Indexing big colored image bank : Texture 3.0

Etienne Caillaud, Thomas Le Bris, Ibrahima Gueye, Gaetan Adier

June 11, 2015

Contents

1 Introduction

2 Team presentation

The project took place in the XLIM-SIC laboratory of the University of Poitiers.

This Laboratory works on fields of research of Graphic Informatique, Color images and Communication systems.

We worked on indexation images in a big database whith the team composed by :

- Noel Richard (Researcher in Color images): Supervisor
- David Helbert (Researcher in Signal-Image-Communications): Supervisor
- Thierry Urruty (Researcher in Color images): Client

3 User requirement

4 State of art

Color image processing has become a major issue since a few years, most of the colour texture discrimination having been explored using the marginal colours method. The issue is that we are now able to do colour image recognition on digital pictures but the results on nature pictures are rather mediocre.

The CLEF contest has been created as an answer to that problematic, making universities' and laboratories' own solutions compete against each other in order to find the best colour texture feature.

In this document we will introduce key-points and their use in the various descriptors. We will go first with the standard ones which are SIFT SURF and opponent SIFT. The last one being the descriptor used by FINKI, the laboratory from the last year contest we chose as reference to compare our results. We will then use a new descriptor provided by Noel Richard, the C_2O.

4.1 Key-points

4.1.1 Scale-space extrema detection

For this type of key-point detection, the aim is to only retrieve the useful key-points (which are characterizing the image the best) without taking an "abstract" of the whole image as the dense grid method does. To do it, the first thing that is done is to create a pyramidal tree containing some copies of the image at different resolutions. These copies will be blurred by different Gaussian filters.

$$L(x, y, k\sigma) = G(x, y, k\sigma) * I(x, y) \tag{4.1}$$

The images are grouped by octaves (an octave being a level of the pyramidal tree). The resolution is divided by 2 with each consecutive octave : for the difference of Gaussians, it's equivalent to multiply σ by 2.

$$D(x, y, \sigma) = L(x, y, k_i\sigma) - L(x, y, k_j\sigma) \tag{4.2}$$

Then, for each octave, the difference between every two consecutive blurred images (by the filter at scale $k * \sigma$) is computed. The remaining objects on the image obtained are the ones which size is included between σ and $k * \sigma$. The parameter k is a constant defined according to the precision wanted.

Thanks to the different resolutions, for each octave the difference result will keep higher and higher object that will allow us to detect approximately all the sizes of important features on the image. With all these difference calculated, the algorithm will take the maxima of each one as key-points.

After doing that, the algorithm must discriminate and specify the coordinate of a part of the key-point. Indeed, using different resolutions give some inaccurate coordinate so it's necessary to make an interpolation to obtain the coordinate corresponding to the original image for key-point extracted from the most reduced resolutions images. It is necessary to remove some points too. The points which have not enough contrast in comparison to the

Figure 1: Difference of Gaussians illustration (example from wikipedia)

others are removed, same thing for the points that are on ridges (these points are really unstable and could move or disappear for many reasons so it's better to remove them).

When these operations have been computed, we obtain a set of key-points that characterize the image. This method allows to obtain the key-points to characterize an image without taking a predefined set of points. So the set of points obtained is potentially lighter than the one obtained by the dense grid method and it is more accurate because only important points are saved, whichever their location.

4.1.2 Dense Grid

The dense grid method is the easiest way to extract key-points. The image has first to be divided into k sub-images and the intersections of the sub-images' outlines become the key-points.

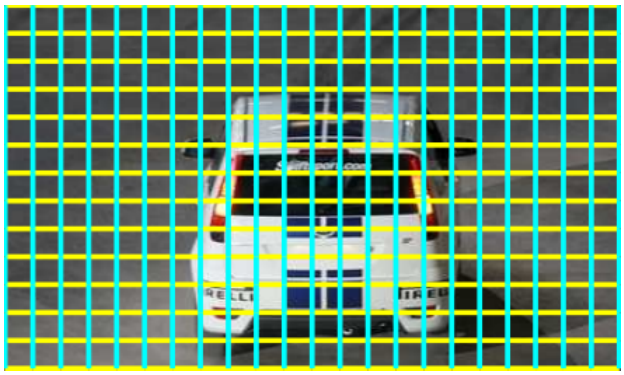


Figure 2: Dense grid

4.2 Descriptors

The detection of key-points in images is increasingly used because it helps to do many tasks for example recognition, images assembly, 3D modeling, image indexing, video tracking etc. The key-points extraction in one image allows characterizing this image. By comparing the key-points of two images we can deduce if they have common information or not.

4.2.1 SIFT

SIFT (scale-invariant feature transform) is an algorithm used in computer vision for detection and identification of similar elements between different digital images. The principal method proposed by the author David Lowe is to calculate the SIFT descriptors on studied pictures. These descriptors are numeric information which derived of local analysis of the image and they characterize the visual content of this image so that this one is independent of the scale, the framing, the angle of observation and the luminosity.

- Orientation assignment: On the base of local image gradient detections each key-point detected is assigned to one or many orientations. Insofar as descriptors are calculated from these orientations, it is important to safeguard the invariance of these descriptors to the rotation because whatever the orientation we must obtain the same descriptors using the same image.

  For example with a key-point (x_0,y_0,σ_0), the calculation is done on the Gradient of the pyramid L(x,y,σ_0) which factor is nearest the scale factor of the point. In this way the calculation is also independent to the scale variance. With the symmetric finite difference, the gradient and the amplitude are calculated for each position around the key-point. The calculation of these two factors is given by the following equations:

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}$$

$$\theta(x,y) = atan2(L(x,y+1) - L(x,y-1), L(x+1,y) - L(x-1,y))$$

  A histogram with 36 intervals is realized on the vicinity and each interval covering an angle of 10 degrees.

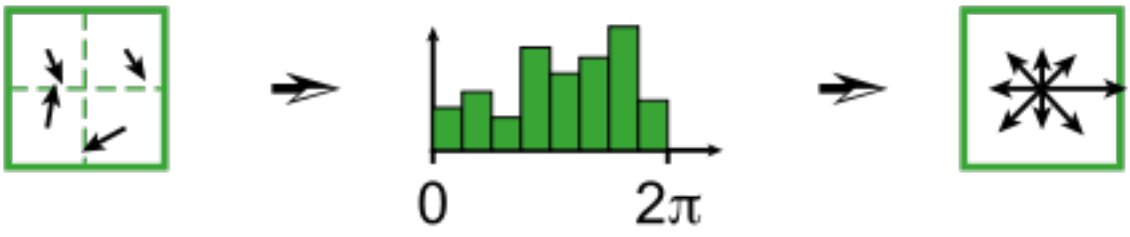


Figure 3: SIFT histogram construction

On one hand, the histogram is moderated by a circular Gaussian window with a factor equal to 1.5 times of the scale factor of the key-point. On the other hand by the amplitude of each point. The peaks in this histogram correspond to the dominant

orientations. All dominant orientations allow to have at least 80% of the maximum value are taking in consideration and other additional key-points are detected. These new key-points detected are only different by the principal orientation.

- key-point descriptor: In this stage, a descriptor vector is created for each key-point detected such the descriptor is distinctive and invariant to the remaining variations like illumination, 3D viewing, etc. This stage is made on image which is more near the scale of the key-point scale.

  The local system coordinates is modified for guarantee the invariance to the rotation by using an angle rotation equal to the orientation of the key-point in reverse direction. Then, an area of 16x16 pixels is taken around the key-point; subdivide in 4x4 zones of 4x4 pixels each one. A histogram including 8 intervals is calculated on each zone. At each point of the zone, the orientation and the amplitude are calculated like previously. The orientation determines the interval to increment in the histogram which done by double weighted by the amplitude and by a center window Gaussian on the key-point of parameter 1.5 times of the scale factor of the key-point.

  After that, the 16 histograms with 8 intervals each one, are concatenated and normalized. In object to reduce the sensibility of the descriptor to the changes of the luminosity, the values are fixed to 0.2 and the histogram is calculated again for finally give the descriptor of the key-point of 128 dimension.

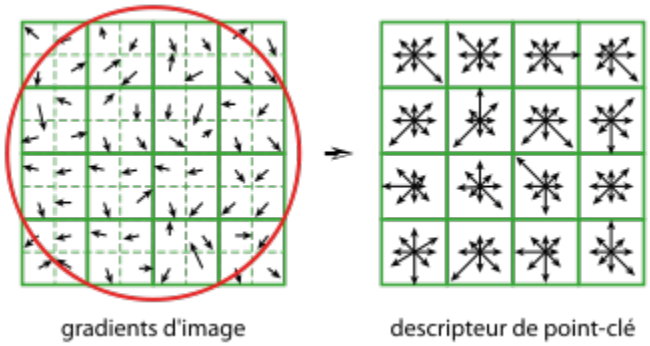


Figure 4: SIFT gradient magnitude and orientation computation

4.2.2 SURF

SURF for Speeded Up Robust Features is a descriptor and algorithm of feature detection. It is also used in the sector of computer vision and image understanding for objects detection or 3D reconstruction.

SURF is partially inspired by the SIFT descriptor but is faster to process. According to the authors, it is also more robust for different image transformations.

SURF is based on sums of 2D Haar wavelets responses and uses efficiently the whole images.

4.2.3 Opponent SIFT

The opponent SIFT descriptor is an algorithm using the same method as the standard SIFT descriptor except for the fact that it calculates three descriptors for each Key-points, obtained from the color opponent channel and defined as

$$O_1 = \frac{R-G}{\sqrt{2}}, O_2 = \frac{R+G-2B}{\sqrt{6}}, O_3 = \frac{R+G+B}{\sqrt{3}}. \tag{4.3}$$

The opponent SIFT describes the three opponent color spaces : the two first channels O_1, O_2 contain some intensity information, but they are sensible to changes in light intensity. The last channel will contain the intensity information.

The strength of this method is that it uses a color space, and we can see directly information of that with the algorithm of the SIFT descriptor. The weakness is that the color space used is the RGB one.

4.2.4 C_2O

C_2O feature is a color descriptor which aim is to characterize an image by its color and texture characteristics. Indeed, the descriptors previously presented are satisfying to characterize images in gray or color levels but are pretty weak for highly textured images like nature pictures. That's why the university of Poitiers has worked on a descriptor named C_2O (Color Contrast Occurrence), based on a vector including texture and color information. To compute it, there is two steps to follow : the calculation of the Color Contrast Occurrence Matrix and the extraction of the feature (descriptor) from the matrix.

4.2.4.1 The Color Contrast Occurrence Matrix

To compute this descriptor, the aim is to calculate a matrix which represents each key-point by a probability. To compute it, the image has to be used in a color space which is able to separate best the color and the luminance information. Tests have shown that the CIE L* a* b* space separate "has minimum correlation between luminance and chrominance information" ("Color Contrast Occurrence, a full vector for color and texture"). So the image is passed in the CIE L* a* b* color space before the calculation of the descriptor. Before the calculation of the L* a* b* space, its needed to transform our image through the XYZ space that is a perceptual space based on a linear transformation of the RGB space.

$$A = \begin{pmatrix} X_r & X_g & X_b \\ Y_r & Y_g & Y_b \\ Z_r & Z_g & Z_b \end{pmatrix}$$

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = A * \begin{pmatrix} R \\ G \\ B \end{pmatrix} \tag{4.4}$$

With A a matrix which coefficients are depending on the chosen standard illuminance. Once this XYZ space has been computed, we have to compute the following transformation to get our image in the L* a* b* space.

$$L^* = \begin{cases} 116 * (\frac{Y}{Y_0})^{\frac{1}{3}} - 16 & si \frac{Y}{Y_0} > 0.008856 \\ 903.3 * (\frac{Y}{Y_0}) & si \frac{Y}{Y_0} < 0.008856 \end{cases} \tag{4.5}$$

$$a^* = 500 * \left[f(\frac{X}{X_0}) - f(\frac{Y}{Y_0}) \right] \tag{4.6}$$

$$b^* = 200 * \left[f(\frac{Y}{Y_0}) - f(\frac{Z}{Z_0}) \right] \tag{4.7}$$

After that, we can calculate the descriptor. The principle is simple : for each key-point, we have to calculate the probability to have a specific color difference between two pixels separated by a spatial vector. The color difference is calculated by considering the angles created by the L* a* b* representation and a perceptual distance.

So we define the color contrast occurrence value as : $\overrightarrow{\Lambda(C_i, C_j)}$

$$\overrightarrow{\Lambda(C_i, C_j)} : prob(\overrightarrow{\Lambda(C_i, C_j)} = \overrightarrow{\Lambda_\chi} \tag{4.8}$$

$$with \ \|\overrightarrow{\Lambda(C_i, C_j)}\| = \Delta E_\chi \tag{4.9}$$

$$and \ (\overrightarrow{Oa}, \overrightarrow{c_i c_j}) = (\alpha, \beta) = \overrightarrow{\Lambda_\chi} \tag{4.10}$$

This computation gives us a cloud of point which characterizes the key-point by its color and texture neighborhood (see below).

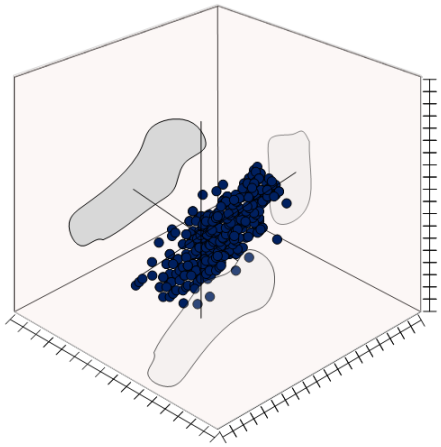


Figure 5: Color Contrast Occurrence Matrix

On the figure shown above, we can see an example of the cloud of points we expect to obtain. There are two things which characterize the image :

- the size and the shape of the cloud characterizing the texture around the key-point.

- the projections on the three plans of the representation which characterize the density of the cloud of points.

4.2.4.2 The Color Contrast Occurrence feature

With the cloud of points obtained by the computation of the Color Contrast Occurrence matrix, we have a 3 dimensional representation of our key-point. To reduce the quantity of data to store and to facilitate the distance calculation, we have to represent this matrix by at least a 2-dimensional feature. To do that, the solution used is to realize a spherical quantization on the cloud of point to have a histogram which will represent our key-point on two dimensions. Mathematically, this quantization is expressed as follows :

$$Sig_{C_2O}(I) = h_{\Delta i \alpha j \beta k)} = prob(\Delta_i \leq \|\overrightarrow{\Lambda(C_i, C_j)}\| < \Delta_j + \Delta E_{step}) \tag{4.11}$$

$$and\ \frac{2\pi}{n_\alpha} j \leq \alpha < \frac{2\pi}{n_\alpha}(j+1) \tag{4.12}$$

$$and\ 0 \leq \beta < \frac{2\pi}{n_\beta}(k) \tag{4.13}$$

Each sphere will include a number of points of the cloud, but to have a better distribution, each sphere will be split in some part as shown below :

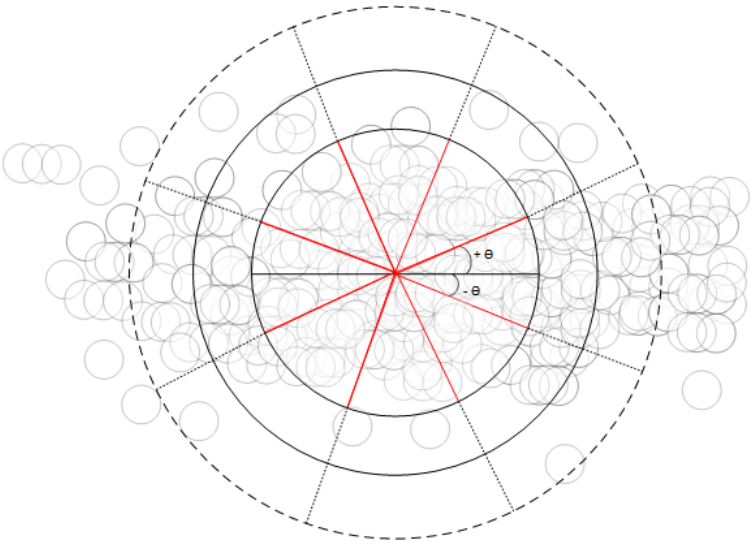


Figure 6: Spheric quantizaton

Here we can see a sectional view of our spherical quantization. Each sphere is divided by n parts as shown above, and the number of points in each part are concatenated one by one in the description vector (quarter after quarter and sphere after sphere).

5 Work achieved

5.1 Process flow

What we call process flow in this project is the main function of the project. Indeed in that function we have to manage almost all the others function, and more precisely features function, classification function, test and validation function, or create or manage the directory files. This program is gives the possibility to switch between the different descriptors (SIFT C_2O) as we want. In this part we will present how the sub-functions inside the process flow work.

5.1.1 Workspace

The aim of the Process flow function is to allow the user to specify his workspace, and choose where he wants to write the results obtained by the software. Here we can see how the folder will be organized by using this function:

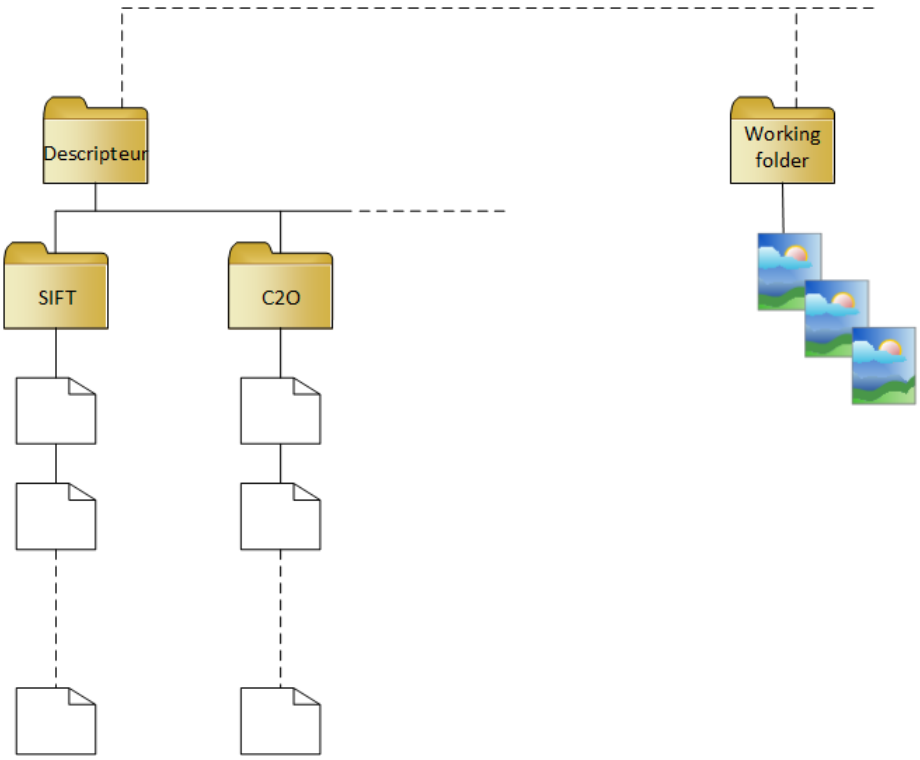


Figure 7: Organization of the workspace

To sum up : we create a folder named after the descriptor, if he doesn't exist already, in this folder we create as much folders as the user wants. For this he only has to enter the name he wants in the parameter *name_desc*.

5.1.2 Processing

The second goal of this function is to compile the whole processing, so if the user wants to compile one particular function in the software he should not use the function *Process_flow*. In this function we will found this line-up:

- Creation of folders and path to the workspace
- Calculation of descriptors and link with IDs for the training database.
- Writing descriptors in text files.
- Calculation of K-means.
- Signature design for whole images.
- Classification of the test database using the K-nn method.

The schedule of the project did not allow us to optimize all the functions, as it can be seen with the K-means function which takes quite long to process. In order to solve this issue, and if you want to test the same signature (same descriptor, and same number of words), we added an option that allows to re-use the last dictionary you calculated.

5.2 SIFT(Scale-Invariant Feature Transform)

SIFT is an algorithm used in the computer vision department for detection and identification of similar elements between different digital images. The main method proposed by the author David Lowe is to calculate the SIFT descriptors on the images to be analyzed. These descriptors are numeric information which derived from the local analysis of the image and they characterize the visual content of this image which results in a descriptor independent of the scale, the framing, the angle of observation and the luminosity.

5.2.1 Stages of the SIFT algorithm

The first stage is to detect key-points on the image. Each key-point is characterized by coordinates x,y on the image and a scale factor σ. After that it's necessary to improve the precision of the key-point localization.

- Scale-space extrema detection
  The gradient scale factor is calculated to smooth the original image. This operation allows to remove details with radius inferior to σ.

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \tag{5.1}$$

Figure 8: Illustration of scale factor gradient (octave 7x7;σ_0 =0; σ_1 =1.6; σ_2 =0.4)

After that, the key-points which have dimension approximately equal to σ are detected by using the Difference of Gaussian(DoG) given by the following equation.

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, k\sigma) \tag{5.2}$$

Where k is the fixed parameter of the algorithm and depends on the precision of the scale desired.

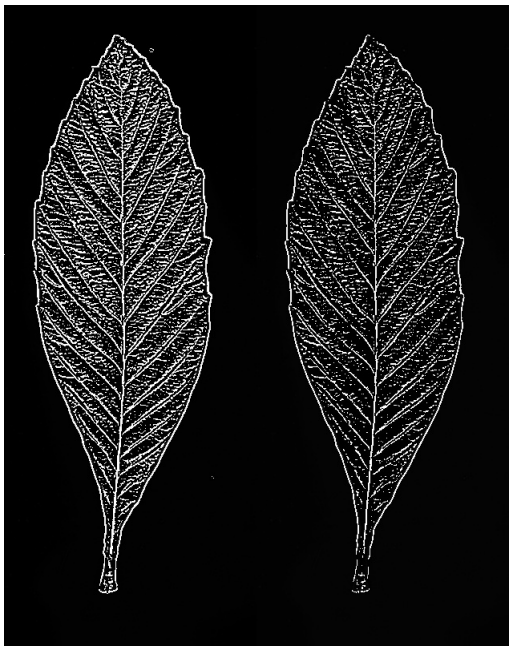


Figure 9: Illustration of the DoG

- Orientation assignment
  On the base of local image gradient detections each key-point detected is assigned to one or many orientations.

  With the symmetric finite difference, the gradient and the amplitude are calculated for each position around the key-point. The calculation of these two factors is given by the following relations:

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}$$

$$\theta(x,y) = atan2(L(x,y+1) - L(x,y-1), L(x+1,y) - L(x-1,y))$$

  A histogram with 36 intervals is realized on the vicinity and each interval covering an angle of 10 degrees.

  The peaks in this histogram correspond to the dominant orientations. All dominant orientations allow to have at least 80% of the maximum value are taking in consideration and other additional key-points are detected. These new key-points detected are only different by the principal orientation.

- Key-point descriptor
  An area of 16x16 pixels is taken around the key-point; subdivided in 4x4 zones of 4x4 pixels each one. A histogram including 8 intervals is calculated on each zone.

  After that, the 16 histograms with 8 intervals each one, are concatenated and normalized. In object to reduce the sensibility of the descriptor to the changes of the luminosity, the values are fixed to 0.2 and the histogram is calculated again to finally give the descriptor of the key-point of 128 dimension.

5.2.2 SIFT test and results

In our project we used the SIFT from the OpenCV library in python. This function allows to detect key-points on images and compute descriptors for each one. The script is adapted to our program in order to calculate descriptors in the whole database of images. Before integrating the script in the process flow, test steps were done in order to verify if the results obtained with the program are satisfying.

Tests consist in applying the sift on image according to different configurations to notice the invariance of the function to the configurations. In a first phase, the tests were done on a basic image design on paint. Here we have the illustration images of the results

Figure 10: Key-points detected before/after luminosity change

Figure 11: Key-points detected after rotation

After that, the test was also done on three images of the database. Three images which look alike are taken in the database and key-points are calculated on each one. These images are matched two by two with the key-points in order to find similarity between images.

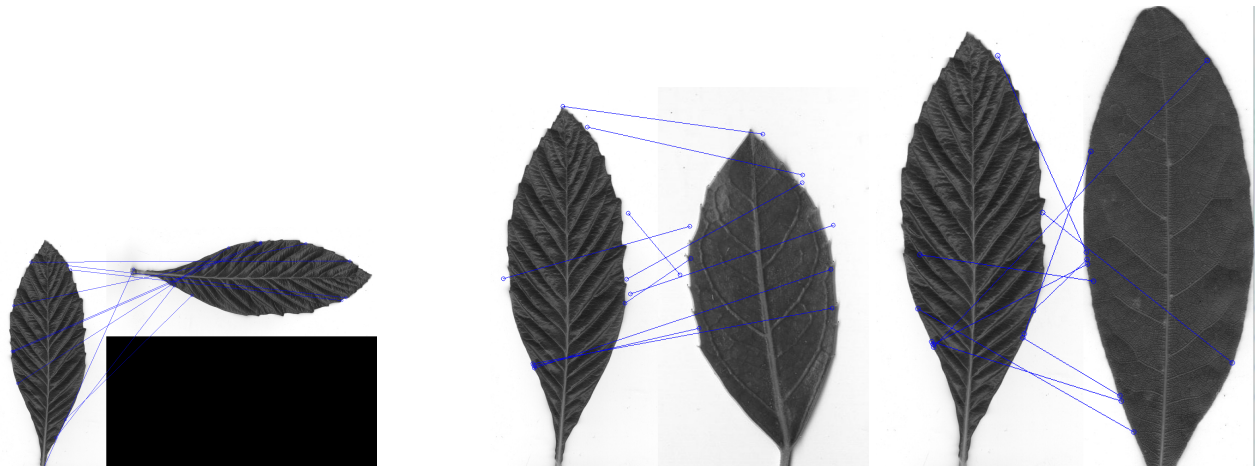


Figure 12: Key-points detected after luminosity change

According to our results obtained through the images above, we concluded that the SIFT descriptor is satisfying and it can be used to compute all descriptors in the whole the database.

5.3 Parallelism test

This test consist in using multiple processors to make the computation faster. It will permit multiple tasks to proceed without waiting for each other. The algorithm consists in decomposing the complete task into independent subtasks and the data flow between them and distributing the subtasks over the processors. With python we have several possibilities to do this. The test was done by using the module multiprocessing because it was the only one available in our version of python. We can see on the following images the elapsed time for the computation before and after using the parallelism.

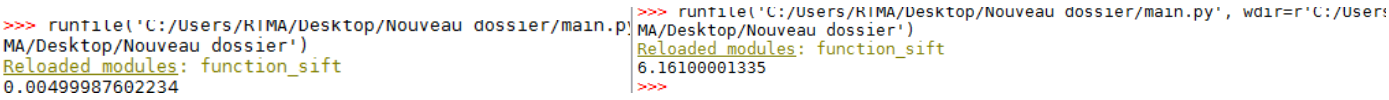


Figure 13: Key-points detected after luminosity change

5.4 Color Constrast Occurence

Here we will see the different steps that we have to follow to compute the C_2O descriptor. There will be three main steps that we will have to study, The transformation through a perceptual space, the computation of the coocurence matrix and the computation of the signature vector.

5.4.1 Transformation through a perceptual space

First of all, we need to pass our image in the L*a*b* space to get directly a representation of it in a space which split the luminance and the chromatic information.

For doing that, we first need to pass the image in the XYZ space and choose the correct standard illuminant A.

$$A = \begin{pmatrix} X_r & X_g & X_b \\ Y_r & Y_g & Y_b \\ Z_r & Z_g & Z_b \end{pmatrix}$$

That parameter depends on the characteristics of the device used to take the picture: it represents how the device interpret real colors. In our case, there are many different devices that have been used to take the different pictures so we will have to make a choice. Some standard illuminant are defined by the illumination of the scene like D50 which corresponds to an "horizon light" contrary to D65 which correspond to a "noon light". Ideally, we will have to make the choice for each image but in front of the huge quantity of images, we had to make a choice. This choice has been to take the Adobe RGB as initial space because it's one of the most used, and to associate it with a D65 standard illuminant.

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = A * \begin{pmatrix} R \\ G \\ B \end{pmatrix} \tag{5.3}$$

After doing this transformation, we can easily transform our image through the L*a*b* space.

$$L^* = \begin{cases} 116 * (\frac{Y}{Y_0})^{\frac{1}{3}} - 16 & si \frac{Y}{Y_0} > 0.008856 \\ 903.3 * (\frac{Y}{Y_0}) & si \frac{Y}{Y_0} < 0.008856 \end{cases} \tag{5.4}$$

$$a^* = 500 * \left[f(\frac{X}{X_0}) - f(\frac{Y}{Y_0}) \right] \tag{5.5}$$

$$b^* = 200 * \left[f(\frac{Y}{Y_0}) - f(\frac{Z}{Z_0}) \right] \tag{5.6}$$

5.4.2 The coocurence matrix

To obtain the coocurrence matrix, we have to calculate an image of the difference of color. For that, the program will take the difference between each point of the image and the point which is at a distance following a Δ vector from it. This calculation will have the same result if we compute the difference between the original image and a copy of it translated following Δ.

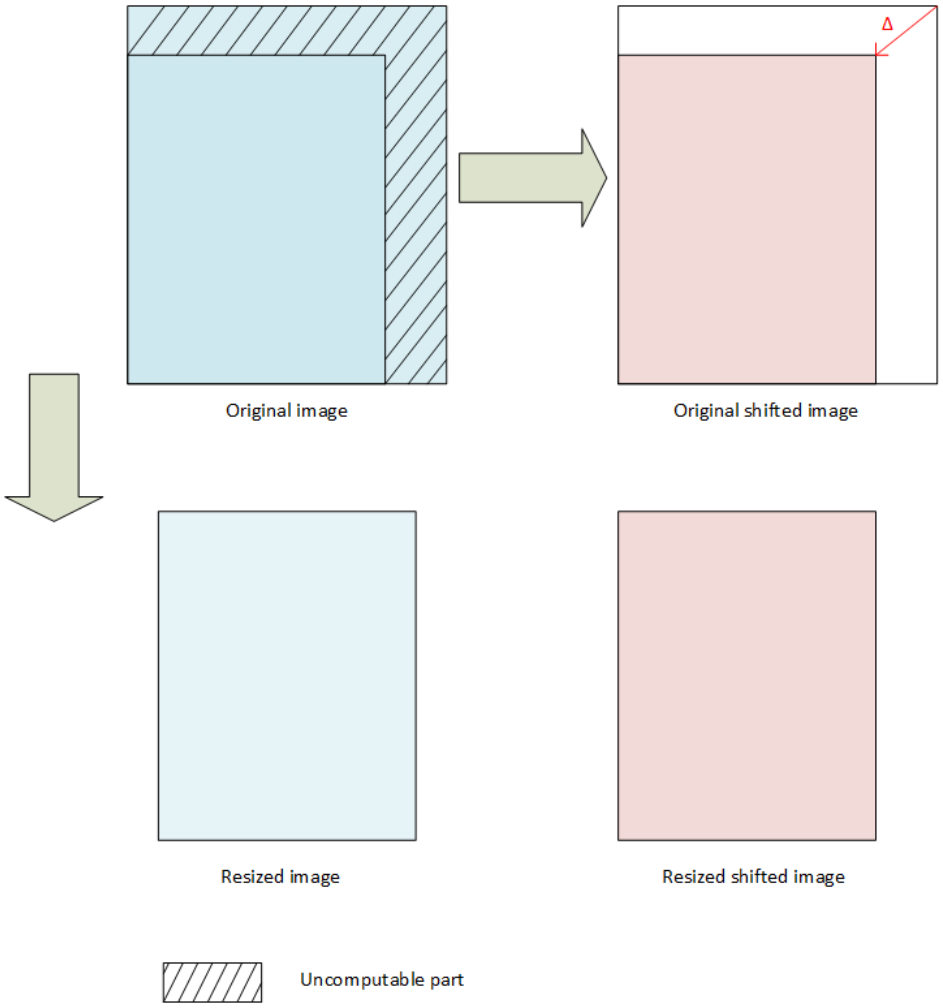


Figure 14: Color Difference illustration

To do that, the program will calculate the equivalence of the translation in horizontal and vertical pixel translation as shown below .

$$\begin{aligned} \sin\theta &= dY/\|\Delta\| \\ dY &= \sin\theta * \|\Delta\| \\ \cos\theta &= dX/\|\Delta\| \\ dX &= \cos\theta * \|\Delta\| \end{aligned} \tag{5.7}$$

The values of $\|\Delta\|$ and θ have to being choose in the aim of get an entire number of pixel.

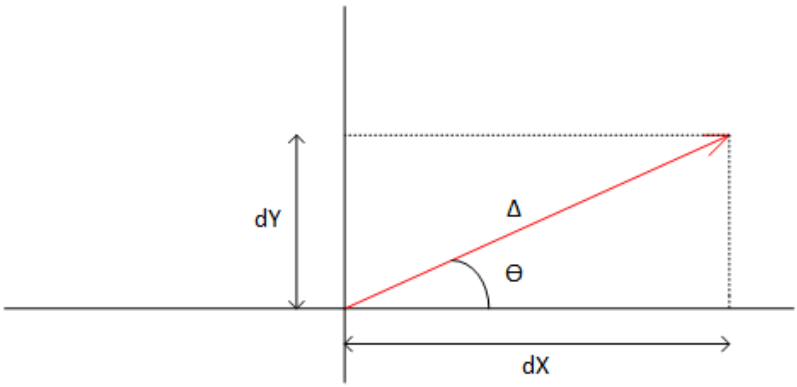


Figure 15: Vector to pixel distance

This computation gives us the C_2O matrix which corresponds to the cloud of point described in the State of art. To validate this part, we have to compare our results with theory. It's known that the L*a*b* space has different component :

- L* which is an achromatic component
- a* which expresses the opposition between the red and the green
- b* which expresses the opposition between the blue and the yellow

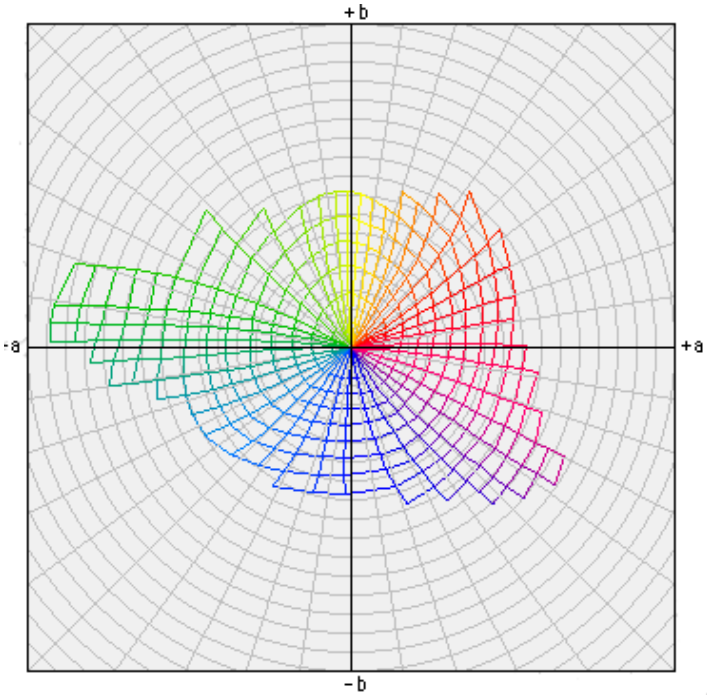


Figure 16: Lab space illustration

We define two test images composed by a succession of lines at different color levels.

So if we are testing our program on images like the two shown below which are containing the two same colors opposed on the same component of the L*a*b* space, the result on the difference must show that :

- For the first one which is in green and red, there will be two points spaced on the a* component but at constant levels on the L* and the b* component.
- For the second one which is in blue and yellow, there will be two points spaced on the b* component but at constant levels on the L* and the a* component.

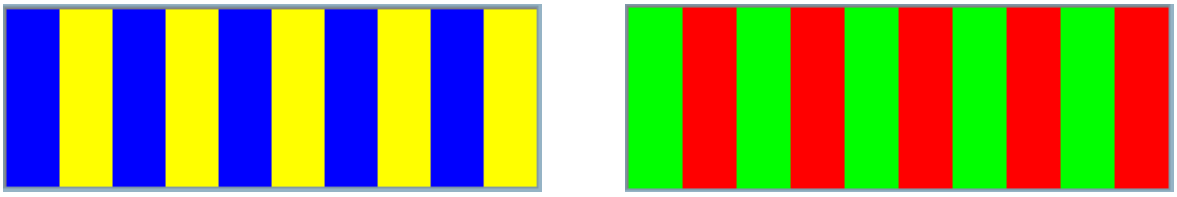


Figure 17: Test images

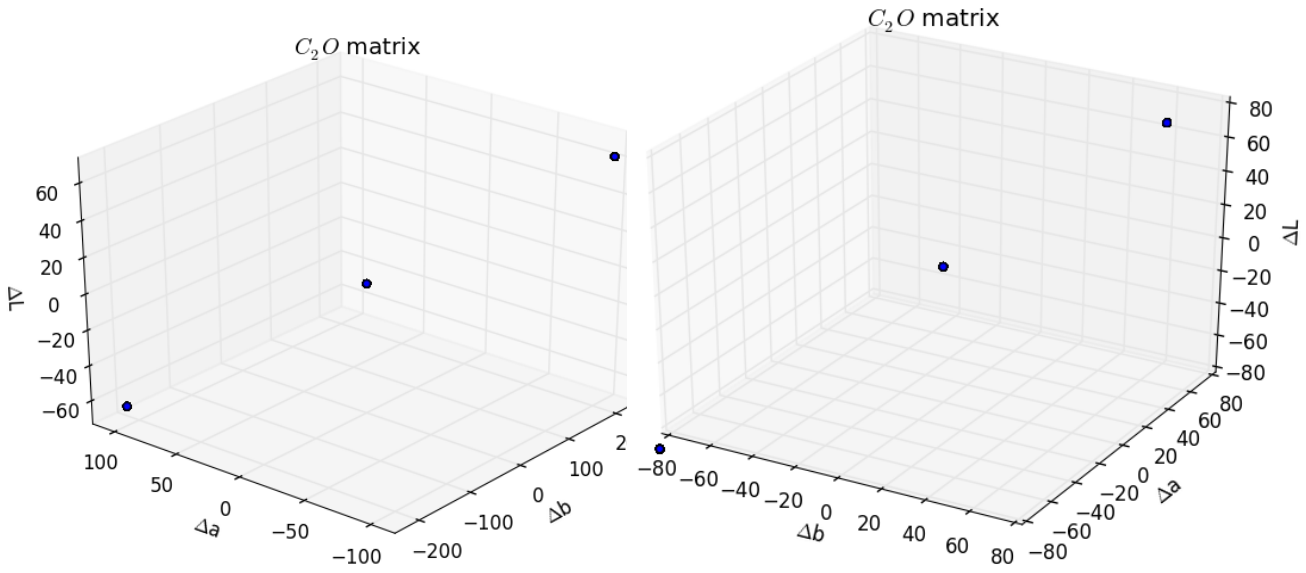


Figure 18: Coocurence matrix obtained from test images

With this step validated by the theory, we will have to test with more colors to verify that we obtain the right number of points on the coocurence matrix for the corresponding number of colors. For each test image, there will be more points as the number of colors on the image is growing because the number of difference of color will grow too.

5.4.3 The signature computing

After doing that, we have to compute the spherical quantization of the probability matrix. For that, we have to transform our difference image from Cartesian coordinates to Spherical coordinates (from these one, the spherical quantization will be easier to compute). So we consider our $L^*a^*b^*$ space like a 3 dimensional repository and for each point, we calculate E the norm of the vector formed by the distance between it and the point (0,0,0), the orientation α formed by the angle between the vector and the a^* plan and finally the orientation β formed by the angle between the vector and the b^* plan.

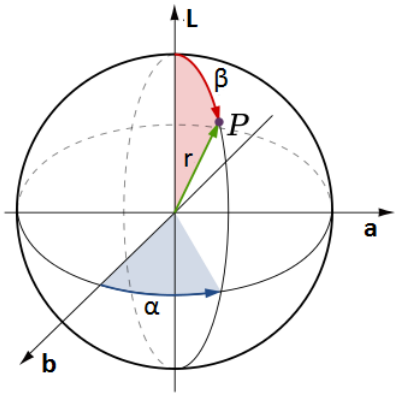


Figure 19: Spherical coordinates

To calculate these coordinates, we have to use the following equations :

$$\begin{aligned} E &= \sqrt{x^2 + y^2 + z^2} \\ \alpha &= \arctan(y/x) \\ \beta &= \arccos(z/r) \end{aligned} \tag{5.8}$$

So for each pixel we obtain :

- Δ E the color distance (approximately equivalent to the contrast)
- $\Delta\alpha$ the orientation of the Λ vector on the a, b plan (that give us an image of the hue)
- $\Delta\beta$ the orientation of the Λ vector on the L, b plan (that give us an image of the luminance)

In that way, we obtain our cloud of points in a spherical repository. Once we get it, we need to calculate our C2O signature by quantifying the cloud of point we obtain by a spherical quantization.

It's computed in three times :

There will be 4 interval of radius for the sphere :

$$\begin{aligned} 0 &\leq \Delta E < 3 \\ 3 &\leq \Delta E < 6 \\ 6 &\leq \Delta E < 9 \\ 9 &\leq \Delta E < infinite \end{aligned} \tag{5.9}$$

Each sphere will be split by α interval to concentrate the information as shown below in the sectional view following the (a^*, b^*) plan. Each α interval will measure $\Delta\alpha = 360/8$

This quantization has to be done for each value of β interval which will measure $\Delta\beta = 180/8$.

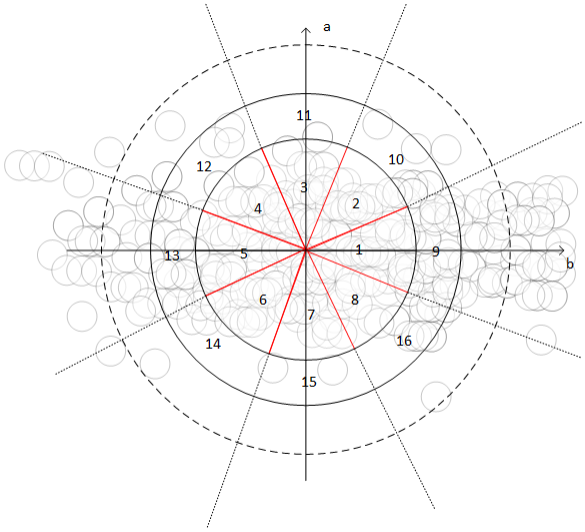


Figure 20: Spherical quantization following α and E

Once this quantization is done, the signature can be constituted by concatenate the values in a vector following a spiral for each value of β.

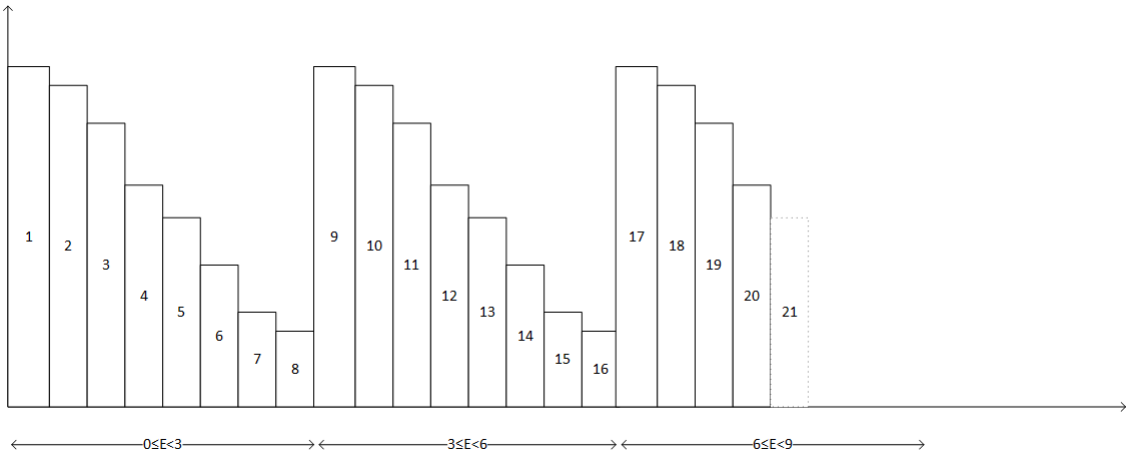


Figure 21: Spherical quantization following α and E

The histograms obtained for each interval of β will be concatenated to obtain the whole signature.

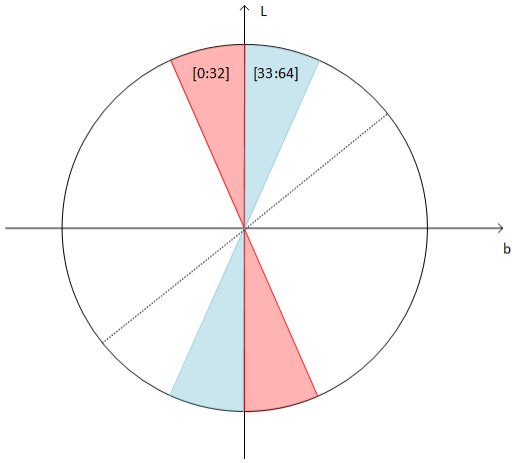


Figure 22: Spherical quantization following β

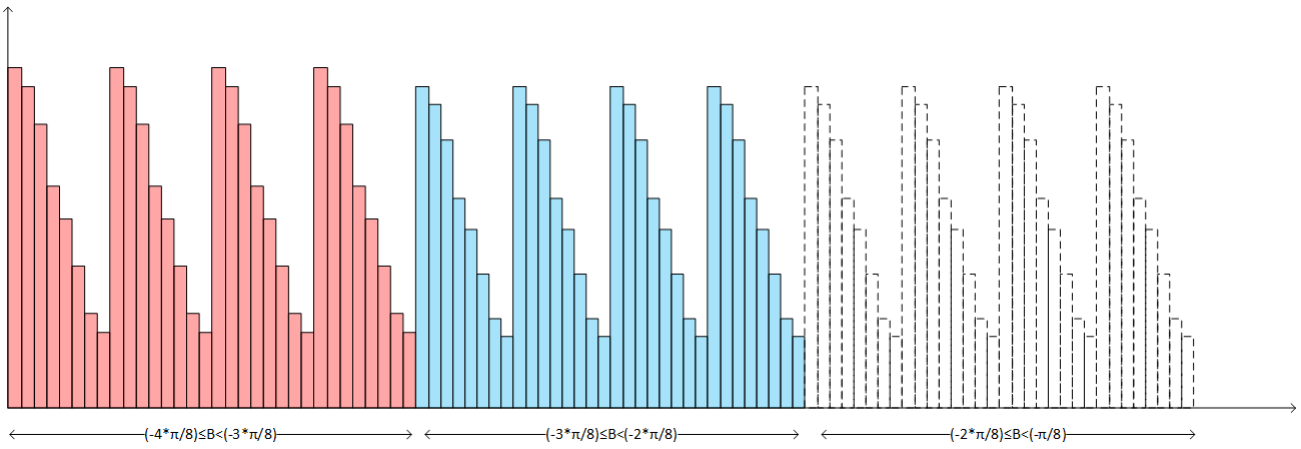


Figure 23: Spherical quantization following β

In this way, we obtain one unique vector of 256 values to describe the image.

5.4.4 Validation step

Once these functions are computed, the most important step is to validate the good operation of all the parts. So in this part we will firstly demonstrate the functioning of the transformation through the L*a*b* space, and after we will talk about the spherical quantization. The functioning of the transformation in spherical coordinates will not be covered here because the test to validate it is just about making the inverse operation and verify is we get the same data..

5.4.4.1 Validation of the transformation through L*a*b* space and color difference

First, we have to choose a reference to compare our results with. The solution chosen is to compare the results of our function with those of the Bruce Lindbloom website's calculator which is a well know reference in image processing.

We have chosen to work with the AdobeRGB space because it's one of the most used. To compute correctly the L*a*b* space, we have to apply a γ factor to make the operation called inverse companding. This correction is made because we consider that the RGB space, since it is captured by actual sensors, is non linear because of the inequality of energy between low and high luminosity levels. The γ inverse companding factor to apply is 2,2 and the standard illuminant is D65. We can compare our results with the Bruce Lindbloom website's calculator to verify that we obtain the right results.

Here we will make a first test with only 2 colors, the red and the green.

$$\begin{aligned} RGB =&[255, 0, 0] \;\; / \;\; L^* = 61.4272 \;\; a^* = 89.5619 \;\; b^* = 75.1487 \\ RGB =&[0, 255, 0] \;\; / \;\; L^* = 83.3027 \;\; a^* = -137.9737 \;\; b^* = 90.8299 \end{aligned} \tag{5.10}$$

Because of the construction of the image on which we are testing our program, there will be only two different values of color difference on each component, so by calculating and comparing it with the theoretical values, we could validate our color difference computation:

ΔL_1 = -21,8755 Δa_1 = 227,5356 Δb_1 = -15,6811
ΔL_2 = 21,8755 Δa_2 = -227,5356 Δb_2 = 15,6811

Here we will make a second test with Yellow and blue.

$$\begin{aligned} RGB = [255, 255, 0] \;\; / \;\; L = 97.0132 \;\; a = -22.5787 \;\; b = 105.3055 \\ RGB = [0, 0, 255] \;\; / \;\; L = 32.9786 \;\; a = 80.3051 \;\; b = -109.3824 \end{aligned} \tag{5.11}$$

The same way it has been done for the first test, we will compare our result of color

difference with the theoretical ones.

ΔL_1 = 64,0346 Δa_1 = -102,8838 Δb_1 = 214,6879
ΔL_2 = -64,0346 Δa_2 = 102,8838 Δb_2 = -214,6879

These results are exactly the same obtained with our software program so we can consider that our L*a*b* transformation and our color difference computation are valid.

5.4.4.2 Validation of the spherical quantization

Once we could be sure that our C_2O matrix is the good one, we have to validate the operating of the spherical quantization which must extract the signature of the image from the C_2O matrix.

For these tests, we will set the number of intervals chosen on each component : there are 4 intervals on ΔE, 8 intervals on $\Delta\alpha$ and 8 intervals on $\Delta\beta$. So we will obtain a signature of 256 values to describe the image. To validate our spherical quantization, we have to create a matrix in spherical coordinates in which we will fix all the values. The first test we will do is to fix our values to have all the points of our matrix in only one interval of α. By doing that, we will be sure that the quantization is good because we will produce the values especially to be in one only interval of the final signature/histogram.

For example here we want to have all the values in the first interval of the histogram.

So the values of ΔE has to been fixed between 0 and 3, values of $\Delta\beta$ has to be between $\frac{-2\pi}{16}$ and $\frac{2\pi}{16}$ and the values of $\Delta\alpha$ has to be between $\frac{-8\pi}{16}$ and $\frac{-6\pi}{16}$ (as show below)

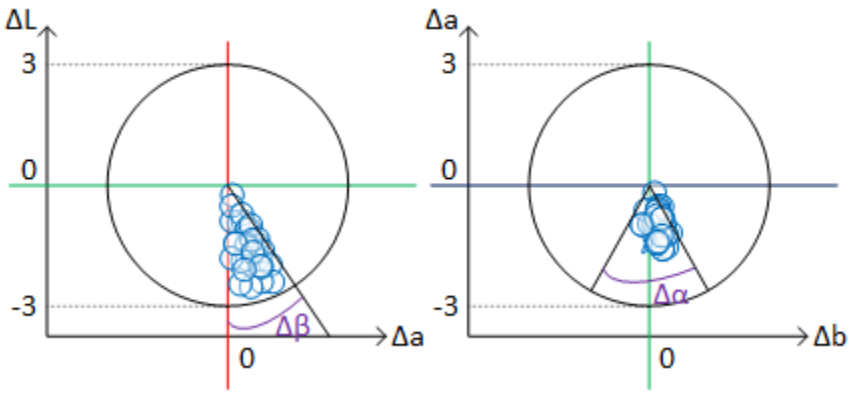


Figure 24: Quantization validation : point generation example

So to do that, we generate these values by using a random function weighted by the right parameters and we obtain the following matrix:

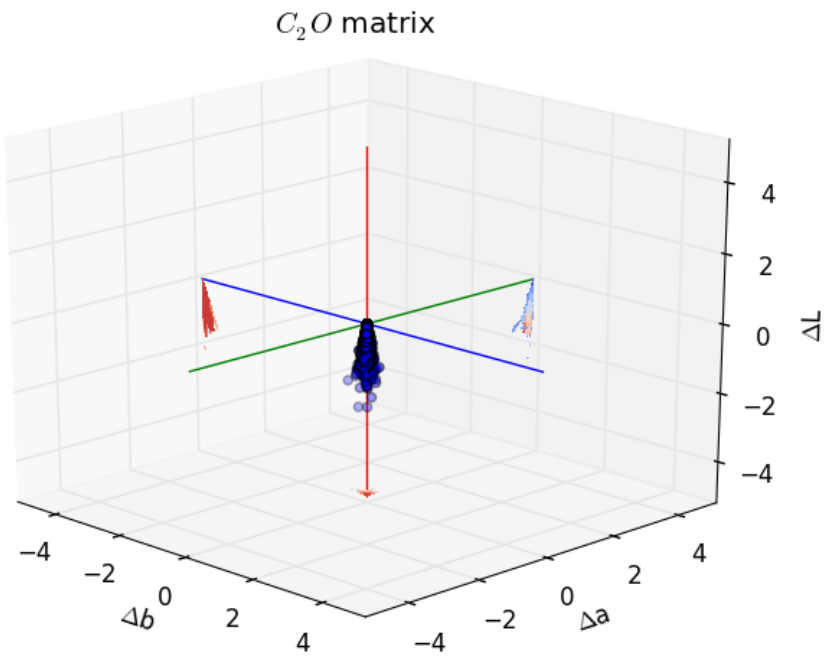


Figure 25: Quantization validation : point generation example

Beginning with the same formulas, we are now able to write a general process to generate values in one only interval of the quantization following the calculation describe below:

$\Delta E = \|(random(30, 30))\|$*3
$\Delta E = \Delta$ E + (3* NbInterE)
$\Delta\alpha = \|(random(30, 30))\|* \frac{\pi}{8}$
$\Delta\alpha = \Delta\alpha + (NbInter\alpha * \frac{2*\pi}{8})$
$\Delta\beta = \|(random(30, 30))\|*\frac{\pi}{8}$
$\Delta\beta = \Delta\beta \ \ ((4\text{-NbInter } \beta) * \frac{\pi}{8})$

For example, if we set NbInterE to 0, $NbInter\alpha$ to 4 and $NbInter\beta$ to 0, we must have all the 900 (30*30) values on the 4^th interval (8*4*0 + 8*0 + 4).

Here we can see the matrix that we obtain with these parameters:

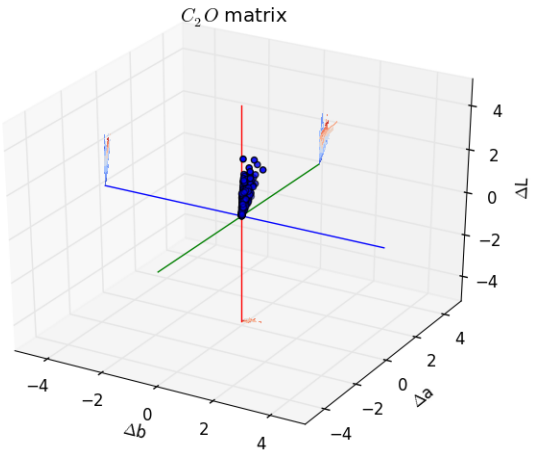


Figure 26: C_2O matrix 4^{th} interval

So we have the following signature:

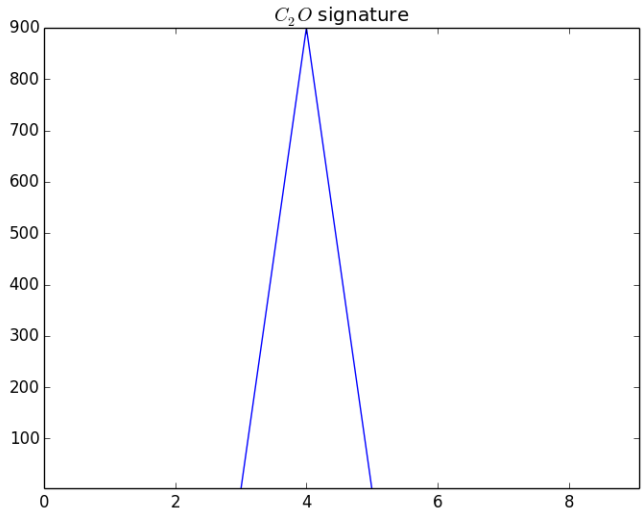


Figure 27: C_2O singnature 4^{th} interval

All the values are on the 4^{th} interval.

Let's test it for another interval. If we set NbInterE to 1, $NbInter\alpha$ to 3 and $NbInter\beta$ to 1, we must have all the 900 (30*30) values on the 43^th interval (8*4*1 + 8*1 + 3).

The following matrix is obtained:

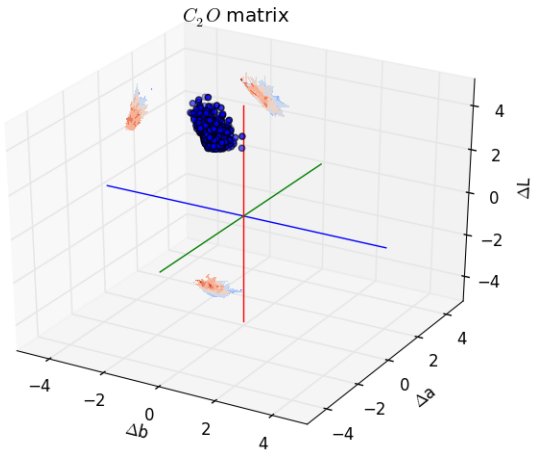


Figure 28: C_2O matrix 43^{th} interval

So we obtain the following signature:

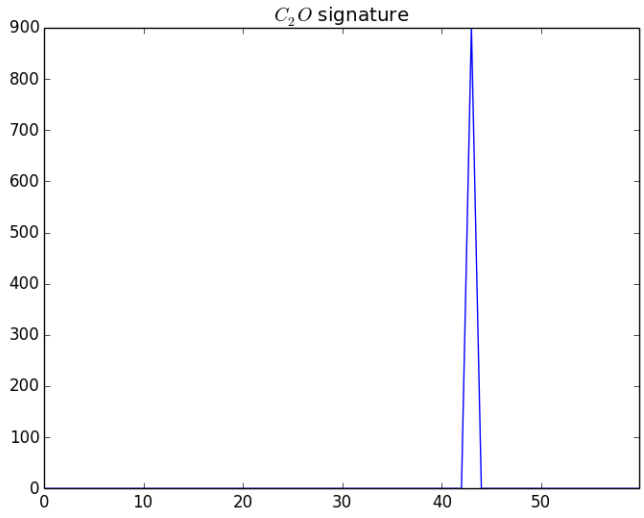


Figure 29: C_2O signature 43^{th} interval

All the values are on the 43^{th} interval.
Considering these results, we can validate the operation of our spherical quantization.

5.5 Metadata

The image database come with xml files containing associated with every pictures respecting the format below :

```
<?xml version="1.0" encoding="UTF-8"?>
<Image>
    <ObservationId>37497</ObservationId>
    <MediaId>2</MediaId>
    <Vote>3</Vote>
    <Content>Flower</Content>
    <ClassId>780</ClassId>
    <Family>Asteraceae</Family>
    <Genus>Erigeron</Genus>
    <Species>Erigeron karvinskianus DC.</Species>
    <Author>herve goeau</Author>
    <Date>2012-11-26</Date>
    <Location>St Leu Les brules de St Leu</Location>
    <Latitude>-21.10527</Latitude>
    <Longitude>55.38092</Longitude>
    <YearInCLEF>PlantCLEF2015</YearInCLEF>
    <ObservationId2014/>
    <ImageId2014/>
    <LearnTag>Train</LearnTag>
</Image>
```

Figure 30: *metadata of the picture number 2*

Among those, only the following can be useful :

- ObservationId: the plant observation ID from which several pictures can be associated, represents a unique environment (same plant, same day, same author);

Figure 31: *One observation of one individual-plant observed the same day by a same author involving several pictures with the same Observation ID*

- content: part of the plant represented in the picture : "Leaf", "Flower", "Fruit", "Stem", "Entire", "Branch", can be used to in the prediction process;
- ClassId: the class number ID that must be used as ground-truth. It is a numerical taxonomical number used by Tela Botanica, unique to each species;
- Species: the species names (containing 3 parts: the Genus name, the Species name, the author(s) who discovered or revised the name of the species);
- Family: the name of the Family, two levels above the Species in the taxonomical hierarchy used by Tela Botanica;
- author: name of the author of the picture, will be of use to balance the metrics ;

5.5.1 metadata retrievers

Every picture of the dataset is bundled with a XML file providing related data. Retrieving relevant data from those files was necessary in order to associate the results of the descriptors with what they represent. Metadata can also be used to limit the database to a un unique family, to certain species or to a specified content(branch, fruit, etc).

For this purpose we created a function which look into those files to return the observationId, classId, author, etc.

5.5.2 relevant metadata storing

At first we expected to recreate a single XML file containing only the relevant metadata so that we don't have to rescan the data bank each time we need something. But the XML format didn't turn to be very practical to use since when reading the files it was not possible to retrieve the parent from the nodes. And without using a tree structure, the XML format loses some of its appeal.

We switched for a simple text file providing with each lines the observationID, the ClassID, the file related and the author.

5.5.3 XML handling

For some reasons, at the beginning we had decided to use XML files files to store the keypoints and their coordinates. The idea was to have something like that :

```
<?xml version="1.0" encoding="utf-8"?\>
 <keypoints>
    <kp1>
        <x></x>
        <y></y>
        <xs:descriptor></xs:descriptor>
    </kp1></kp2>
        <x></x>
        <y></y>
        <descriptor>
        </descriptor>
    </kp2>
 </keypoints>
```

The idea was discarded later as it turned out there was no benefit with this and a simple text file containing at each line

x y descriptor was more than enough and easier to do.

5.6 Quantization

The Quantization is the step we will use to determine which image correspond to which species. For that we will first use the bag of words model to simplify the signature of whole images and then we will use the K-nn method for the classification.

5.6.1 Bag of words

In the bag of words method we found two different steps, the calculation of K-means and the design of a signature for the images.

5.6.1.1 K-means

The K-means is a simple method which consists in reducing the number of points, or vectors in our case. The first step is to determinate randomly k centroid vectors and then, with an Euclidean distance (5.12), we attribute the descriptors to whole images to the nearest centroid vectors.

$$\sum_{k=0}^{centroid} \sum_{i=0}^{desc} \parallel x_k - u_i \parallel^2 \tag{5.12}$$

The last stage is an update step, for each centroid vectors we calculate the means of all the descriptors associated to, so we obtain a new centroid vectors. And we applicate this algorithm for x iteration, as defined by the user.

For an application on a cloud of points we obtain this kind of result:

5.6.1.2 Signature

The creation of the signature is the last step of the bag of words method, they consist in assigning to each image a signature in function of the number of word in the image. The graphic representation of this signature is a histogram:

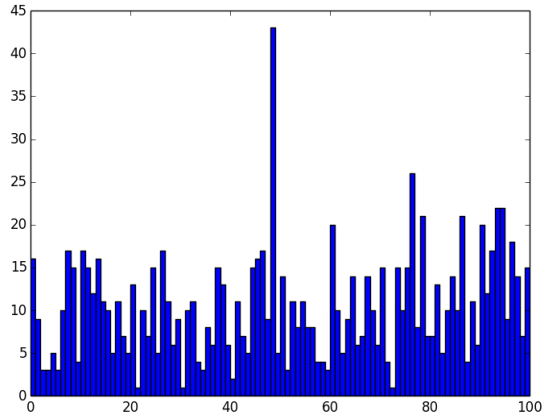



Figure 32: Signature de l'image 128

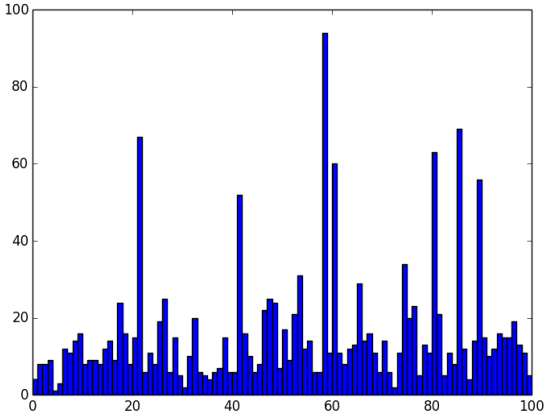



Figure 33: Signature de l'image 132

5.6.2 K-nn

5.6.2.1 The K-nn method

The K-nn method is one of the most simple technique in the classification field. This method consist in associating the data to be classified with the class that corresponds the best to it. It is a supervised learning method : it means that the k-nn has to have a dictionary which contains all the training data sorted by class. To classify a new data, the k-nn calculates the distance between each training data and the new data. After doing that, it will simply takes the 'k' smallest ones, and determine in which class to attribute the new data by taking the one which is the most occurrent in the 'k' smallest distances.

For our project, the data to classify are images and the classification is made by the signature vector extracted by the K-means method. The principle is exactly the same but the distances are calculated on the signature vector. We had chosen two possibilities for the distance calculation : the Euclidian distance and the χ^2 distance which are described below.

- Euclidian distance

$$D_{Euc}(v_1, v_2) = \sum (v_1(i) - v_2(i))^2 \tag{5.13}$$

- χ^2 distance

$$D_{\chi^2}(v_1, v_2) = \sum \frac{(v_1(i) - v_2(i))^2}{(v_1(i) + v_2(i))^2} \tag{5.14}$$

The choice of the type of distance calculation depends on the type of descriptor chosen for the classification. For the SIFT descriptor,the Euclidian distance is appropriate but each descriptor must have one distance which works better than others. That's why the χ^2 distance has been implemented in the program.

5.6.2.2 Validation step

To validate this classification method, the most simple way is to test it on artificial signatures with different constant values. The test to validate is to take as our initial dictionary 7 vectors which we consider to be in 2 different classes. These vectors are full of constant values, they are shown as lines below :

The lowest one (from 0 to 3) are attributed to the first class and the others are attributed to the second one. After that, we simply have to test to classify a signature which is constructed on the same model. When the signature vector is closer to 3 than 6, it has to be attributed to the first class, and to the second when he is closer to 6.

The result obtained by our K-nn function is the one we expected, the function is validated.

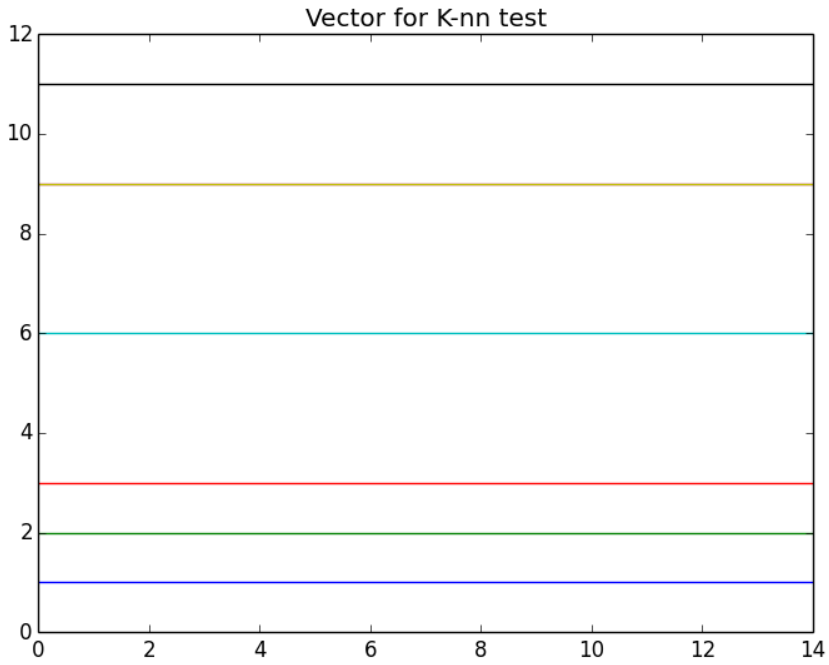


Figure 34: Dictionary of vector for k-nn tests

5.7 Metrics

The first thing to do is to create a file with the results of the run and the related information.

The official run files to be submitted to the challenge have to follow a specified format :

ObservationId;ClassId;rank;score

The ObservationID was explained previously and is here to represent the environment of the picture (being unique to a set of an author, a place, a date and a plant) The ClassID represents the plant supposed to be on the picture by the classification The rank represents the occurrence of the same ObservationID in order to The score is a confidence score of a prediction item (the lower the score the lower the confidence).

Another optional run file, detailing species prediction at image level) could be submitted as below :

test_image_name.jpg;ClassId;rank;score

Since we didn't produce any run we left the score aside.

Once those files produced we can now measure the performance of the run.

The first thing to do is to check if the predicted plants are right and then to retrieve only the correct guesses.

The CLEF challenge offers two metrics :

The primary metric used to evaluate the submitted runs will be a score related to the rank of the correct species in the list of the retrieved species. Each plant observation test will be attributed with a score between 0 and 1 : of 1 if the first returned species is correct and will decrease quickly while the rank of the correct species increases. An average score will then be computed on all test individual plant. A simple mean on all observation test would however introduce some bias. Indeed since the PlantCLEF dataset was built in a collaborative manner, some contributors have provided much more observations pictures than the others. O in order to evaluate the ability of a system to provide correct answers to all users, the mean of the average classification rate per author is used.

1st metric (score observation):

$S1 = \frac{1}{U} \sum_{u=1}^{U} \frac{1}{P_u} \sum_{p=1}^{P_u} S_{u,p}$

U : number of photography authors with at least one image in the test data

$\mathbf{P}_u$ **:** number of individual plants observed by the u-th user

$\mathbf{S}_{u,p}$ **:** score between 1 and 0 equals to the inverse of the rank of the correct species (for the p-th plant observed by the u-th user)

The secondary metric is used with the optional run file providing species determination at the image level. Same as above, in order to avoid the bias brought with the fact that some authors provided many pictures of the same plant, the average classification rate on each plant is used.

2nd metric (score image):

$S2 = \frac{1}{U} \sum_{u=1}^{U} \frac{1}{P_u} \sum_{p=1}^{P_u} \frac{1}{N_{u,p}} \sum_{n=1}^{N_{n,p}} S_{u,p,n}$

U : number of photography authors with at least one image in the test data

$\mathbf{P}_u$ **:** number of individual plants observed by the u-th user

$\mathbf{N}_{u,p}$ **:** number of individual plants observed by the u-th user

$\mathbf{S}_{u,p,n}$ **:** score between 1 and 0 equals to the inverse of the rank of the correct species (for the n-th picture taken from the p-th plant observed by the u-th user)

5.7.1 score results

We didn't manage to evaluate our own scores but the results from the challenge are available to check with.

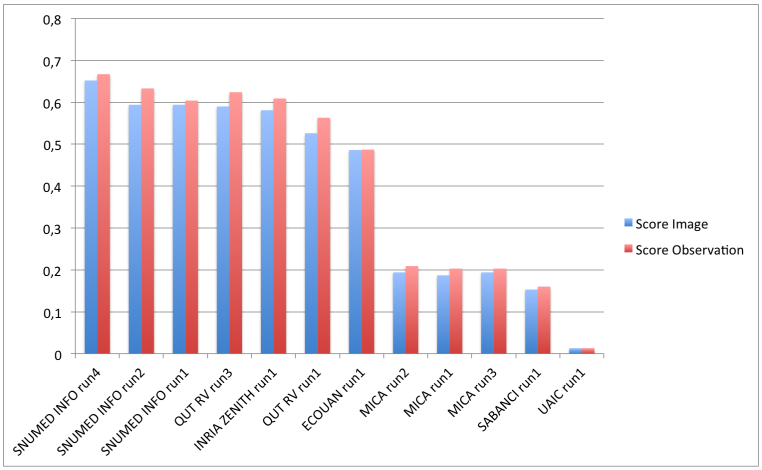


Figure 35: Score 2014

5.7.2 checking the predictions

Once the predictions are done we still have to check whether they are correct or not. To handle this task we created a function storing the metadata and checking if the output classID linked with the filenames.

The right answers are then stored separately in order to be used for the metrics.

5.8 Results

In this part, we will present the results we obtained in this project. We divide this section in two subsections, the first is the procedure we use for testing our descriptors. And the second is the presentation of the results and the comparison of the two descriptors.

5.8.1 Procedure

5.8.1.1 For one image

First, we use the result of the K-means to classify one image using the training database as reference:

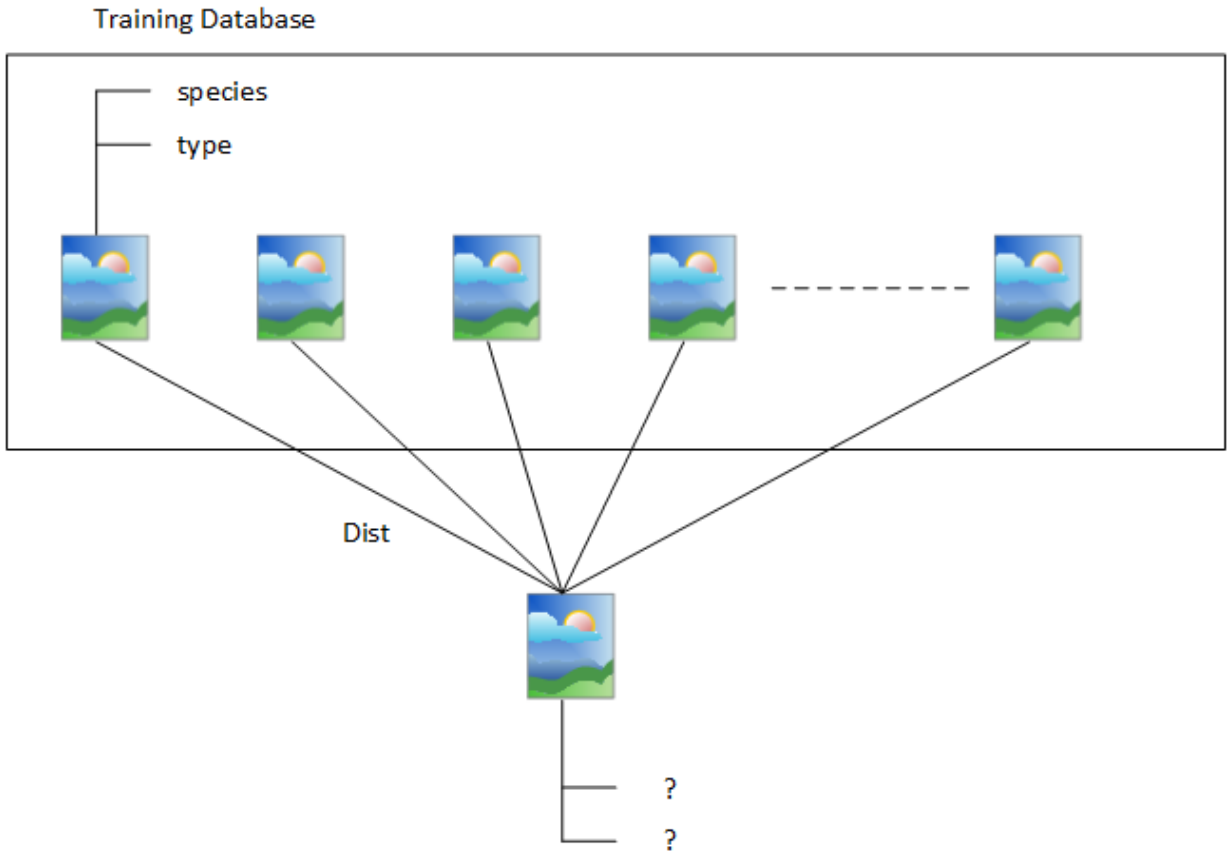


Figure 36: Classification for one picture

For the SIFT descriptor we will compare the Euclidean distance (5.15) and the χ^2 distance (5.16). The comparison of these distances in the C_2O algorithm is made in the K-means algorithm.

After the distance calculation we use the K-nn algorithm to find the species of the picture.

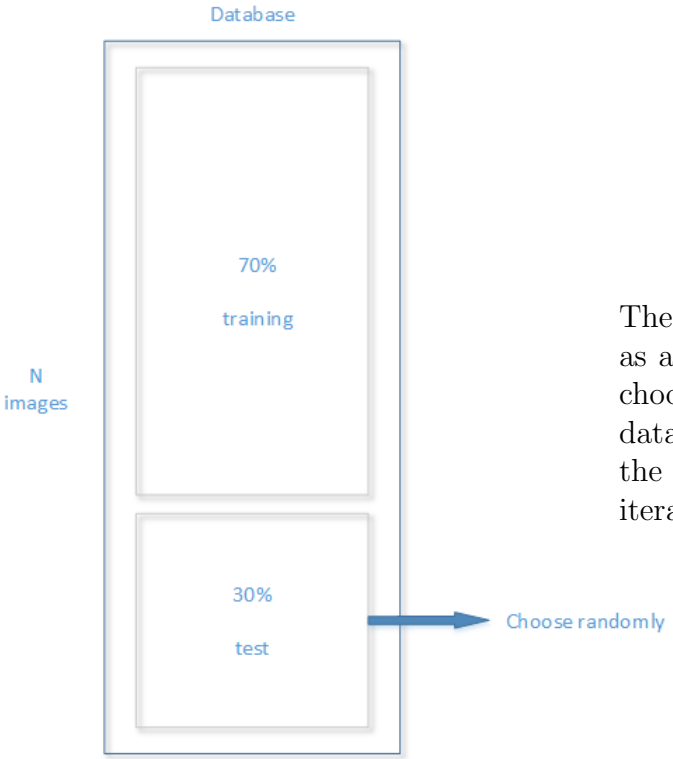
$$T_E(H^1, H^2) = \sqrt{\sum_{i=1}^{n} (h_i^1 - h_i^2)^2} \tag{5.15}$$

$$T_{\chi^2}(H^1, H^2) = \sum_{i=1}^{n} \frac{(h_i^1 - h_i^2)^2}{(h_i^1 + h_i^2)^2} \tag{5.16}$$

5.8.1.2 For a database

For a database we use the same method that we used for one image. For the treatment of a database we can use two different methods:

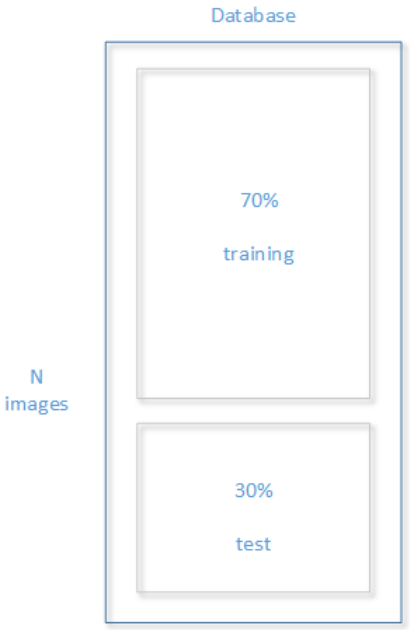
- cross-classification



The difference is that we will take 80% of the images as a training base, and 20% as test. The 20% are choose randomly, but to have a means of the whole database result, we will iterate this process, and the signature of the images choose at the previous iteration can not be in the 20% of test.

Figure 37: Classification for a database

- fixed classification



In this second method, we choose the same training base (70% of the database), and the same test base (30% of the database). This method allows us to see the difference between two different distances, or between the SIFT and the C_2O.

Figure 38: Classification for a database

In scope of respecting our project schedule we have to present results so we will use the second method to have results to compare.

5.8.2 Analysis of result

In this part we will analyze and discuss about the results obtained by the software. In order to get quick results we will take a reduced data-base of 100 images and only 4 species. Here you can see 4 images which represent species:

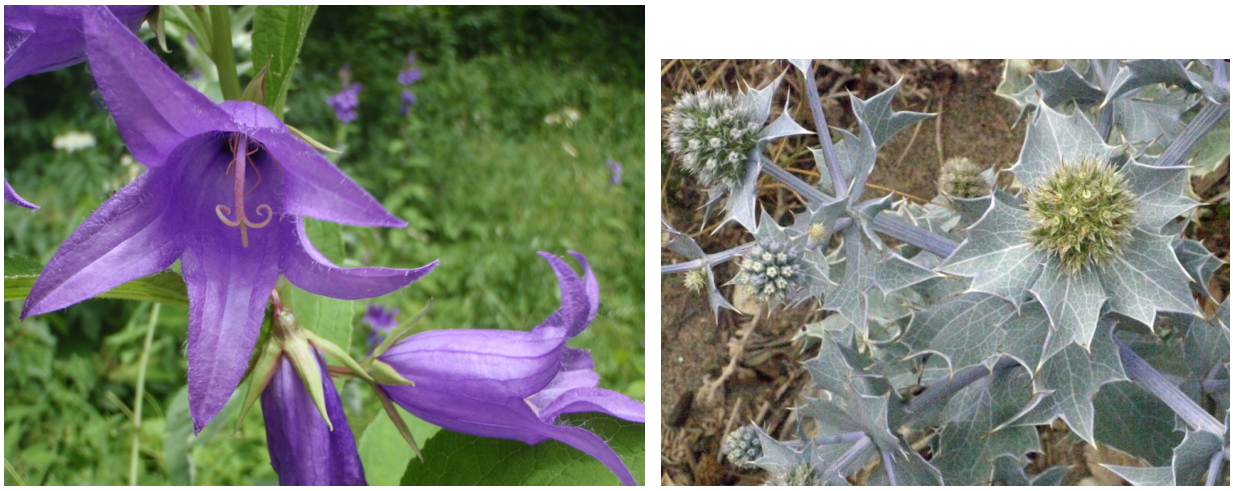


Figure 39: Campanula latifolia L. / Eryngium maritimum L.


Figure 40: Narcissus dubius Gouan / Picris hieracioides L.

5.8.2.1 SIFT

For the descriptors SIFT we obtain that result:

Table 1: SIFT result

ID	Training Base	Test Base	Correct	Accuracy
173	17	8		
1102	22	3		
1889	16	9		
2717	15	10		
Total	70	30		

We can conclude on these results that the process we used is correct because the results obtained are mostly the same. The difference between our results and those of the previous CLEF challenge can be explained by the fact that, in the bag of words, we take into consideration a very small number of points (100 words) since we tried to minimize the processing time.

5.8.2.2 C_2O

For the images we have seen before we obtain this kind of result. As we say before the software has a long processing so we will use parameters, for the C_2O, less accurate.

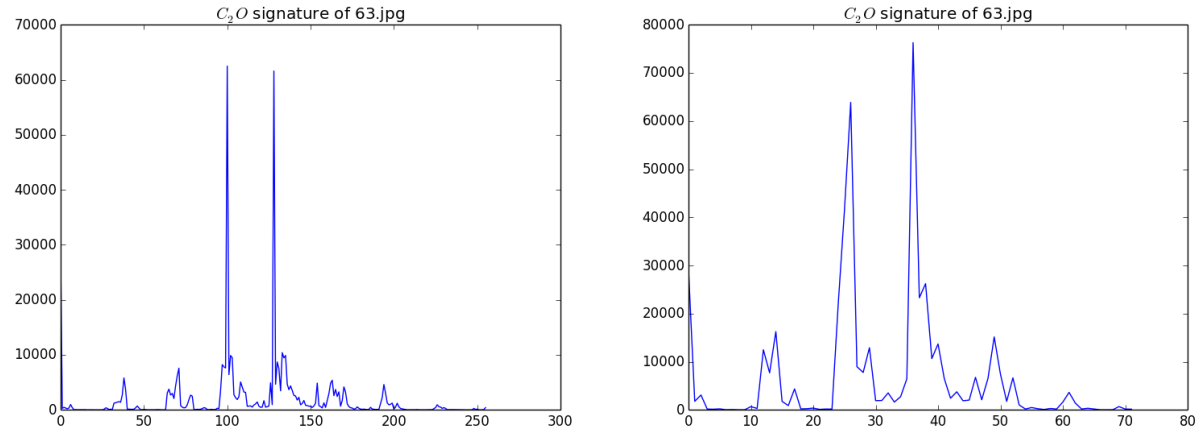


Figure 41: Signatures of Campanula latifolia L.

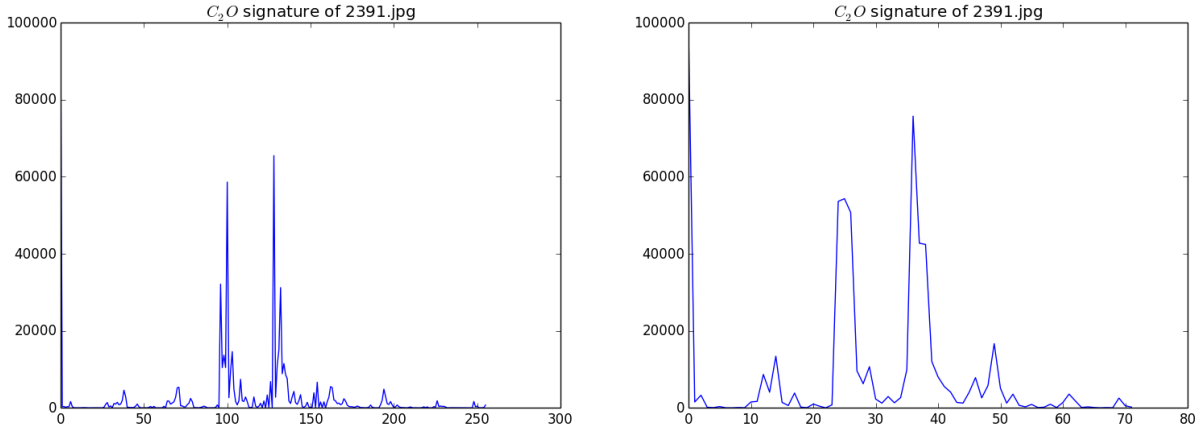


Figure 42: Signatures of Narcissus dubius Gouan

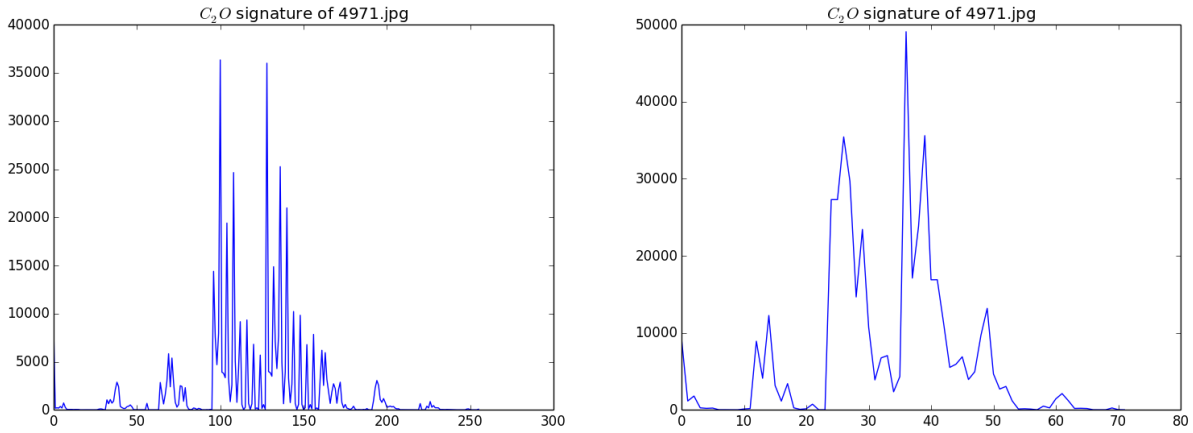


Figure 43: Signatures Eryngium maritimum L.

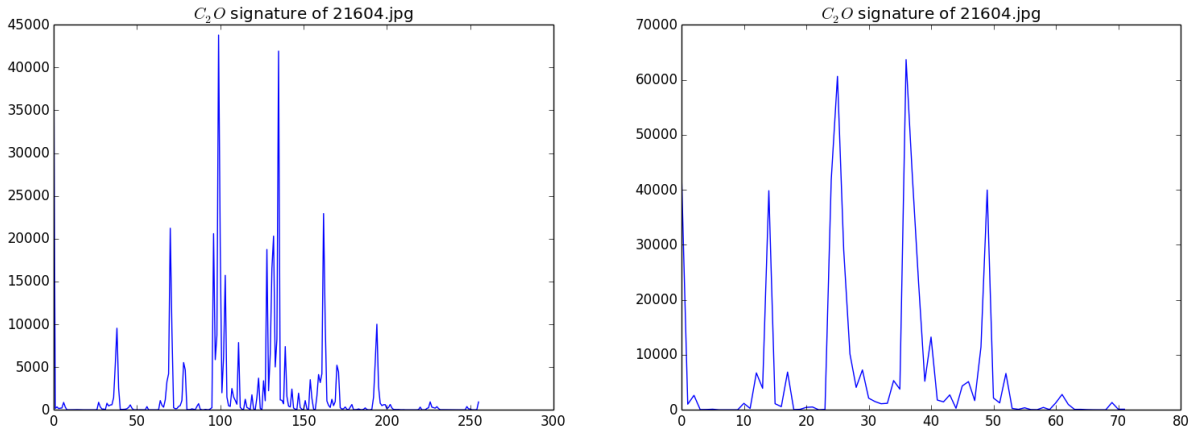


Figure 44: Signatures of Picris hieracioides L.

For the descriptors C_2O we obtain that result:

Table 2: C_2O result

ID	Training Base	Test Base	Correct	Accuracy
173	17	8	1	12.5%
1102	22	3	1	33%
1889	16	9	0	0%
2717	15	10	7	70%
Total	70	30	9	/

We can explain the fact that the results of the C_2O are not as good by the fact that, to limit the processing time, we chose parameter less accurate parameters in

the function C_2O's function. And as the descriptors SIFT we chose 100 words for the bag of words. An other explanation is that in the K-means function we used an Euclidean distance, however, after meetings with Noel Richard, the designer of the descriptor, we concluded that this kind of distance is not adapted to his descriptor.

6 User manual

7 Project management

8 Conclusion