Homework 2

- Thomas Ehling A20432671
- CS 512 : Computer Vision

Review question

1.Noise and Filtering

(a)

The estimation of the SNR is done with the formulas :

$$SNR = Es/En = \sigma s^2/\sigma n^2 = ((1/n) \sum_{i,j} (I(i,j) - \hat{I})^2)/\sigma n^2$$

And :

$$SNR[dB] = 10log10Es/En$$

where :

- σn^2 is the variace for multiple frames of a static scene or the variance in a uniform image region.
- Es is the energy of the signal
- En is the energy of the noise

(b)

Gaussian noise is statistical noise having a probability density function equal to that of the normal distribution, which is also known as the Gaussian distribution. In other words, the values that the noise can take on are Gaussian-distributed.

Impulse Noise is a general term for single-pixel bright or dark spots that are not authentic imagery. This artifact can have several different causes, each with a slightly different appearance.

A **median filter** would handle better impulsive noise.

(c)

IF we apply a 3x3 convolution matrices to an image with the value of 2 in each pixel, then each pixel which isn't on the image borders, will have the value : 2 x 1 x 9 = 18

(d)

Rather than convolving with a 2D Gaussian, we can convolve with 1D Gaussian along rows and then columns.

(e)

For the convolution boundaries, there are 3 options :

- **Zero Pull** : fill all the pixels that are out of range with zeros
- **Mirorring** : Mirror the values of the closest pixel
- **Ignore** : keep the actual value of the pixels

(f)

A basic smooth filter would be : 1/9 $\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$

The sum of all entries will be **1**, as we divide each cell by the total value of the coeeficient.
Reason : The sum to be selected as it is so we can calculate a average value for the center pixel, according to the values of it neighbors

(g)

We can convolve with 1D Gaussian along rows and then columns.
It is more efficient that a convolution with a 2D Gaussian, but we can still implement it if we want.

(h)

Given a Gaussian filter with a σ=2, the size of the filter should be at least m=3σ=6, so a 1x6 filter.
A regular size is m = 5σ = 10, so a 1x10 filter would work nice.

(i)

A Gaussian Image pyramid is produced by convolving an image with gaussian filters of different sizes, and downsampled it by two at each operation.

So, For any level j of the pyramid, we have :
$Ij \rightarrow G \rightarrow$ Downsample by 2 $\rightarrow Ij - 1$

We produced such pyramids beacause doing a multiple scale analysis can produced a better final result. It is mainly used in Texture Synthesis.

(j)

A Laplacian pyramid is very similar to a Gaussian pyramid but saves the difference image of the blurred versions between each levels :
At every level, we apply the corresponding gaussian filter and downsample by two, but we alo save The Residues, that is the difference between the original imae and the final one that have been upsample by 2 and convoluted again.

Only the smallest level is not a difference image to enable reconstruction of the high resolution image using the difference images on higher levels. This technique is used in Image compression.

2. Edge detection

(a)

The purpose of detecting sharp changes in image brightness is to capture important events and changes in properties of the world. It can be shown that under rather general assumptions for an image formation model, discontinuities in image brightness are likely to correspond to:

- discontinuities in depth,
- discontinuities in surface orientation,
- changes in material properties and
- variations in scene illumination.

In the best scenario, applying a edge dector filter will allow us to detect the boundaries of an object, otherwise it will still reduce the amount of data to process.

Edge detection is one of the fundamental steps in image processing, image analysis, image pattern recognition, and computer vision techniques.

(b)

The edge detection steps are :
- 1) **Smooth** to reduce noise
- 2) **Enhance edges**, by emphasizes pixels where there is a significant change in local intensity values, usually with the gradient magnitude.
- 3) **Detect edges**
- 4) **Localize edges**, Filter the result of the detection, tolocalize edges.

(*c*)

There are two filters to compute the image gradient :
-1) **Forward differences** : that follow this equation

$$\sigma I(x,y)/\sigma x = (I(x+h,y) - I(x,y))/h = I(x+1,y) - I(x,y)$$

for Hence, h =1.
-2) **Central differences** : that follow this equation :

$$\sigma I(x,y)/\sigma x = (I(x+h,y) - I(x-h,y))/2h = I(x+1,y) - I(x-1,y)$$

for Hence, h =1.

An **image gradient** is a directional change in the intensity or color in an image. The gradient of the image is one of the fundamental building blocks in image processing.

Image gradients can be used to extract information from images. Each pixel of a gradient image measures the change in intensity of that same point in the original image, in a given direction. To get the full range of direction, gradient images in the x and y directions are computed.

(d)

A sober filter can be produced by doing the convolution between the smoothing matrix
and the derivative matrix.

Sobel's filter derivatives, Δx,S and Δy,S, respectively detect horizontal edges and vertical edges.

$$\Delta x,S = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} * \begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$\Delta y,S = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

(e)

The elements of a filter for more accurate derivative computation are :

- x-derivative : Ix = I * G'(x) * G(y)
- y-derivative : Iy = I * G'(y) * G(x)

Where, G(x) = $\exp(-x^2/ 2\sigma^2) = \exp(-x^2/ 8)$
and G(y) = $\exp(-y^2/ 2\sigma^2) = \exp(-y^2/ 8)$

with σ=2, the fiter size is 5.
x = y = [-2,-1,0,1,2]
G(x) = G(y) = [G(-2), G(-1), G(0), G(1), G(2)]

$$G = \begin{bmatrix} 0.678 & 0.535 & 0.606 & 0.535 & 0.367 \\ 0.535 & 0.779 & 0.882 & 0.778 & 0.535 \\ 0.606 & 0.882 & 1 & 0.882 & 0.606 \\ 0.535 & 0.779 & 0.882 & 0.779 & 0.535 \\ 0.368 & 0.535 & 0.606 & 0.535 & 0.368 \end{bmatrix}$$

(f)

For the first derivative, we should define threshold to detect a change (positive to negative slope or reverse) in the curve of the derivative function.Therefore, the localization is not very precise.

For the second derivative, the edge is very well localized since we can have his location by looking at the zero crossing of the function.

(g)

For σ=1 :

LoG(x, y)
= - $1/\pi\sigma^4$ [1-($(x^2+y^2)/2\sigma^2$)] $\exp(-((x^2+y^2)/2\sigma^2$
= - $1/\pi$ [1-($(x^2+y^2)/2$)] $\exp(-((x^2+y^2)/2$

Edge detection with LOG :

1. **Smoothing** : Smooth the image with a Gaussian filter with spread σ
2. **Gradient** : Compute gradient magnitude and direction at each pixel of the smoothed image
3. **Thresholding** : Threshold the gradient magnitude image such that strong edges are kept and noise is suppressed
4. **Non-maximum suppression (thinning)** : Zero out all pixels that are not the maximum along the direction of the gradient (look at 1 pixel on each side)

(h)

The Process of Canny edge detection algorithm can be broken down to 5 different steps:

1. Apply Gaussian filter to smooth the image in order to remove the noise
2. Find the intensity gradients of the image
3. Apply non-maximum suppression to get rid of spurious response to edge detection
4. Apply double threshold to determine potential edges
5. Track edge by hysteresis: Finalize the detection of edges by suppressing all the other edges that are weak and not connected to strong edges.

Canny edge detection is a technique to extract useful structural information from different vision objects and dramatically reduce the amount of data to be processed. It has been widely applied in various computer vision systems. Canny has found that the requirements for the application of edge detection on diverse vision systems are relatively similar. Thus, an edge detection solution to address these requirements can be implemented in a wide range of situations. The general criteria for edge detection include:
- Detection of edge with low error rate, which means that the detection should accurately catch as many edges shown in the image as possible
- The edge point detected from the operator should accurately localize on the center of the edge.
- A given edge in the image should only be marked once, and where possible, image noise should not create false edges.

To satisfy these requirements Canny used the calculus of variations – a technique which finds the function which optimizes a given functional. The optimal function in Canny's detector is described by the sum of four exponential terms, but it can be approximated by the first derivative of a Gaussian.

(i)

Non-maximum suppression is an edge thinning, it is applied to find "the largest" edge. After applying gradient calculation, the edge extracted from the gradient value is still quite blurred. The algorithm for each pixel in the gradient image is:

1. Compare the edge strength of the current pixel with the edge strength of the pixel in the positive and negative gradient directions.
2. If the edge strength of the current pixel is the largest compared to the other pixels in the mask with the same direction (i.e., the pixel that is pointing in the y-direction, it will be compared to the pixel above and below it in the vertical axis), the value will be preserved. Otherwise, the value will be suppressed.

Programming review

Abstract

This report discuss and evaluate the performance of the solution developed to answer the Assigment 1 statement.
With the library Opencv and the programming language python, a serie of operation will be computed on an selected image (from computer or camera).

Problem Statement

The image should be taken from computer file if a path is specified. If not, the image should be taken from the camera.
Whenever a key is entered, the image should be updated with the following operations :

- i : reload original image
- w : save the current modified image
- g : convert to grayscale with OpenCv
- G : convert to grayscale with a personnal implementation
- c : cycle through color channels
- s : convert to grayscale and smooth with openCV
- S : convert to grayscale and smooth with a personnal implementation
- d : downsample the image by 2
- D : downsample the image by 2 and smooth it
- x : convert to grayscale and apply x derivative filter
- y : convert to grayscale and apply y derivative filter
- m : convert to grayscale and show magnitude
- p : convert to greyscale and plot gradient vectors
- r : convert to greyscale and rotate
- h : display help

Proposed Solution

i : reload original image

A copy of the image is save as soon as the image as soon as the image is loaded.
When the key 'i' is entered, the current image is changed with this copy.

w : save the current modified image

The current image is saved under the name "out.png" with the function :

```
cv2.imwrite('out.png', img)
```

g : convert to grayscale with OpenCv

Before trynig to convert the image, it's necessary to check if it is'nt already in grey. To do so, the number of channels is checked and need to be different from 2 (grey). Then the image is changed with the opencv function :

```
if (len(img.shape) != 3):
 print("not enough channels to convert to grey")
else:
 img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

G : convert to grayscale with a personnal implementation

Same verification that before, but the convertion to grayscale is done with a personnal implementation.
A common matrices to convert an image to grey is composed with the following coefficients : 0.114, 0.587, 0.299. After defining the matrice, it is applied to the image :

```
coefficients = [0.114, 0.587, 0.299]
m_grey = np.array(coefficients).reshape((1, 3))
img = cv2.transform(img, m_grey)
```

c : cycle through color channels

To cycle through color channels,two prerequisites are needed. The image isn't in grayscale because the image dimension would be 2, and at the first iteration the three images for the three channels should be extracted, as it s impossible to get the green channels from the red channel image.
A variable index is defined to know what channel should be display :

```
#at the first iteration, init the variables
if(first_call_color):
 b, g, r = cv2.split(img)
 col_nul = r.copy()
 col_nul[:] = 0
 first_call_color = False

#select the channel to filter
img = cv2.merge((b, g, r))
if (index_color == 0):
  img = cv2.merge((b, col_nul, col_nul))
elif (index_color == 1):
  img = cv2.merge((col_nul, g, col_nul))
else:
  img = cv2.merge((col_nul, col_nul, r))

#update the index :
index_color = index_color + 1 if index_color < 2 else 0
```

s : convert to grayscale and smooth with openCV

The convertion to greyscale is done as explained before.
The amount of smoothing is selected according to the current value on the corresponding trackbar.
The smoothing itseld can be done with several filters. The filter that has been selected is the most basic one. (but any other, such a gaussian would work fine) :

```
x_s = cv2.getTrackbarPos("s", "CS512_HW2")
x_s = 2 * x_s + 1
img = cv2.blur(img, (x_s, x_s))
```

S : convert to grayscale and smooth with a personnal implementation

The convertion to greyscale is done as explained before.
The Trackbar for the personnal smoothing can take 4 values :

- 0 : defalut value
- 1 : use a 3x3 matrice
- 2 : use a 5x5 matrice
- 3 : use a 7x7 matrice

The pixels at the boundaries are ignored to add code clarity. (other possibilities would be zero padding or duplication of neighbors values).

The selected filter for this operation is a median filter.

For each pixel in the image ,we extrate the NxN matrice where this pixel is the center. Then we sum up the value of all the cells and divide by the number of them :

```
#the 3 user input correspond to these sizes:
#  1 = 3x3 matrice
#  2 = 5x5 matrice
#  3 = 7x7 matrice
x = 3 if x==1 else (5 if x==2 else 7)

#used a new image to store the covuluted pixels
img_smoothed = img.copy()

#convolution function to compute
for ind_x in range(x+2, img.shape[1]-x-2):
    for ind_y in range(x+2, img.shape[0]-x-2):
        mat_temp = img[ind_y-(x-1)/2:ind_y+(x-1)/2+1, ind_x-(x-1)/2:ind_x+(x-1)/2+1]
        img_smoothed[ind_y][ind_x] = np.sum(mat_temp/(x**2))

img = img.smoothed.copy()
```

d : downsample the image by 2

An openCV fbuilt-in function can redimensionnate an imageby downsample or upsample it. The parameters are the current width and height divided by 2 :

```
rows = img.shape[1] // 2
cols = img.shape[0] // 2
img = cv2.resize(img, (rows, cols), interpolation=cv2.INTER_NEAREST)
```

D : downsample the image by 2 and smooth it

This is just a combination od the command ‘d’ and ‘s’ explained before.

x : convert to grayscale and apply x derivative filter

The convertion to greyscale is done as explained before.
An openCv funciton exist to use a Sobel filter on a specify axis, the x axis is choosed as a parameter. Then

the values are normalised to 255 :

```
img = cv2.Sobel(img, cv2.CV_8U, 1, 0)
scale = np.max(img) / 255
img = (img / scale).astype(np.uint8)
```

y : convert to grayscale and apply y derivative filter

The convertion to greyscale is done as explained before.
An openCv funciton exist to use a Sobel filter on a specify axis, the y axis is choosed as a parameter. Then the values are normalised to 255 :

```
img = cv2.Sobel(img, cv2.CV_8U, 0, 1)
scale = np.max(img) / 255
img = (img / scale).astype(np.uint8)
```

m : convert to grayscale and show magnitude

The convertion to greyscale is done as explained before.
The magnitude is equal to the square root of the square values of the gradient in x and y. So we calculate the gradient as before, then compute the magnitude adn normalise the values to 255 :

```
dx =  cv2.Sobel(img, cv2.CV_64F, 1, 0)
dy =  cv2.Sobel(img, cv2.CV_64F, 0, 1)
img = np.sqrt(dx**2 + dy**2)
scale = np.max(img) / 255
img = (img / scale).astype(np.uint8)
```

p : convert to greyscale and plot gradient vectors

The convertion to greyscale is done as explained before.

The trackbar value is the space in pixel between two gradient vectors.

For every every n pixels we extract a matrice, corresponding at the n pixels around the selcted one. In order to plot the gradient vectors, the gradient in x and in y are computed a newpoint is compute with these values.

Finally, lines are drawn between the selected pixels and there computed values, representing the gradient vectors of these pixels.

```
for ind_x in xrange(0, img_grad.shape[1], x):
    min_x = ind_x
    max_x = ind_x + x
    for ind_y in xrange(0, img_grad.shape[0], x):
        min_y = ind_y
        max_y = ind_y + x

        #draw the square for the matrices
 cv2.line(img_grad, (min_x, min_y), (min_x, max_y), (0, 125, 125), thickness=1, lineType=8, sh
 cv2.line(img_grad, (min_x, max_y), (max_x, max_y), (0, 125, 125), thickness=1, lineType=8, sh
 cv2.line(img_grad, (max_x, max_y), (max_x, min_y), (0, 125, 125), thickness=1, lineType=8, sh
 cv2.line(img_grad, (max_x, min_y), (min_x, min_y), (0, 125, 125), thickness=1, lineType=8, sh

        #get the corresponding matrice
 mat_temp = img_grad[min_y:max_y, min_x:max_x]

        #calculate the center of the matrice
 center = (min_x + x // 2, min_y + x // 2)

        #make sure the matrice is complete
 if(mat_temp.shape[0] != 0 and mat_temp.shape[1] != 0):

            #calculate of the gradient in y axis
     grad_y = mat_temp.copy()
            grad_y = cv2.Sobel(grad_y, cv2.CV_8U, 0, 1)
            g_y = np.mean(grad_y) / 255 * 100

            #calculate of the gradient in x axis
     grad_x = mat_temp.copy()
            grad_x = cv2.Sobel(grad_x, cv2.CV_8U, 1, 0)
            g_x = np.mean(grad_x) / 255 * 100

     #calculate the end point of the vector, with the gradients
     dest = (min_x + x / 2 + int(g_x), min_y + x / 2 + int(g_y))

            #draw the line
     cv2.line(img_grad, center, dest, (255,0,0),  thickness=2, lineType=8, shift=0)

img = img_grad.copy()
```

r : convert to greyscale and rotate

The convertion to greyscale is done as explained before.

The rotation is made with the OpenCv functions “getRotationMatrix2D” and “warpAffine”, around the center of the image and coording to it dimension :

```
cols = img_rot.shape[0]
rows = img_rot.shape[1]
center = (rows / 2, cols / 2)
x_r = cv2.getTrackbarPos("r", "CS512_HW2")
mat_rot= cv2.getRotationMatrix2D(center, x_r, 1.0)
img_rotated = cv2.warpAffine(img_rot, mat_rot, (rows, cols))
img = img_rotated.copy()
```

h : display help

For displaying the help, An image have been created listing all the commands.
And this image is displayed in an another windown, so the user can keep using the program with the help on the side.

Implementation details

Folder requirement

As the help displayed when the key “h” is enterd is an image on the computer. The programm needs to be launch from a folder with the help image within it.

Selection of the image

- If the path to an image is given as a command line argument, the program will use it.
- Otherwise, the computer camera is turned on, and press any key to capture an image.

Get User Input

The OpenCv function “waitkey” is used to detect the user input.
However, As this function can return a different value or not for capital letters according to the computer, a general solution has been implemented :
- **

User enter the key “SHIFT” then the desired key letter for Capitals :
Example : for “S” : enter “SHIFT” THEN "s"

**

Used the trackbar :

The user can changed the trackbar without affecting the image if the mode is not selected.

When the key is entered, the mode is turned on and changing the trackbar value will change the image in real time.

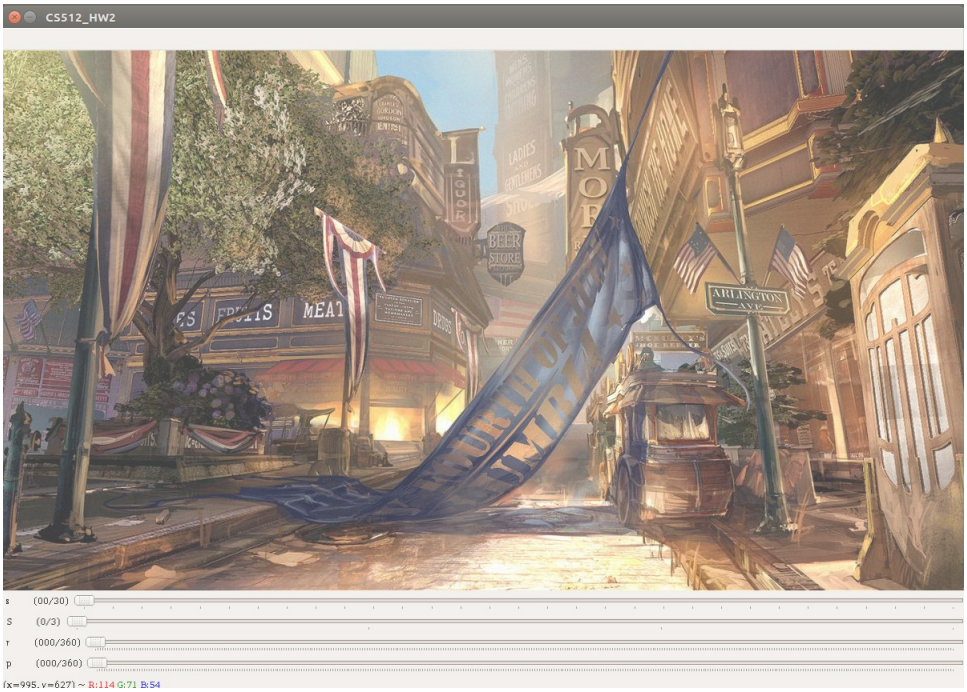
Example for smoothing with “s” :
- 1. User enter the key “s”
- 2. The mode smooth is triggered and the image is updated with the current value on the trackbar
- 3. Whenever the value on the trackbar is changed the image is updated accoringly
- 4. When any key is entered, the image is save with the current state

Result and Discussion

The operations can be combined, when performed one after the other.

Synthetic data

To analyze the results and discuss the performance, the output of the computation will by compute on the following model :



For the report to be more clear, only this model will be used here.
However, All the following functions have been tested on several images.

i : reload original image

The image is reloaded each time i is entered.
This has been check after eery other computation method.

w : save the current modified image

The current image is save under a file name “out.png”.
The image as been save and open after every computation and each time the

image was exactly at it was on the screen after the operations.

g : convert to grayscale with OpenCv

Command	Result
g - Grayscale with OpenCv	

Tests and results :

- **Visual verification** : the final image is composed by shade of grey
- **Image dimension** : If the function is called again, the program detect that the dimension are now 2 instead of 3 (r,g,b), and let the image as it already is. In addition to prevent a runtime error, this confirmed that the dimension has been changed
- **Computation time** :very fast, instant change

G : convert to grayscale with a personnal implementation

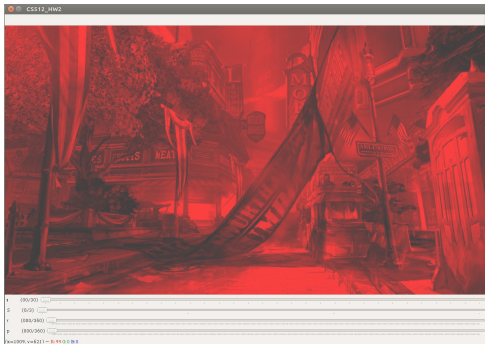
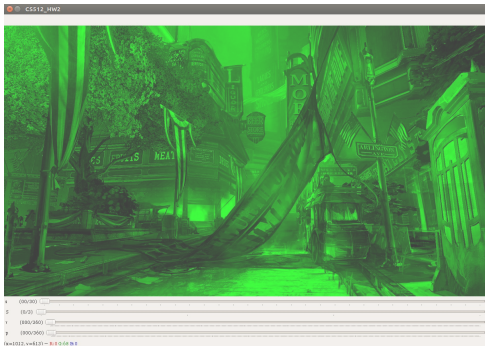
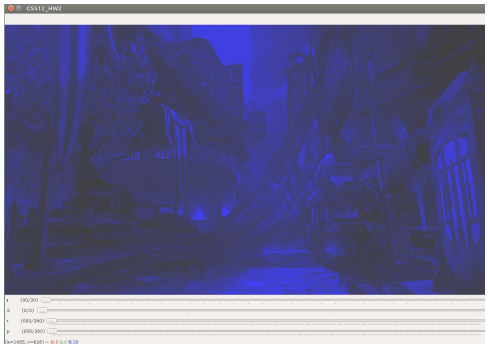
Command	Result
G - Grayscale with my code	

Tests and results :

- **Visual verification** : the final image is composed by shade of grey

- **Image dimension** : If the function is called again, the program detect that the dimension are now 2 instead of 3 (r,g,b), and let the image as it already is. In addition to prevent a runtime error, this confirmed that the dimension has been changed
- **Computation time** :very fast, instant change
- **Difference with opencv function** : No difference between both, as for computational time or result.

c : cycle through color channels

Command	Result
c - Cycle through color channels	
c - Cycle through color channels	
c - Cycle through color channels	

Tests and results :

- **Visual verification** : the final image is composed only by shades of red, green of blue.
- **Cycle** : whenever c is entered, another color is shown. If the three colors has been shown, the first one is shown again.
- **Computation time** :very fast, instant change

s : convert to grayscale and smooth with openCV

Command	Result
s - Grayscale and Smooth with OpenCv	

Same test and results for grayscale as before.

Test and results for smoothing :

- **Visual verification** : the final image is blurred, and the amount of smoothing is proportional to the value on the trackbar.
- **Trackbar input** Whenever the trackbar value is changed but the key haven't been enterd, nothing happened.
- **Trackbar input** If the key has been entered, the image will update in real-time with the trackbar value.
- **Computation time** : very fast, instant change.

S : convert to grayscale and smooth with a personnal implementation

Command	Result

Command	Result
S - Grayscale and Smooth with my code	

Same test and results for grayscale as before.

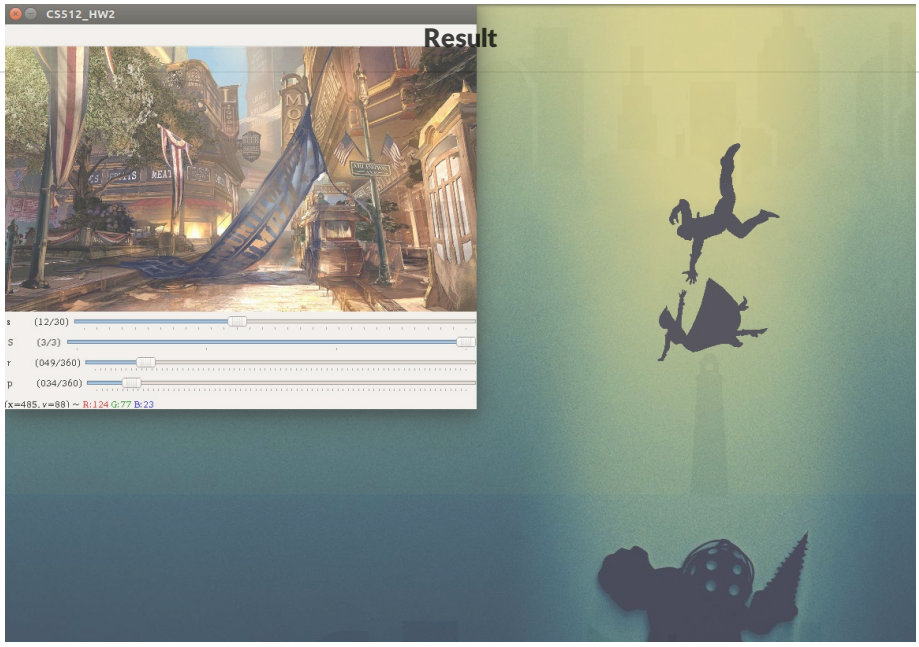
Test and results for smoothing :

- **Visual verification :** the final image is blurred, and the amount of smoothing is proportional to the value on the trackbar. However the smoothing is light and the choice to ignore the boundaries is clearly visible, especially for a 7x7 matrice.
- **Trackbar input** Whenever the trackbar value is changed but the key haven't been entered, nothing happened.
- **Trackbar input** If the key has been entered, the image will update in real-time with the trackbar value.
- **Computation time :** very slow, take a few seconds to be updated. The higher the input valueis, the bigger the filter matrices is, and the longer the computation time is.
- **Difference between OpenCv function :** Much slower, and the amount of blurring is limited and more difficult to notice.

Remarks : Because the computation time is very long, the function isn't optimise especially compared to the OpenCv function.
Possible solution : implement a multi-threading program to reduce computation time.

d : downsample the image by 2

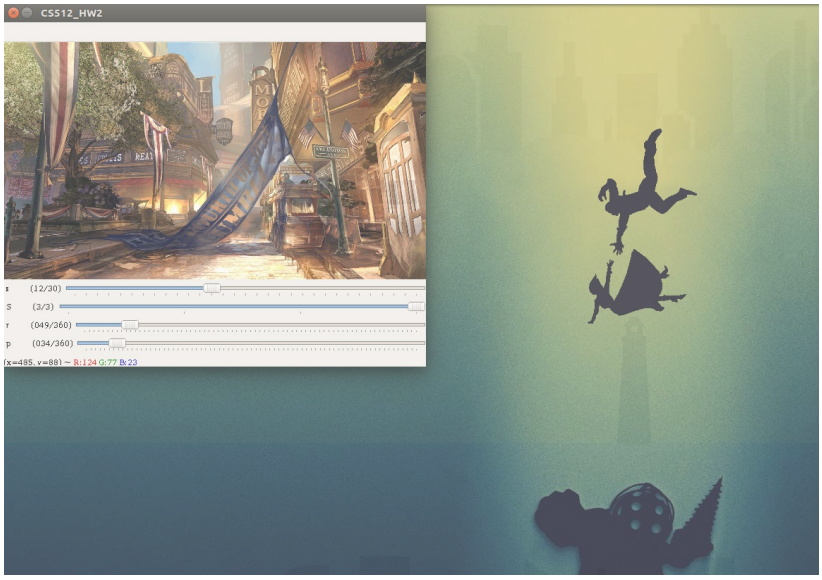
Command	Result

Command	Result
d - Downsample by 2	

Tests and results :

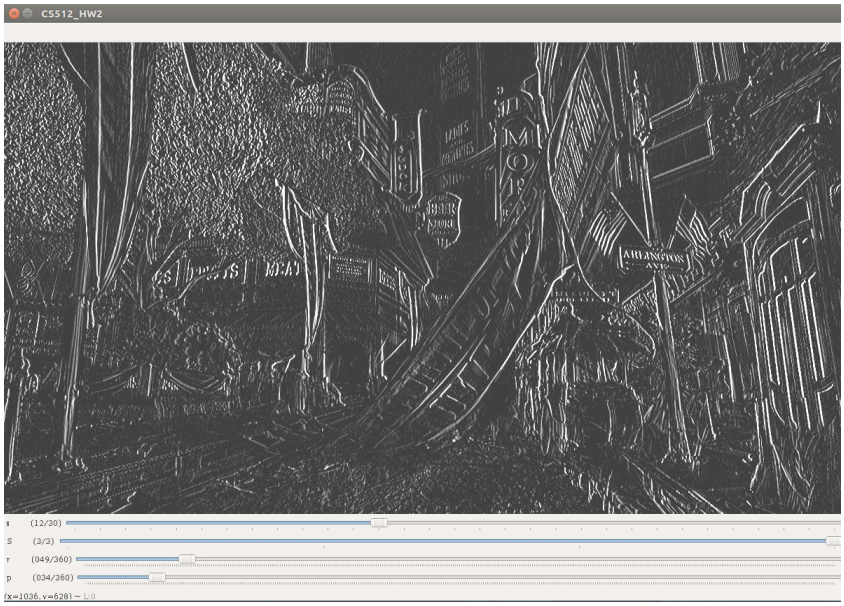
- **Visual verification** : the final image is two times smaller than the original.
- **Multiple calls** : whenever d is entered again, the image is divided by two, with a limit of 15 pixels.
- **Computation time** :very fast, instant change

D : downsample the image by 2 and smooth it

Command	Result
D - Downsample by 2 and smooth	

All the test and results are the same that those for the command “d” and “s”

x : convert to grayscale and apply x derivative filter

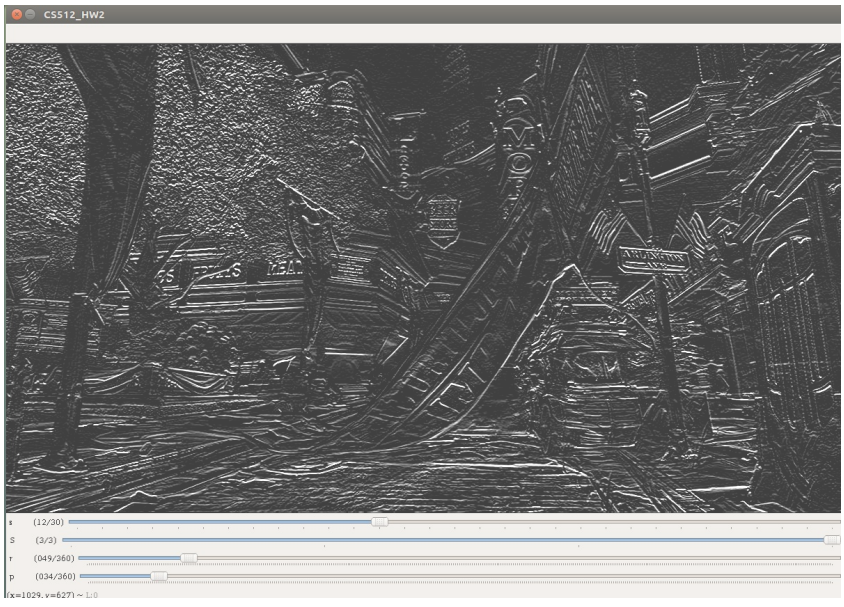
Command	Result
x - Grayscale and Gradient in x	

Same test and results for grayscale as before.

Tests and results :

- **Visual verification** : the final image is the gradient in x of the original one;
- **Multiple calls** : whenever x is entered, the gradient in x of the current image is diplayed.
- **Computation time** :very fast, instant change

y : convert to grayscale and apply y derivative filter

Command	Result
y - Grayscale and Gradient in y	

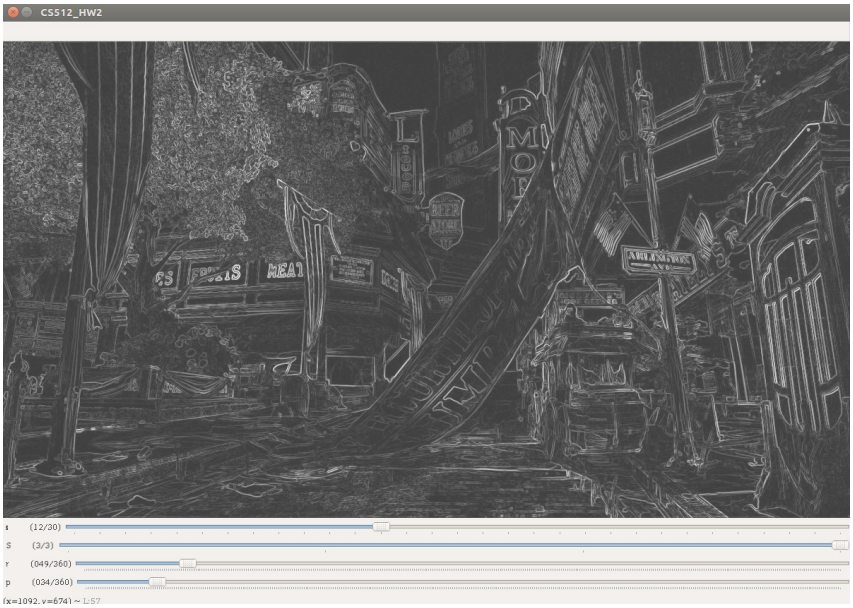
Same test and results for grayscale as before.

Tests and results :

- **Visual verification** : the final image is the gradient in y of the original one;
- **Multiple calls** : whenever y is entered, the gradient in y of the current image is diplayed.
- **Computation time** : very fast, instant change

m : convert to grayscale and show magnitude

Same test and results for grayscale as before.

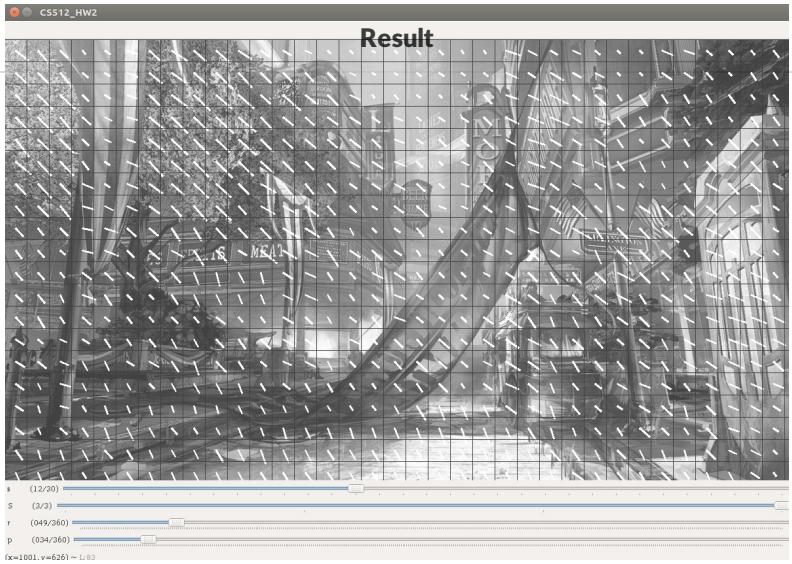
Command	Result
m - Grayscale and Magnitude	

Tests and results :

- **Visual verification** : the final image is the magnitude of the original one;
- **Multiple calls** : whenever m is entered, the magnitude of the current image is diplayed.
- **Computation time** :very fast, instant change

p : convert to greyscale and plot gradient vectors

Command	Result

Command	Result
p - Grayscale and plot gradient vectors	

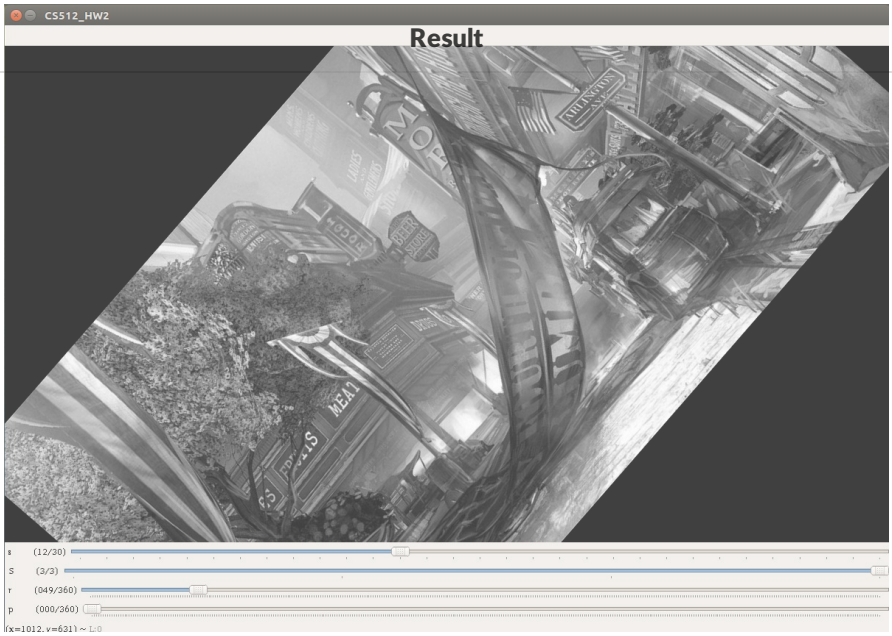
Same test and results for grayscale as before.

Test and results :

- **Visual verification :** the gradient vectors are plotted on the final image, the amount of vectors is proportional to the value on the trackbar, and the direction of the vectors seems to be the right one.
- **Trackbar input** Whenever the trackbar value is changed but the key haven't been entered, nothing happened.
- **Trackbar input** If the key has been entered, the image will update in real-time with the trackbar value.
- **Computation time :** very fast, instant change, exept for very values of N. to prevent a freeze for small value of N, a limit has been fixed at 25 pixels.

r : convert to greyscale and rotate

Command	Result

Command	Result
r - Grayscale and Rotate	

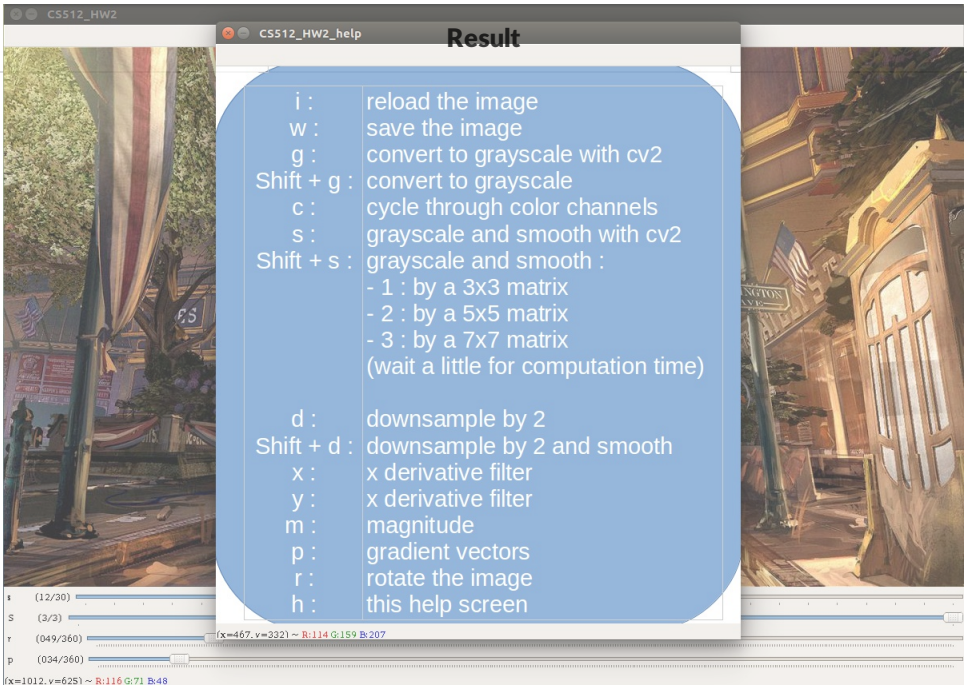
Same test and results for grayscale as before.

Test and results :

- **Visual verification :** the final image is rotated and the angle is proportional to the trackbar value.
- **Trackbar input** Whenever the trackbar value is changed but the key haven't been entered, nothing happened.
- **Trackbar input** If the key has been entered, the image will update in real-time with the trackbar value.
- **Computation time :** very fast, instant change.

h : display help

Command	Result

Command	Result
h : help	

Test and results :

- **Visual verification :** the help appear on a new window.
- **Multiple calls** If the key is entered but the help is already displayed, no other windows will pop up
- **Persistence** If an other key is entered, the help window stay open on the side
- **Relevance** The information on the windows are relevant.
- **Computation time :** very fast, instant display.

Remark : as the help is the display of an image save on the computer, The program always have to be launched in the same folder as the help image.

References

All the support from the course CS512-Computer vision.
The official OpenCv documentations.
The forum StackOverFlow.