Spatial filtering and Mathematical Morphology

1 Introduction

The objective of this session is to practice with spatial filtering and mathematical morphology, following the lecture 2 of Computer Vision course. You can get more information about lectures and associated practical sessions on my website: `http://chateaut.fr`. Essential needed knowledges includes a beginning level of computer science (Linux, Python programming with `Opencv`, `Numpy`, `Matplotlib`, and `Pytorch` (for sessions on Deep Learning)) and applied mathematics.

The listing 1 shows an "Hello Word" python script that loads and convert an image using `opencv`; and displays it with `matplotlib` and illustrates relevant functions.

2 Spatial filtering of images

Spatial linear filters are widely using in image processing and computer vision. Given an image `I` and a kernel `W`, filtered image is computed with a convolution operation:

$$\mathtt{I_f} = \mathtt{W} * \mathtt{I} \tag{1}$$

where $*$ denotes the discrete convolution operation.

$$\mathtt{I_f}(x,y) = \sum_{i=-m}^{m} \sum_{j=-n}^{n} W(i,j).\mathtt{I}(x+i, y+j) \tag{2}$$

Image processing operations like denoising or edge detection are achieved with linear spatial filters. Moreover, convolulional filters are the basis blocs for state-of-the-art Deep Convolutional Neural Networks. `Opencv` already implements a function to process linear spatial filters on images:

```
imf = cv2.filter2D(img, cv2.CV_32F, W)
```

with:

- `img`: the original image
- `W`: the kernel matrix

The function returns the filtered image. For Gaussian filtering, you should use:

```
WX = cv2.getGaussianKernel(kernel_size,sigma)
```

```
WG = WX @ np.transpose(WX)
```

to get a 2D Gaussian kernel Applying a gaussian filter can also be achieved using the function:

```
blur = cv2.GaussianBlur(img,(5,5),0)
```

where $(5, 5)$ is the std. of the Gaussian 2D kernel along each axis.

Study to realize

You should work from the baseline script `testfilter.py`.

1. **Image denoising**:

   Noise generally produces high frequency signal in the image. low pass filtering is the basic image processing tool to reduce such noise:

   - Medium filter (example: of size 3×3)
     $W_1 = \frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$
   - Gaussian filter

   $$W_G(i,j) = C \exp\left[-\frac{i^2+j^2}{2.\sigma^2}\right]$$

   For a std. value of Gaussian noise (line 23), of 50, compare a Medium filter of size 3×3 with one of size 11×11. which one has the highest bandwidth?

   For a std. value of Gaussian noise of 50, compare a Gaussian filter of $\sigma = \sqrt{2}$ with one of $\sigma = 3$. which one has the highest bandwidth? **Tip**: the kernel size for a Gaussian filter is related to the bandwidth (linked to σ). A correct approximation of the Gaussian function is realized for $\pm 2\sigma$. The reulting size of the kernel should be adjusted to the first odd integer greater than 4σ ($2 \times 2\sigma$).

2. **Gradient estimation (toward edge detection)**:

   Edge detection was one of the most popular operations for image processing. It reduces the data (1 bit per pixel) providing a higher level representation than the raw image. One way to compute edges into images is to estimate derivation (so-called gradient). Since image is a 2D-signal, gradient estimation is decomposed into x and y components. Linear spatial filtering is a widely used tool to do the job with Sobel filters:

   $$\texttt{Wx} = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \; ; \; \texttt{Wy} = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

   Sobel operators approximates the first spatial derivative of an image:

   $$\texttt{Gx} = \texttt{Wx} * \texttt{I} \; ; \; \texttt{Gy} = \texttt{Wy} * \texttt{I}$$

   The norm and orientation pseudo-images of gradient can be computed from $\texttt{Gx}, \texttt{Gy}$ by:

   $$||\texttt{G}|| = (\texttt{Gx}^2 + \texttt{Gy}^2)^{0.5} \; ; \; \boldsymbol{\theta} = \text{atan2}(\texttt{Gy}, \texttt{Gx})$$

   where atan2 denotes the operator that return the pseudo-image of angles into a $[0, 2\pi]$ interval corresponding to the vector of components $\texttt{Gx}, \texttt{Gy}$.

   You should modify your script to compute and display $\texttt{Gx}, \texttt{Gy}, ||\texttt{G}||, \boldsymbol{\theta}$ for an image **with no additional noise**.

Now, add noise (50) and display the gradient estimation. What do you conclude on the robustness of the estimator related to high frequencies?

What happen if you apply a Gaussian denoising filter before estimating gradients? Conclude?

3. **Example of non-linear filter: median filter**: Median filter replaces each pixel by the median value of neighbouring pixels. This is a non linear operation. This filter preserves edges and reduces blur resulting to other classical denoising filters.

   `Opencv` implements such filter into the following function:

   ```
   imfm = cv2.medianBlur(img,sf)
   ```

   where `img` is the noisy image, sf is the aperture linear size; it must be odd and greater than 1, for example: 3, 5, 7. The function returns the filtered image. Compare the median filter to the gaussian and the medium ones on a noisy image.

3 Binary image processing with morphological operators

This section illustrates the principal morphological operators used in pre-processing of binary images. `opencv` already implements several morphological operator:

```
erosion = cv2.erode(imb,kernel,iterations = 1)
```

This is an example for erosion. `dilate`. Opening and Closing are provided with:

```
opening = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)
```

. See the documentation of opencv for further information.
You should work with the script `testmorph.py`.

1. Compare the erode operation using a 3×3 square kernel and a 7×7 square kernel.
2. Compare three erode operations using a 3×3 square kernel with one erode operation using a 7×7 square kernel. How many operations are achived for one pixel with 3 times the 3×3 square kernel and with only one time the 7×7 square kernel? Conclude.
3. Find a set of operations that remose the noise and close the plate into the image.
4. Compute the inner and outer edge of the plate using MM.

4 Application to perimeter and area estimation

Let X be a company that produces some composite plies (cf. fig 1). A control system is developed to estimate the perimeter and the area of each piece. A classical algorithm is:

1. Read the image
2. binarization of the image

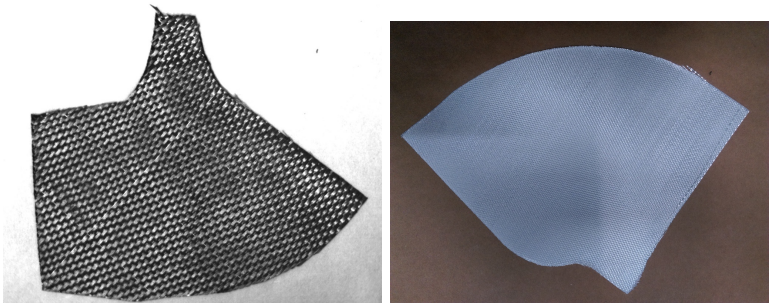

Figure 1: some images captured from the composites plies

3. MM operation to extract only the desired shape
4. Area will be estimated with the number of pixels that belong to the ply. The number of pixels that belong to the edges will be used to estimate the perimeter of the ply. Both output and input edge computation will be used to estimate a interval of the perimeter.

You have to implement this algorithm and test it on the image `im1.bmp`. The python script `testmorpho.py` should be a baseline for your work.

Listing 1: HelloWord Python script: load an image and convert it using opencv then display using matplotlib

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Aug  6 08:46:29 2018
hello_wold simply loads an image with opencv, converts it to RGB and GRAYSCALE
and displays it using matplotlib
@author: thierrychateau
"""
import cv2
from matplotlib import pyplot as plt
import numpy as np
# This is an example of function with a string argument that returns another
# string
def my_func(my_arg):
    string2 = my_arg + ' toto ';
    return string2
# This is the main function that is launched when this file is run as a script
def main():
    # call function my_func
    st = my_func("my name is")
    print(st)
    # Read an image file (im is a numpy matrix
    # (nblines,nbcolums, BGR 3 channels))
    filename = "billard_large.jpg"
```

```
    img = cv2.imread(filename)
    # Convert to RGB
    imgrgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    # Convert to grayscale image
    imggray=cv2.cvtColor( imgrgb, cv2.COLOR_RGB2GRAY )
    ## Display the RGB and gray images
    # divide plot screen into 1 line, 2 columns and select first subscreen as
    # current screen
    plt.subplot(121)
    plt.imshow(imgrgb)
    plt.title('Color RGB image')
    plt.subplot(122)
    plt.imshow(imggray, 'gray',vmin=0, vmax=255)
    plt.title('Grayscale image')
#
if __name__ == "__main__":
    # execute only if run as a script
    main()
```