Assignment #2

This assignment is due on Saturday 19th 23:59:59 class starts via email to christian.wallraven+CVF2019@gmail.com.

Important: You need to name your file properly. If you do not adhere to this naming convention, I may not be able to properly grade you!!!

If you are done with the assignment, make one zip-file of the assignment1 directory and call this zip-file `STUDENTID1_STUDENTID2_STUDENTID3_A2.zip` (e.g.: `2016010000_2017010001_A2.zip` for a team consisting of two students or `2016010000_2017010001_2017010002_A2.zip` for a three-student team). The order of the IDs does not matter, but the correctness of the IDs does! **Please double-check that the name of the file is correct!!**

Also: Please make sure to comment the code, so that I can understand what it does. Uncommented code will reduce your points!

Finally: please read the assignment text carefully and make sure to implement EVERYTHING that is written here – if you forget to address something I wrote, this will also reduce your points! Precision is key ☺!

Part1 Implementing the Eigenvalue and Harris corner detectors (60 points):

In an iPython notebook `corners.ipynb`, import the following

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import time
import numpy as np # this is the only math-ish library,
you are allowed to import!! Do not import OpenCV!
```

Implement corner detectors based both on the minimum Eigenvalue and the Harris h measure. The corner detectors should be coded in a function called `cornerDetect(img, winSize=7, type=0),`

where the input arguments are

`img = image (grayscale!),`

`winSize = total size of window for summation in pixels,`

`type = {0 – Eigenvalue, 1 – Harris}`

The function should return the corner strength as a `numpy` array that has the same size as your input image.

The steps the function has to do are:

1) Filter `img` with Sobel kernel to obtain derivatives ix, iy
2) For each pixel in `img`, determine second moment matrix H summed up over the window

3) Depending on type, either determine the minimum eigenvalue OR the Harris corner measure and store in return array

Test your corner detector on this picture

https://users.fmrib.ox.ac.uk/~steve/susan/susan/img74.gif

[Read this with `matplotlib`! Depending on your installation of `matplotlib`, you will need to convert this to PNG first, since some installations can only read PNG. Again, do NOT use OpenCV!]

Test the picture with window sizes (3,5,7,9,11,15,21) for both types of corner measures and TIME the speed. For each window size, plot the output of the corner measures in a nice way. Plot the overall speed comparing the two measures in another plot

In the ipython notebook, answer the following questions:

1) At which window sizes can you detect all "meaningful" corners?
2) Which method is faster – Eigenvalue or Harris? By how much?

Bonus (10 points): Extend the `cornerDetect` function such that it works with color images – note that it should NOT create three images, but still only one corner value, but now taking into account all three color channels. Insert the code as a function `cornerDetectColor` and plot the results for this picture:
https://miro.medium.com/max/700/1*KvDEGIpfwdJtUFwB5acYEA.png

Part2 Implementing the LoG scale blob detector (60 points):

In an iPython notebook `logscale.ipynb`, import the following

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import time
import numpy as np # this is the only math-ish library,
you are allowed to import!! Do not import OpenCV!
```

Implement the Laplacian of Gaussian blob detector including scale space tracking. For this, first write a function `LoG(sigma=1, stencil=19)`, in which the first argument is the width of the Gaussian kernel and the second argument is the overall image size of the filter (here by default 19x19 pixels). The function should return a `numpy` array with the filter values. At each pixel (x,y), the LoG has the value of (note, that the values have to be CENTERED in the stencil window!!):

$$\mathrm{LoG}(x,y) = -\frac{1}{\pi\sigma^4}\left(1-\frac{x^2+y^2}{2\sigma^2}\right)e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Insert code that shows LoG filters for sigmas from 1 to 10 in steps of 1 with a stencil size of 19 pixels.

Insert another function called `filterImg(img,fil)`, which performs a convolution of `img` with `fil` and returns the filtered image as its return value. MAKE SURE that the return image has the SAME size as the input image `img`!!!

Insert another function called trackScale(img,sigmas,stencil,threshold) that

1) Creates `len(sigmas)` LoG filters with the sigmas in the array for the specified stencil size
2) Filters `img` with each LoG filter
3) For each pixel in the stack of filtered images, finds the maximum value and stores this in a return array called `totScale`
4) Plots the input picture `img` with overlays of the SCALED blobs that were found similar to the pictures in class on slides 79,84

Use the following picture (again, you may need to convert to PNG so you can read it in with `matplotlib` code)

https://img1.southernliving.timeinc.net/sites/default/files/styles/4_3_horizontal_inbody_900x506/public/image/2018/07/main/the_sunflower_fields_at_neuse_river_greenway_trail.jpg?itok=ZOlvAuIg&1532035249

Insert code that uses good parameters for sigmas, stencil size, and threshold so that you can find as many sunflower "blobs" as possible.

BONUS PART Implement the solution to the heat equation for images (40 points):

Follow the slides and create an ipython notebook called `heatImage.ipynb` that implements the numerical solution to the 2D heat equation using an image as the starting "temperature" L(i,j,n), where i,j index pixels and n indexes the time step.

The easiest way is to simply use the final equation on the final slide and loop over all pixels to create the next estimate of L(i,j,n+1). You have to be a little careful with how you treat the pixels at the edge…

Use the sunflower image from above and run 100 time steps, plotting the result every 10 time steps.