Image Processing - Pattern recognition and drone controlling with OpenCV", Gabriel Hidasy, Science without Borders - CAPES
Under supervision of Dr Adam Schiffer, University of Pécs - PMMIK
June 13, 2015

1 The Problem

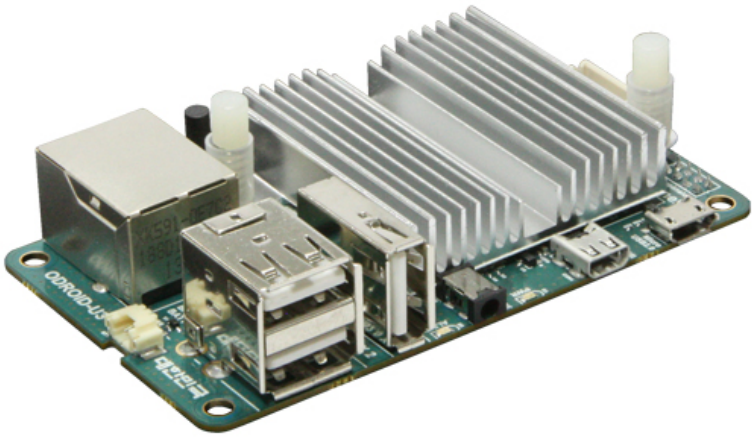
In the context of this project a drone is an unmanned aerial vehicle, capable of hover, particularly a Quad-copter.

Drones can be controlled remotely by a human or by an autonomous program,this project aims to produce a framework for controlling the drone from a computer and integrate it with a pattern recognition algorithm to follow the pattern in a room. This project does not aim to guarantee an stable flight but the modular nature of it makes it easy to integrate it in another program.

2 The equipment

The quad-copter used in this project is a Parrot AR-Drone 2.0, the drone has a front facing camera and is controlled by WiFi.

The controlling application is written in JavaScript and tested in a notebook and an ARM board (that has the added advantage of being light enough to be carried by the drone, enabling long range autonomous flights)



3 The solution

The solution is divided in 2 independent modules, a controlling application that communicates with the drone and the pattern matcher that finds the pattern and gives commands to the controller, this applications communicate through a web interface and do not need to be hosted in the same machine.

The controlling application is based in a NodeJS library that handles the low level communication, it exports a series of URLs that correspond to various functions of the drone.
The API is very high level and abstracts away details related to flight (for example, instead of dealing with the default coordinate system for aircraft's, [roll, pitch, yaw], which would require a very fine tuned propeller control (incidentally making quad-copters almost impossible to flight without computer aid). Commands are given based on the drone current coordinate system, in a [X,Y,Z] frame centered in the middle of the drone, with the X axis pointing towards the camera, the Z axis being the altitude and the Y axis from left to right.
The camera being positioned along the X axis is a great feature in controlling the drone both in First Person View flight and in controlling using images, as it permits a direct association between the position of objects in the image and the commands needed to move then to the center of the image (for example, if in the 640x360 image the point is in 100,300, we know the object is at left and below the drone) (0,0 being the top left corner and 640,360 the bottom right)

The functions available in the API are:

- takeoff
  Start propellers, and rise to about 1m.

- land
  Stop any movement, levels the drone and descend until close the ground, then stop propellers.

- up ([speed, optional, default 0.5],[time, optional, default 500ms])
  This command makes the drone climb, increasing Z

- down ([speed, optional, default 0.5],[time, optional, default 500ms])
  This command makes the drone descend, decreasing Z

- front ([speed, optional, default 0.5],[time, optional, default 500ms])
  This command makes the drone lean in its front, making it move forward, increasing X.

- back ([speed, optional, default 0.5],[time, optional, default 500ms])
  This command makes the drone lean in its back, making it move backward, decreasing X.

- left ([speed, optional, default 0.5],[time, optional, default 500ms])
  This command makes the drone lean on its lean on its left side, making it move to the left, decreasing Y

- right ([speed, optional, default 0.5],[time, optional, default 500ms])
  This command makes the drone lean on its lean on its right side, making it move to the right, increasing Y

- rotatel ([speed, optional, default 0.5],[time, optional, default 500ms])
  This command makes the drone rotate counter-clockwise over Z

- rotater ([speed, optional, default 0.5],[time, optional, default 500ms])
  This command makes the drone rotate clockwise over Z

- stop
  This command levels the drone immediately

- flip
  This command performs a flip, do not execute it near the ground, walls or other structures

- img.jpg
  This command returns a jpg snapshot from the drone

The functions are exported by a Web API available by accessing the URL IP:8002/functionName. Parameters can be passed by GET or POST, eg: 127.0.0.1:8002/up?speed=0.8&time=100

The speed parameter determines how much the drone will unbalance itself, being 1 the maximum supported value and 0 not tilting at all. Higher values lead to faster movements. The time parameter defines how long until the drone stabilizes itself again.

More then one application can access this API at a time, there are no security features implemented for now, but it would not be hard to add an authorization token.

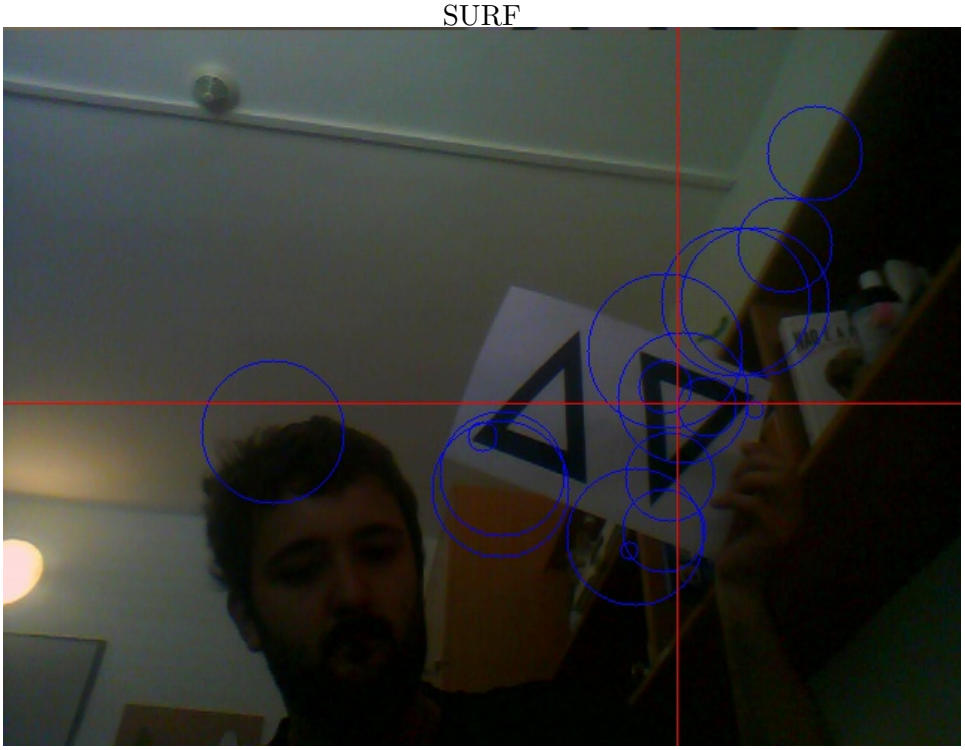
The pattern matching application is composed of:

- A SURF detector from OpenCV

```
img = image from drone camera, already binarized and filtered
template = image of the template
surf = cv2.SURF(400) #400 is an adequate threshold according to tests
skp, sd = surf.detectAndCompute(img,None)
#skp contains the image key-points, sd contains the descriptors
tkp, td = surf.detectAndCompute(template,None)
```

SURF is responsible for finding key-points in an image. A key-point is a point in the image with some robust feature. A robust feature is a detail of an image that can be detected even if its scaled, rotated, or slightly deformed.
The images bellow have their key-points marked, the first one has a red cross marking the position of the pattern in the image.

SURF



The size of the circle indicates the confidence in the match, this circles are already filtered.

The template is a pair of triangles as well marked corners are easy to match.

- A FLANN matcher from OpenCV
  FLANN is a smart algorithm to find matches in a set of data points, it was used in place of an exhaustive search because, despite not being as accurate, it is good enough and a lot faster, enabling the algorithm to run in a cheap ARM board with low power requirements.
  This piece of code below runs OpenCV FLANN on both key-point sets and find the best matches

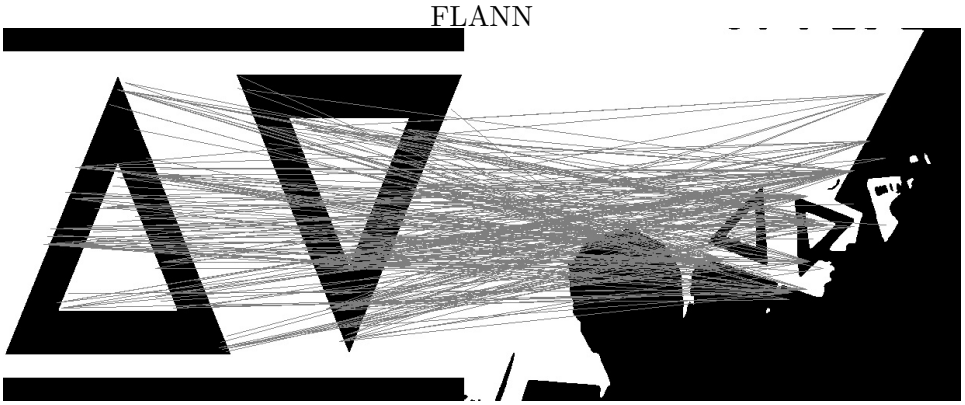
```
flann_params = dict(algorithm=1, trees=4)
flann = cv2.flann_Index(sd, flann_params)
idx, dist = flann.knnSearch(td, 1, params={})
del flann

dist = dist[:,0]/2500.0
dist = dist.reshape(-1,).tolist()
idx = idx.reshape(-1).tolist()
indices = range(len(dist))
indices.sort(key=lambda i: dist[i])
dist = [dist[i] for i in indices]
idx = [idx[i] for i in indices]
skp_final = []
for i, dis in itertools.izip(idx, dist):
    if dis < distance:
        skp_final.append(skp[i])

flann = cv2.flann_Index(td, flann_params)
idx, dist = flann.knnSearch(sd, 1, params={})
del flann

dist = dist[:,0]/2500.0
dist = dist.reshape(-1,).tolist()
idx = idx.reshape(-1).tolist()
indices = range(len(dist))
indices.sort(key=lambda i: dist[i])
dist = [dist[i] for i in indices]
idx = [idx[i] for i in indices]
tkp_final = []
for i, dis in itertools.izip(idx, dist):
    if dis < distance:
        tkp_final.append(tkp[i])

return skp_final, tkp_final
```



As seem in the image, SURF was applied to a binarized image, this was done as it reduce problems related to illumination.

- A filter: It is possible to see many imperfect matches, they are filtered out by iteractively removing the points farther away from the gravity center of the set.
- A decision algorithm: A simple algorithm to decide in witch direction the drone should move based on where the pattern is found (for up, down, left and right), and the size it appears to have (for front and back), at this iteration of the project the drone is set to move at only 30% of its speed and move for 0.5s at a time.
  The drone stays still when no pattern is detected, in a future iteration it could be useful to have the drone rotating or flying in circles, as it would increase the chance it would come across the pattern changing the point of view.

The application process a image in less then 50ms, but the rate is limited to 2Hz as its more then enough to control the drone and leave a lot of room for both improvements in this application and other applications running in the controller (as a flight stabilizer or a GPS tracker).

4 The Results

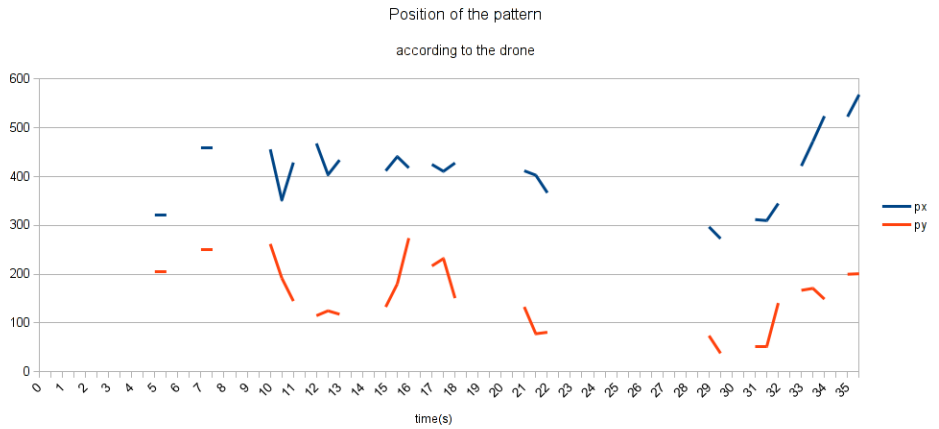
In the following 3 pages a log of a test flight is included, it consists of the output from the pattern matcher:

At this iteration the pattern matcher was not modulating the speed and time for each function. It was also not commanding the drone to go front or back as this section of the code is still not reliable enough for pure autonomous flight.

```
Time Time(s) px py Position R L U D F B
00.0 371 268 (371, 268, (360, 640)) 1 0 0 1 1 0
00.5
01.0 367 268 (367, 268, (360, 640)) 0 0 0 1 1 0
01.5
02.0
02.5
03.0
03.5
04.0
04.5
05.0 322 205 (322, 205, (360, 640)) 0 0 0 0 1 0
05.5 322 205 (322, 205, (360, 640))
06.0
06.5
07.0 459 251 (459, 251, (360, 640)) 1 0 0 1 1 0
07.5 459 251 (459, 251, (360, 640))
08.0
08.5
09.0 292 303 (292, 303, (360, 640)) 0 0 0 1 1 0
09.5
10.0 456 262 (456, 262, (360, 640)) 1 0 0 1 1 0
10.5 352 192 (352, 192, (360, 640)) 0 0 0 0 1 0
11.0 429 145 (429, 145, (360, 640)) 1 0 0 0 1 0
11.5
12.0 468 115 (468, 115, (360, 640)) 1 0 1 0 0 0
12.5 404 125 (404, 125, (360, 640)) 1 0 1 0 0 0
13.0 434 118 (434, 118, (360, 640)) 1 0 1 0 0 0
13.5
14.0
14.5
15.0 412 133 (412, 133, (360, 640)) 1 0 0 0 0 0
15.5 441 180 (441, 180, (360, 640)) 1 0 0 0 1 0
16.0 418 274 (418, 274, (360, 640)) 1 0 0 1 1 0
16.5
17.0 425 217 (425, 217, (360, 640)) 1 0 0 0 1 0
17.5 411 232 (411, 232, (360, 640)) 1 0 0 1 1 0
18.0 428 151 (428, 151, (360, 640)) 1 0 0 0 1 0
18.5
19.0 408 183 (408, 183, (360, 640)) 1 0 0 0 1 0
19.5
20.0 423 136 (423, 136, (360, 640)) 1 0 0 0 1 0
20.5
21.0 412 133 (412, 133, (360, 640)) 1 0 1 0 1 0
```

```
21.5 403  78 (403, 078, (360, 640)) 1 0 1 0 1 0
22.0 367  81 (367, 081, (360, 640)) 0 0 1 0 1 0
22.5
23.0
23.5
24.0 416  58 (416, 058, (360, 640)) 1 0 1 0 1 0
24.5
25.0 398  68 (398, 068, (360, 640)) 1 0 1 0 1 0
25.5
26.0
26.5
27.0 406  58 (406, 058, (360, 640)) 1 0 1 0 1 0
27.5
28.0
28.5
29.0 297  74 (297, 074, (360, 640)) 0 0 1 0 1 0
29.5 273  38 (273, 038, (360, 640)) 0 0 1 0 1 0
30.0
30.5
31.0 312  52 (312, 052, (360, 640)) 0 0 1 0 1 0
31.5 310  51 (310, 051, (360, 640)) 0 0 1 0 1 0
32.0 345 141 (345, 141, (360, 640)) 0 0 0 0 1 0
32.5
33.0 422 167 (422, 167, (360, 640)) 1 0 0 0 1 0
33.5 472 171 (472, 171, (360, 640)) 1 0 0 0 0 0
34.0 524 149 (524, 149, (360, 640)) 1 0 0 0 0 0
34.5
35.0 523 200 (523, 200, (360, 640)) 1 0 0 0 1 0
35.5 568 201 (568, 201, (360, 640)) 1 0 0 0 0 0
```

As seen in the logs the drone correctly reacts to the commands sent, further advances need to be made in modulating the force of the response based in the position and distance of the target pattern. The following graph is extracted from this log.



Example of the drone following a pattern