prevent the vehicle overshooting the line and a possible oscillation.

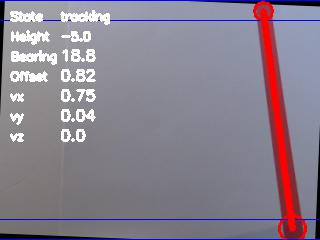
As before, start with the best value of yawP from Activity02 and use it for yawP in Activity03. Start with small values of vMax, vy and vyP and increase them gradually depending on your observations.

## Head up Display

To help analyse your data, follow3.py displays key variables directly on the camera image. This means that the line itself and the results of calculations can all be seen at the same time. It helps to answer questions such as:

* Is the vehicle turning quickly enough (increase yawP and/or reduce vMax)?
* How well is the vehicle staying over the line (increase vy and/or vyP)?

Let’s look at an example image:

Height. The value of -5m shows this is in the simulator. Real flight will be at about a value of -1 (so 1m above ground level).

Bearing: The line is actually 18 degrees clockwise from the direction the vehicle is facing. This looks ‘odd’, but is due to the perspective and angle of the camera. Its why we do the maths!

Offset: The 0.82 is a signal that the line beneath us is to the right.

Vy: The vehicle is attempting to correct for the offset by moving to the right at 4cm/s. Is that enough, perhaps?

## How to Run Code in the Simulator

1. Start the simulator if necessary and minimise the terminal, console and map windows.
2. Open the folder algorithm\_3
3. Select ‘Tools’ in the menu bar
4. Select ‘Open a terminal’ from the dropdown options
5. Use the up/down arrow keys to find the command ‘python follow3.py –connect “udp:127.0.0.1:5500” and press enter. Nb: The actual address in this line may vary from this.
6. The terminal will indicate a connection has been made and you can see the simulated motion on the map and console.
7. To close the simulation, select the folder terminal and Ctrl-C. Optionally type ‘mode rtl’ in the simulator terminal to have the drone return to land.

## Going Deeper

Whilst the techniques used in Activity 3 are an improvement, they could still be even better. However, you will need to examine and make modifications to the code itself in follow3.py to do this.

Here are some hints, but you may have your own ideas.

* You might have noticed that the speed of the vehicle actually increases around the bends. This is because the forward velocity vMax is fixed, but the additional component vy increases the resultant velocity. Is vy likely to be greater on the bends? Is there perhaps a better way of coordinating vMax (the x component of velocity) and vy (the y component)? Perhaps to keep the resultant velocity constant? You might like to consider how these two components were combined in Activity 2.
* Is there a way to fly faster on the straights? Don’t forget the vehicle has to slow down again for the bends!
* An important factor might also be the number of frames per second (fps) that are being analysed and acted on. This can be seen from the log file. The last digit in the first column is the current second and an entry is made every 10 frames. What is the frame rate (per second)? Consider that flying at 2m/s with a frame rate of 10 fps means that the line is only being ‘seen’ every 20cm. Is that enough? Annotating images and saving images is very intensive for the processor. Turning off these functions (once you don’t need them) may improve performance further.
* For the same reason, ‘if’ statements often used to turn on debugging statements also slow processing down. But be careful not to break things – it’s best to comment lines out rather than delete them!