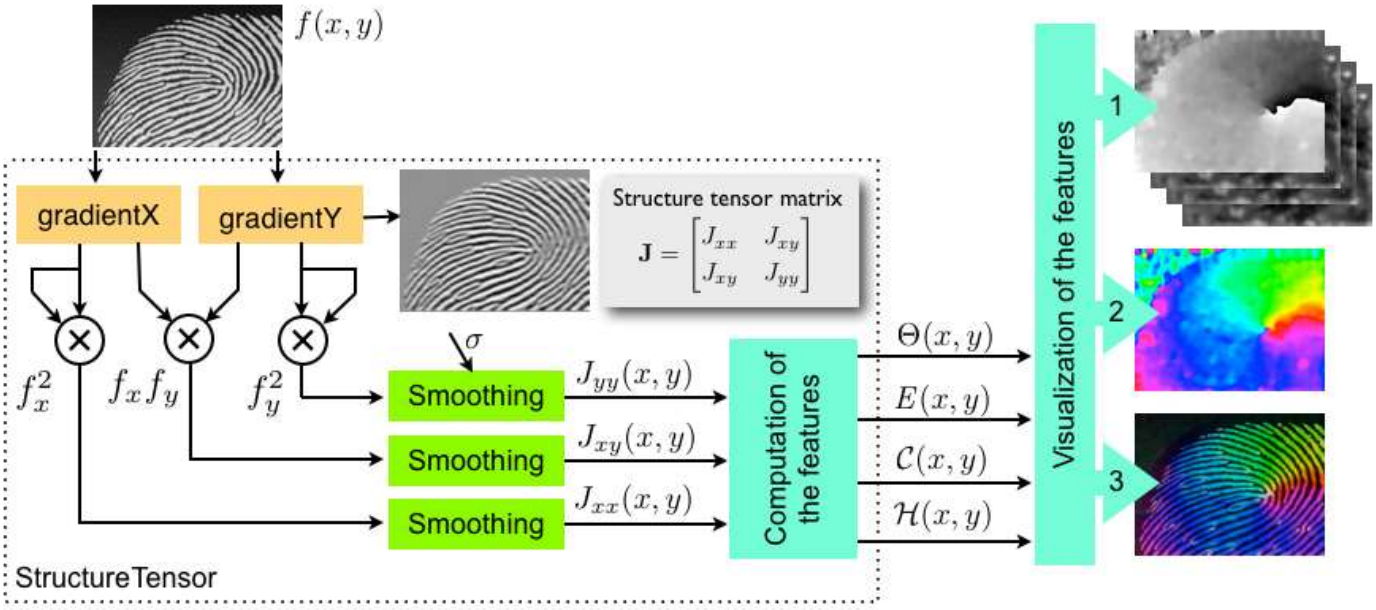
Biomedical Imaging Group

IP-LAB: Image Processing Laboratories

Orientation analysis in images

1. Introduction

We propose to study the implementation of directional analysis tools based on the computation of a structure tensor. The overall algorithm is shown in the following flowchart where f(x,y) is the graylevel input image.



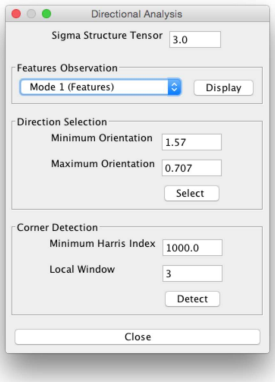
Your task is to implement successively:

- gradientX and gradientY: plugins **Gradient X** and **Gradient Y** (Question 2);
- smoothing: plugin **Gaussian Smoothing** (Question 3);
- the computation and three visualization schemes of the features: plugin **Directional Analysis**, button Display (Question 4);

then, based on the features:

- a tool to select specific orientations, plugin **Directional Analysis**, button Select (Question 5)
- the Harris corner detector, plugin **Directional Analysis**, button Detect (Question 6)

We want you to consider mirror boundary conditions for all modules in this session.



2. Gradient

A simple way to estimate the gradient of an image is to convolve it with the partial first derivative of a 2D Gaussian function g(x,y). To get the gradient in the horizontal direction, convolve with ∂g(x,y)/∂x. To get the gradient in the vertical direction, convolve with ∂g(x,y)/∂y. Here, ∂g(x,y)/∂x and ∂g(x,y)/∂y are on a L-tap mask filter. The length L of the mask is constrainted to an odd value according to the expression L=2*(int)ceil(σ*3)+1. We assume that the size of the image is larger than 2*L.

$$g(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

a. Give the explicit expression of ∂g(x,y)/∂x and ∂g(x,y)/∂y for $\sigma=2^{-1/2}$. Fill the report.
b. Write the method `gradientX()` and the method `gradientY()` for the hard-coded value of $\sigma=2^{-1/2}$. Test it on the image ***fingerprint*** by using the plugins **Gradient X** and **Gradient Y**. Fill the report.

Note: These two filters are separable. You can choose a separable or a non-seperable implementation. Nevertheless, the non-separable version is much easier to code because the ImageAccess routine `getNeighborhood()` takes care of the mirror boundary conditions.

3. Gaussian Smoothing

We choose the Gaussian filter as smoothing operator. An efficient approximation is provided by a cascade of N symmetric exponential filters. In practice, we choose N=3. The value of the standard deviation σ is entered through the dialog box.

3.1 Determination of the poles

The equivalent variance σ^2 of a cascade of filters is the sum of the variances of the individual operators; here, $\sigma^2 = 2 * N * a / (1-a)^2$ (See Course Notes Image Processing I, page 3-52). Express *a* (with |a|<1) in function of σ and N; this yields the pole value to be used in Section 3.2. Fill the report.

3.2 Implementation of a 2D Gaussian filter

Write the routine `smoothing()`, which implements the Gaussian filter in a separable manner. We provide the method:

```
double[] out = Convolver.convolveIIR(double input[], double poles[])
```

that performs the symmetrical IIR filtering of a 1D signal `input[]` with a filter defined by its poles `poles[]`. Replace `a=0.5` in the code by the correct pole value (|a|<1).

3.3 Applying the smoothing

The method can be tested individually by using the plugin **Gaussian Smoothing**. Apply the smoothing filter on ***dendrochronology*** for σ=1, σ=4, and σ=30. Observe the Fourier transform of the output images. Measure the mean and the standard deviation of the output images. Fill the report.

4. Directional Analysis

4.1 Implementation of the main routine of the structure tensor

Write the main method `structureTensor()` which implements the whole chain of processing of the algorithm described in the flowchart. This algorithm has to compute features for every pixel of the image: orientation (in radians, from -Π/2 to -Π/2), gradient energy, coherency, and Harris index. The features are stored in an array of 4 ImageAccess objects.

Note on the implementation: If $J_{yy}+J_{xx} < 0.01$, then set the coherency to 0 to avoid a division by zero.

The features can be individually visualized using the plugin **Directional Analysis**, button Display, Mode 1 (provided). Apply your routine to ***fingerprint*** with σ=3 and fill the report.

Orientation	$\Theta = \frac{1}{2}\arctan\left(\frac{2J_{xy}}{J_{yy}-J_{xx}}\right)$
Gradient Energy	$E = J_{xx} + J_{yy}$
Coherency	$\mathcal{C} = \frac{\sqrt{(J_{yy}-J_{xx})^2 + 4J_{xy}^2}}{J_{yy}+J_{xx}}$
Harris Index	$\mathcal{H} = \det(\mathbf{J}) - \kappa\,\mathrm{trace}(\mathbf{J})^2 \quad \text{with} \quad \kappa = 0.05$

4.2 Visualization of the features in color

A better visualization for an angle is to use the hue channel of the color representation in HSB (hue, saturation, brightness). The class `new ImageAccess.Display(false, H, S, B, "Title")` displays a HSB color image if the H is ImageAcccess object from -Π/2 to -Π/2, S and B are two ImageAccess objects from 0 to 255. Complete the method `visualizeFeatures()` to obtain the following modes.

- Mode 2: H is the orientation, S and B are two constant images containing 255.
- Mode 3: H is the orientation, S is the coherency image rescaled to 255, and B is the input image.

Experiment your algorithm on the images ***wave-ramp*** and ***fingerprint*** with different σ. Fill the report.

5. Selection of specific directions

We proposed to develop a tool that only selects local area with a specific orientation. In particular, the algorithm has to preserve pixels which have the following structure-tensor features:

- $E(x,y) > 0.5 * E_{max}$ (E_{max} is the maximum energy)
- $C(x,y) > 0.5$
- $\theta_{min} < \theta(x,y) < \theta_{max}$

Write the method `selectDirection()` that shows an output image keeping intact the pixels with the given features, while the other pixels are set to 0. The parameters θ_{min} and θ_{max} are given through the dialog box of the plugin **Directional Analysis**, button Select. This method has to call first the method `structureTensor()` to compute the features.

Apply your algorithm to the image ***fingerprint***. By selecting the right parameters σ, θ_{min} and θ_{max}, select:

- the horizontal lines that correspond to the orientations around 0 radian (+/- 0.2 radians)
- the lines that correspond to the angle of the dominant orientation (+/- 0.2 radians). Hint: estimate the dominant orientation by studying the content of the orientation map.

Give the parameters and the output images in the report.

6. Harris corner detection

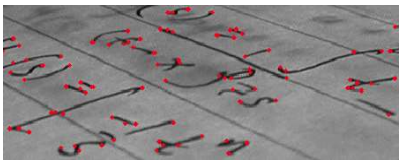
6.1 Implementation

The Harris corners are located at the positions (x,y) where H(x,y) is above a threshold *T* and H(x,y) is a local maximum within a square neighborhood window of size *LxL* (L is odd). The parameters *L* and *T* are given through the dialog box of the plugin **Directional Analysis**, button Detect.

Write the main method `detectCorner()` that first calls the method `structureTensor()` and that then draws a red cross over every detected corners. The red cross of size 10x10 can be overlaid at (x,y) using the following ImageAccess commands:

```
ImageAccess.Display display = in.show();
display.overlayCross(x, y, 10);
```

6.2 Test your algorithm



Test your algorithm on the image ***harris-corner***. Give a set of parameters that yield exactly 12 corners for the letter H. Fill the report.

6.3 Comparaison

The article "Corner Detection" of Wikipedia presents several algorithms of corner detection at https://en.wikipedia.org/wiki/Corner_detection. The results of the algorithms are shown on the image corners.png. Use the image ***corners***, set the parameters of your algorithm and run it to obtain the same quantity of corners. We expect between 40 and 100 corners well located on the keypoints of the image.

Notes of programming			
Compute the square root	y = Math.sqrt(x);	Compute the arctan dy/dx in [-Π,Π]	a = Math.atan2(dy, dx);
Give the value of π	Math.PI	Show an ImageAccess object	image.show("Title");

```
import ij.IJ;

public class Code {

	public ImageAccess gradientX(ImageAccess in) {
		// TODO: Add you code here
		return in;
	}

	public ImageAccess gradientY(ImageAccess in) {
		// TODO: Add you code here
		return in;
	}

	public ImageAccess smoothing(ImageAccess in, double sigma) {
		double poles[] = new double[3];
		double a = 0.5; // TODO: Replace the expression of a
		poles[0] = poles[1] = poles[2] = a;
		// TODO: Add you code here
		return in;
```

```
        }

        public ImageAccess[] structureTensor(ImageAccess in, double sigma) {
                ImageAccess orient              = new ImageAccess(in.nx, in.ny);
                ImageAccess coherency   = new ImageAccess(in.nx, in.ny);
                ImageAccess energy              = new ImageAccess(in.nx, in.ny);
                ImageAccess harris              = new ImageAccess(in.nx, in.ny);
                // TODO: Add you code here
                ImageAccess features[] = new ImageAccess[] {orient, energy, coherency, harris};
                return features;
        }

        public void visualizeFeatures(ImageAccess in, double sigma, int mode) {
                ImageAccess[] features = structureTensor(in, sigma);
                if (mode == 1) {
                        features[0].show("Orientation");
                        features[1].show("Energy");
                        features[2].show("Coherency");
                        features[3].show("Harris");
                }
                if (mode == 2) {
                        // TODO: Add you code here
                        // new ImageAccess.Display(false, H, S, B, "HSB 2");
                }
                if (mode == 3) {
                        // TODO: Add you code here
                        // new ImageAccess.Display(false, H, S, B, "HSB 3");
                }
        }

        public void selectDirection(ImageAccess in, double sigma, double minOrientation, double maxOrientation) {
                ImageAccess[] features = structureTensor(in, sigma);
                // TODO: Add you code here
        }

        public void detectCorner(ImageAccess in, double sigma, double min, int L) {
                ImageAccess[] features = structureTensor(in, sigma);
                ImageAccess.Display display = in.show();
                // TODO: Add you code here
                display.overlayCross(in.nx/2, in.ny/2, 10); // A cross at the center
        }
}
```

---

webmaster.big@epfl.ch • 19.02.2020