# Derivations used in `Image.hpp` calculations

## First order smoothing recursive filtering

Given a raw image, the smoothed version of this image using a 1st order exponential recursive filter, writes for each channel:

$$output[i, j] = \sum_{uv} c\, a^{|u-i||v-j|}\, input[u, v],\ c \overset{\text{def}}{=} \left(\tfrac{1+a}{1-a}\right)^2$$

The parameter is related to the smoothing filter window size in pixel, with 90% of the signal to be in a the pixel window:

$$a = (1 - p)^{\frac{1}{window}}$$

with $p = 0.9$.

The implementation is a simple four directional 1st order recurrent filter, writing for a $\{0, H\{ \times \{0, V\{$ discrete image:

$$\begin{array}{rcll} output[i, j] & \leftarrow & a\, output[i - 1, j] + (1 - a)\, input[i, j] & i = 1 \cdots H - 1 \\ output[i, j] & \leftarrow & a\, output[i + 1, j] + (1 - a)\, output[i, j] & i = H - 2 \cdots 0 \\ output[i, j] & \leftarrow & a\, output[i, j - 1] + (1 - a)\, output[i, j] & j = 1 \cdots V - 1 \\ output[i, j] & \leftarrow & a\, output[i, j + 1] + (1 - a)\, output[i, j] & j = V - 2 \cdots 0 \end{array}$$

In particular the computation time does not depend on the smoothing window.

## Color edge eigen elements

Given a blue, red, green $(b, g, r)$ three-channel color image the local intensity derivative $3 \times 2$ matrix SVD writes:

$$\mathbf{J}(\mathbf{p}) \overset{\text{def}}{=} \partial_{\mathbf{p}} \mathbf{i}(\mathbf{p}) = \underbrace{\left( \begin{array}{cc} k_b & l_b \\ k_g & l_g \\ k_r & l_r \end{array} \right)}_{\mathbf{U}} \left( \begin{array}{cc} G & 0 \\ 0 & g \end{array} \right) \left( \begin{array}{cc} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{array} \right), \mathbf{p} \overset{\text{def}}{=} (i, j)^T, \mathbf{i} \overset{\text{def}}{=} (b, g, r)^T$$

with $\mathbf{U}^T\, \mathbf{U} = \mathbf{I}$ and we easily obtain, writing $J_{xx} \overset{\text{def}}{=} \sum_{k \in \{b,g,r\}} J_{kx}^2$, $J_{xx} \overset{\text{def}}{=} \sum_{k \in \{b,g,r\}} J_{kx}\, J_{ky}$, $J_{yy} \overset{\text{def}}{=} \sum_{k \in \{b,g,r\}} J_{ky}^2$,

$$\left\{ \begin{array}{rcll} \sqrt{J_{xx} + J_{yy}} & = & \sqrt{G^2 + g^2} & \text{edge magnitude} \\ \frac{1}{2} \arctan\left( \frac{2\, J_{xy}}{Jyy - J_{xx}} \right) & = & \theta & \text{edge orientation} \end{array} \right.$$

as verified by the following piece of code:

```
with(LinearAlgebra):
J := Matrix([[k_b, l_b], [k_g, l_g], [k_r, l_r]]) . Matrix([[G, 0], [0, g]]) . Matrix([[cos(theta), -sin(theta)], [sin(theta), cos(theta)]]):
eq := simplify({
 Jxx = J[1][1]^2 + J[2][1]^2 + J[3][1]^2,
 Jxy = J[1][1] * J[1][2] + J[2][1] * J[2][2] + J[3][1] * J[3][2],
 Jyy = J[1][2]^2 + J[2][2]^2 + J[3][2]^2
}, {
 k_b^2 + k_g^2 + k_r^2 = 1,
 k_b * l_b + k_g * l_g + k_r * l_r = 0,
 l_b^2 + l_g^2 + l_r^2 = 1
}):
zero := factor(combine(simplify(subs(eq, {
Jxx + Jyy                        - (G^2 + g^2),
2 * Jxy / (Jyy - Jxx)            - tan(2 * theta),
(Jyy - Jxx)^2 + (2 * Jxy)^2      - (G^2 - g^2)^2,
Jxx * Jyy - Jxy^2                - G^2 * g^2
}))));
```

It generalizes the mono-channel monochrome edge magnitude $G \overset{\text{def}}{=} \sqrt{I_x^2 + I_y^2}$ and orientation $\theta = -\arctan\left(I_y/I_x\right)$, or a three-channel color image with proportional gradients, i.e., with $k_b^2 + k_g^2 + k_r^2 = 1$, while $k_g = k_r = 0$ in the mono-channel case:

$$\mathbf{J}(\mathbf{p}) = \partial_{\mathbf{p}} \mathbf{i}(\mathbf{p}) \overset{\text{def}}{=} \left( \begin{array}{c} k_b \\ k_g \\ k_r \end{array} \right) (I_x, I_y) = \underbrace{\left( \begin{array}{c} k_b \\ k_g \\ k_r \end{array} \right) (1, 0)}_{\mathbf{U}} \left( \begin{array}{cc} G & 0 \\ 0 & 0 \end{array} \right) \left( \begin{array}{cc} \frac{I_x}{G} & \frac{I_y}{G} \\ -\frac{I_y}{G} & \frac{I_x}{G} \end{array} \right),$$

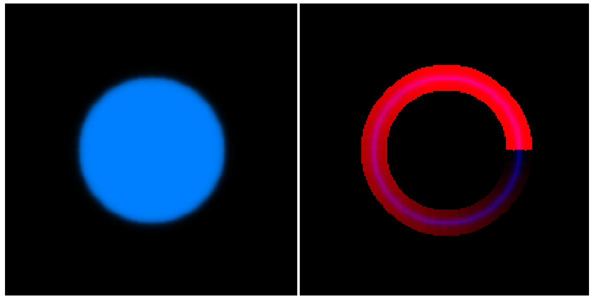
i.e., it corresponds to the case where $g = 0$.

As a consequence, $g/G$ provides an indication of how much the three-channels are linearly coherent.

Then the tricky point is to manage the angle $\theta$ in the four quadrants, since the angle $\theta$ is estimated in $[-\Pi/2, \Pi/2]$, and in fact only up to a $\Pi$ constant (adding $\Pi$ in the equation does not modified then). we thus must estimate the direction of the gradient to obtain this $\Pi$ constant. Here we estimate this direction averaging the gradient of the three channels, i.e., use the following experimental rule, writing $J_x \overset{\text{def}}{=} \sum_{k \in \{b,g,r\}} J_{kx}$ and $J_y \overset{\text{def}}{=} \sum_{k \in \{b,g,r\}} J_{ky}$, in order to obtain $\theta \in [0, 2\Pi[$:

$$\begin{array}{rcl} \theta & \leftarrow & \text{if } \theta < 0 \text{ then } \Pi + \theta \text{ else } \theta \\ \theta & \leftarrow & \text{if } J_y > 0 \text{ or } (J_x > 0 \text{ and } J_y = 0) \text{ then } \Pi + \theta \text{ else } \theta, \end{array}$$

as visible for this test image:



remembering that the vertical axis is up side down in images, and noticing that the image has been smoothed before the edge calculation, in order to reduce the discretization local errors. The left image is a circular blob, and the right image displays the edge magnitude in blue, and the edge orientation in red, from 0 to 360 degree.

**Image blob segmentation, using a standard colorization algorithm**

Here are the algorithm and derivations used in `Image::channel({do: blob_segments`.

We consider an image with a background and “blobs”, i.e., disconnected regions.

We consider as input binary image (a “calque”) `I[p]` $\in \{-1, M\{$, `p = (i, j)` $\in \{0, width\{ \times \{0, height\{$, where $M \overset{\text{def}}{=} width\, height$ is the image size, while `I[p] < 0` outside a blob.

On output `I[p]` $\in \{-1, M\{$ corresponds to the index of the obtained blob.

Here we consider the following indexing: `I[p] = q, q = i + width j`, stating that the pixel at `p` is chained to the pixel at `q`, unless `I[p] < 0`, i.e. a pixel outside a blob.

We consider the following algorithm to collect the different connected blobs:

```
- Initialization: Each pixel is considered as a singleton, i.e., a blob of size 1, setting:
   I[p] = p
- Repeat for each pixel:
  - Chained index reduction: Each pixel index is set to the smallest value, i.e.:
   while(I[I[p]]< I[p]) I[p] = I[I[p]]
  - Merge connected pixels, horizontaly and verticaly (i.e., 4-connectivity), i.e.:
   if (0 <= I[i, j] && 0 <= I[i-1, j]) I[i, j] = I[i-1, j] = min(I[i, j], I[i-1, j])
   if (0 <= I[i, j] && 0 <= I[i, j-1]) I[i, j] = I[i, j-1] = min(I[i, j], I[i, j-1])
  Untill no more index change.
```

At the convergence, each pixel index corresponds to the blob smallest pixel index.

Then each blob are scanned in order to calculate basic parameters: size, centroids $\mathbf{c}$, enclosing rectangle, second order momenta geometry, i.e. using the very standard derivation:

$$\begin{array}{rcl} m_{pq} & \stackrel{\rm def}{=} & \sum_{ij} I[i,j]\, i^p\, j^q, \\ \mathbf{c} & \stackrel{\rm def}{=} & \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}}\right)^T, \\ \mu_{pq} & \stackrel{\rm def}{=} & \sum_{ij} I[i,j]\, (i - c[0])^p\, (j - c[1])^q, \\ \mathbf{C} & \stackrel{\rm def}{=} & \left(\begin{array}{cc} \mu_{20} & \mu_{11} \\ \mu_{11} & \mu_{02} \end{array}\right) = \left(\begin{array}{cc} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{array}\right) \left(\begin{array}{cc} L & 0 \\ 0 & l \end{array}\right) \left(\begin{array}{cc} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{array}\right). \end{array}$$

yielding:

$$\left\{\begin{array}{rcl} L & = & \frac{\mu_s + \mu_q}{2}, \\ l & = & \frac{\mu_s - \mu_q}{2}, \\ \tan(2\,\theta) & = & \frac{-2\,\mu_{11}}{\mu_d}, \\ e & = & \frac{\mu_q}{\mu_s}, \end{array}\right. \text{ with } \left\{\begin{array}{rcl} \mu_s & \stackrel{\rm def}{=} & \mu_{20} + \mu_{02}, \\ \mu_d & \stackrel{\rm def}{=} & \mu_{20} - \mu_{02}, \\ \mu_q & \stackrel{\rm def}{=} & \sqrt{\mu_d^2 + \mu_{11}^2}. \end{array}\right.$$

Here $L \geq l$ and the thinness is $0$ for a circular form and $1$ for a line segment, as verified by this piece of `maple` algebra:

```
with(LinearAlgebra):
assume(mu_20 > 0, mu_02 > 0, mu_11 :: real):
Mu := Matrix([[mu_20, mu_11], [mu_11, mu_02]]):
mu_R := Matrix([[cos(theta), -sin(theta)], [sin(theta), cos(theta)]]):
mu_D := Matrix([[L, 0], [0, l]]):

mu_s := mu_20 + mu_02:
mu_d := mu_20 - mu_02:
mu_q := sqrt(mu_d^2 + mu_11^2):
sl := {
  L = (mu_s + mu_q)/2,
  l = (mu_s - mu_q)/2,
  sin(theta_2) = -2 * mu_11 / mu_q,
  cos(theta_2) = mu_d / mu_q
}:
zero := simplify(subs(sl, combine(subs(theta = theta_2 / 2, simplify(convert(Mu - Transpose(mu_R) . mu_D. mu_R, set))))));
```

## Normalized correlation similarity

We consider the standard image correlation ratio between two images $I_0[i,j]$ and $I_1[i,j]$ of the same sizes $S = width\, height$, writing:

$$\mu_0 \overset{\text{def}}{=} \frac{1}{S} \sum_{ij} I_0[i,j], \mu_1 \overset{\text{def}}{=} \frac{1}{S} \sum_{ij} I_0[i,j], \sigma_0 \overset{\text{def}}{=} \sqrt{\frac{1}{S-1} \sum_{ij} (I_0[i,j] - \mu_0)^2}, \sigma_1 \overset{\text{def}}{=} \sqrt{\frac{1}{S-1} \sum_{ij} (I_1[i,j] - \mu_1)^2}$$

we obtain:

$$R(I_0, I_1) \overset{\text{def}}{=} \frac{1}{S-1} \sum_{ij} \left(\frac{I_0[i,j]-\mu_0}{\sigma_0}\right) \left(\frac{I_1[i,j]-\mu_1}{\sigma_1}\right) = \frac{S\, m_{01}^{11} - m_{01}^{10}\, m_{01}^{01}}{\sqrt{(S\, m_{01}^{20} - (m_{01}^{10})^2)\, (S\, m_{01}^{02} - (m_{01}^{01})^2)}} \in [-1, 1]$$

writing $m_{01}^{pq} = \sum_{ij} I_0[i,j]^p\, I_1[i,j]^q$, thus $\mu_0 = m_{10}/m_{00}$, $\mu_1 = m_{01}/m_{00}$, $\sigma_0^2 = (m_{20} - m_{10}^2/m_{00})/(m_{00} - 1)$, $\sigma_1^2 = (m_{02} - m_{01}^2/m_{00})/(m_{00} - 1)$.

It is known to be symmetric and normalized $-1 \leq R(I_0, I_1) \leq 1$.

It is related to the Euclidean distance between the two normalized images:

$$D(I_0, I_1) \overset{\text{def}}{=} \frac{1}{S-1} \sum_{ij} \left(\frac{I_0[i,j]-\mu_0}{\sigma_0} - \frac{I_1[i,j]-\mu_1}{\sigma_1}\right)^2 = 2\, (1 - R(I_0, I_1)) \in [0, 4]$$

## Incremental affine transformation

- The transformation is a general or specialized affine transform written:

$$\begin{pmatrix} i \\ j \end{pmatrix} \leftarrow \begin{pmatrix} translation_i \\ translation_j \end{pmatrix} + \begin{pmatrix} zoom + warp & -rotation + twist \\ rotation + twist & zoom - warp \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix}$$

where:
- $horizontal$: considers only an horizontal translation, 1 parameter,
- $translation$: considers a 2D horizontal and vertical translation, 2 parameters,
- $rigid$: considers translation and a (local) 2D rotation, 3 parameters,
- $similarity$: considers translation, rotation and a scale factor change (i.e., a zoom), 4 parameters,
- $affine$: considers translation, rotation, zoom and warp and twist shears, i.e., a 6 parameters affine transform.