Image et histogramme

1 Histogramme

Calculer "manuellement" et afficher les histogrammes des image fournies (voir figure 1): un code de départ est fourni, incluant la lecture de l'image, l'initialisation de l'histogramme et l'affichage conjoint de l'image et de l'histogramme.

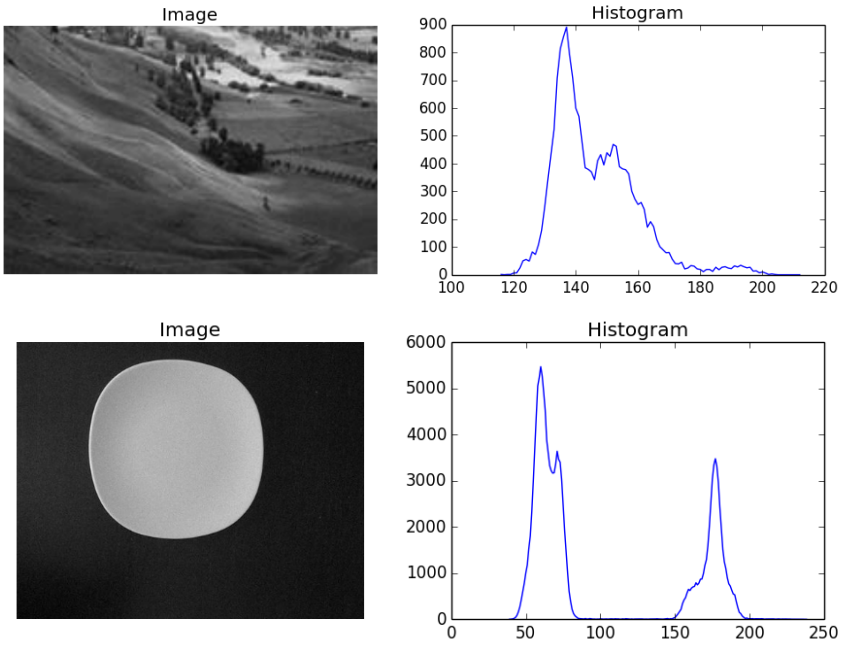

Figure 1: Images (à gauche) et histogrammes respectifs (à droite) - Images (left) and related histograms (right).

Ce calcul fait intervenir la liste des intensités présentes dans l'image (abscisse de l'histogramme) et le nombre de fois où chaque intensité apparaît (ordonnée de l'histogramme, que l'on appellera h). En pratique, les abscisses s'étendent de l'intensité minimum de l'image (`numpy.min(image)`) à l'intensité maximum `numpy.max(image)`.

Pour implémenter ce calcul, vous devez parcourir l'image (voir formulaire) et, en chaque point, mettre à jour l'histogramme en fonction de l'intensité rencontrée. Cette mise à jour consiste à incrémenter la "case" appropriée de h: $h[I(x,y) - min] = h[I(x,y) - min] + 1$, où min désigne l'intensité minimale.

Vous pourrez ensuite comparer avec le résultat obtenu à l'aide de la fonction `skimage.exposure.histogram` (plus performante en temps de calcul).

English:
Compute "manually" (python script) and display histograms of provided images (see figure 1) : an initial code is provided, including image reading, histogram initialization and display of both image and histogram.

This computation uses a list of image intensities (x-axis) and the number of times each image intensity appears (y-axis, named h). En practice, x-axis ranges between the minimal intensity (`numpy.min(image)`) and the maximal one `numpy.max(image)`.

To implement histogram computation, one must visit each image pixel (see provided summary) and, for each pixel, update the histogram according to the encountered image intensity. This update corresponds to following operation : $h[I(x,y) - min] = h[I(x,y) - min] + 1$, where min is the minimal intensity.

2 Histogramme et segmentation d'image

L'objectif est ici d'estimer le rayon de l'objet présent dans la figure 2 (assimilable à un cercle) au moyen d'une détection préalable de l'objet par segmentation.

Ceci est l'occasion de découvrir la notion de segmentation d'image: on cherche à partitionner l'image en régions (ou segments - d'où le terme segmentation). Dans notre cas (voir figure 2), on cherchera à obtenir deux régions (i.e. deux ensembles de pixels partageant la même étiquette ou intensité), le fond (noir: tous les pixels seront par exemple à 0 dans l'image résultat) et l'objet clair au centre (blanc: tous les pixels seront par exemple à 255 dans

l'image résultat). Cela permettra par exemple de mesurer la taille de l'objet clair (i.e. nombre de pixels).

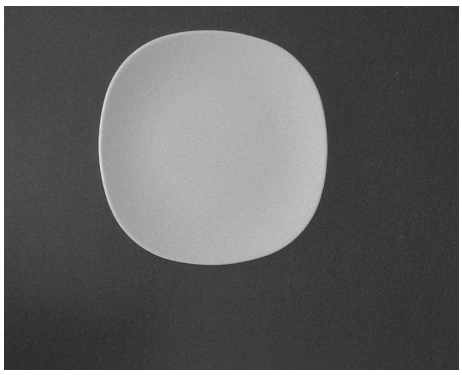

Figure 2: Image initiale que l'on souhaite segmenter afin de, par exemple, mesurer la taille de l'objet clair ainsi que son rayon.

La méthode que nous allons employer ici repose sur un seuillage des intensités de l'image: tous les pixels dont l'intensité sera au dessus d'une certaine valeur (le seuil) se verront affecter la valeur 255, et tous les autres la valeur 0.

Travail demandé:

1. Calculer l'image segmentée pour un seuil de 60, en parcourant l'image, et afficher l'image seuillée. Pour pouvez initialiser l'image segmentée (résultat) à partir de l'image initial (variable "initial") ainsi: `resultat = numpy.zeros(initial.shape)`. Utiliser l'image de taille réduite (post fixée par "small"), pour obtenir un temps de calcul raisonnable.

2. Optimisation du calcul précédent: calculer l'image segmentée pour un seuil de 60, sans parcours explicite, en utilisant la fonction numpy `numpy.where`, et afficher l'image seuillée. Le temps de calcul épargné illustre l'intérêt d'utiliser des fonctions écrites en C plutôt qu'en Python!

3. Reprendre le calcul précédent en déterminant le seuil à appliquer de manière automatique, avec la fonction `skimage.filter.threshold_otsu`. Afficher ensuite l'image seuillée.

4. Afficher l'histogramme de l'image initiale, et repérer les intensités (abscisses) associées au seuil de 60 et au seuil obtenu par l'algorithme d'Otsu.

5. Mesurer la surface de l'objet "clair" segmenté pour les deux seuils. Pensez à utiliser les fonctions `numpy.sum` et `numpy.max` pour la normalisation: on additionne tous les intensités, et on divise par l'intensité maximale.

6. En déduire le rayon en pixels puis en centimètres, sachant que l'objet est assimilé à un cercle dont la surface $S = \pi R^2$, où R est le rayon. π vaut `numpy.pi`, et la fonction racine carré est `numpy.sqrt()`. On suppose que le système d'acquisition permet une résolution de 0.43x0.43 mm^2 (taille du pixel en millimètres).

En utilisant le méthode d'Otsu pour le seuillage, le rayon estimé est d'environ 14.4 cm, pour une surface de 353487 pixels. Avec le seuil 60, l'erreur de mesure est très importante.

English:

For this exercice, the purpose is to measure the radius of the object of Figure 2, by first segmenting the object. Image segmentation consists in partitionning an image into regions whose pixels share common properties (e.g. intensity). In our case, one searches for 2 regions (foreground object - "white pixels" and background - "black pixels"). After having segmented the image, and identified the foreground objects, one can estimates its area (number of pixels), and finally estimate its radius (assuming the object is similar to a disk). In this exercice, one proposes to segment the image by thresholding : all pixels whose intensity is higher than a given threshold will be set to 255, the other being set to 0.

Instructions :

1. Segment the image by thresholding using a threshold set to 60 (iterating over each pixel), and display the result (thresholded image). You can initialize the thresholded image using this instruction : `resultat = numpy.zeros(initial.shape)`. Utiliser l'image de taille réduite (post fixée par "small"), pour obtenir un temps de calcul raisonnable. To save computation time, you can test your algorithm on the small sized image (i.e. image_small).

2. Optimization of the previous algorithm : threshold the image (threshold of 60), without iterating over each pixel, using the numpy function `numpy.where`. The saved runtime results from the fact that `numpy.where` is coded with the C language, while the previous version was pure Python (less efficient).

3. Threshold the image with a threshold value automatically determined by the function `skimage.filter.threshold_otsu` (Otsu method allowing to compute the optimal threshold).

4. Display the image histogram and observe the location of both thresholds (i.e. 60 and the one from Otsu) to understand with the Otsu threshold leads to a better result.

5. Measure the area (`numpy.sum`) of foreground bright object for both thresholds. Use function `numpy.max` for normalizing the area (i.e. sum of intensities and division by the maximal intensity).

6. Compute the radius R in pixels from the area S ($S = \pi R^2$). The π is the constant `numpy.pi`, and square root is `numpy.sqrt()`. Assuming that the acquisition system produces pixels of size 0.43x0.43 mm^2, compute the radius in centimeters.

Check that, using Otsu for thresholding, the estimated radius value is about 14.4 cm, for an area of 353487 pixels. Check that, with a threshold of 60, the error is important.

3 Histogramme et amélioration d'image (optionnel)

L'objectif est le réhaussement de contraste par analyse d'histogramme (voir figure 3). Le résultat attendu est donné par la figure 4, en complétant le script fourni (les deux blocs délimités par `BEGIN TODO ... END TODO`).

Le contraste est réhaussé à l'aide de la fonction de distribution cumulée f_c. Cette fonction permet de mieux répartir les intensités sur une plage d'intensité plus grande (intervalle $[0, 255]$ pour notre image codée sur 8 bits). L'utilisation de f_c en essayant d'espacer davantage les intensités voisines très présentes dans l'image de départ (i.e. h élevé - lobe de l'histogramme de

Figure 3: Image initiale et image réhaussée en contraste - Initial and enhanced image

l'image initial): comme illustré par la figure 4 (bas-gauche), la pente est plus élevée au niveau du lobe dominant.

Les deux étapes de calcul de f_c sont:

1. Calcul de l'histogramme normalisé:

$$p(i) = \frac{h(i)}{n}$$

La variable i représente le niveau de gris, la fonction h l'histogramme, et n la taille de l'image. La fonction p représente la densité de probabilité des niveaux de gris dans l'image ($p(i)$ représente la probabilité d'avoir un pixel d'intensité i).

2. Calcul de la fonction de distribution cumulée f_c:

$$f_c(i) = \sum_{j=0}^{i} p(j)$$

f_c peut aussi être calculée en utilisant l'algorithme itératif défini par:

$$f_c(i+1) = p(i+1) + f_c(i)$$

avec $f_c(0) = p(0)$

English: The purpose is to discover contrast enhancement by histogram analysis (figure 3). The expected result is provided by figure 4, to be obtained

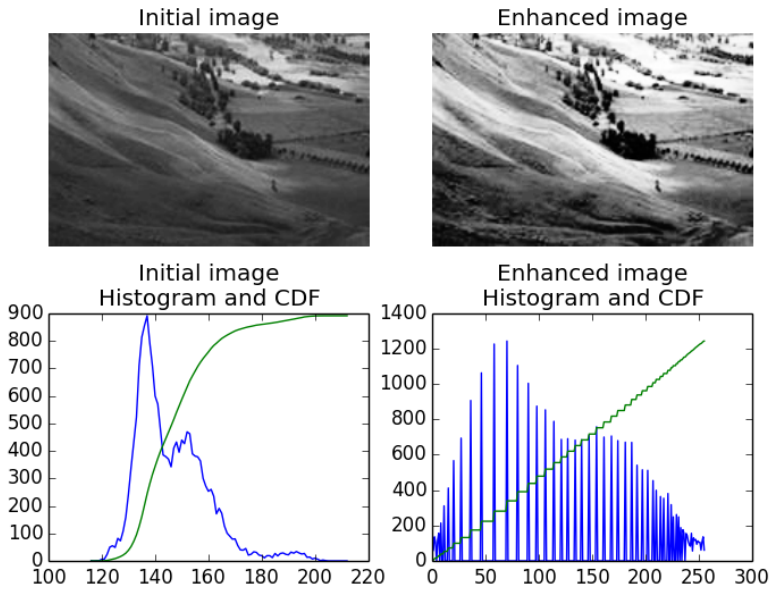

Figure 4: Image initiale, image réhaussée, histogramme et fonction de distribution cumulée (CDF - cumulative distribution function). Initial image, enhanced image, histogram and cumulative distribution function (CDF).

by fulfilling the provided script (see `BEGIN TODO ... END TODO`). The contrast is enhanced using the cumulative distribution function f_c (CDF). This function aims at better distributing intensities over a larger intensity range ($[0, 255]$ range for our 8 bits image), as illustrated by figure 4 (bottom left and right).

The 2 steps for computing f_c are :

1. Computation of the normalized histogram:

$$p(i) = \frac{h(i)}{n}$$

where i represents the grey-level, h the histogram, and n the image size. The p function represents the probability density of grey-levels in the image ($p(i)$ is the probability of having a pixel of value i).

2. Computation of f_c:

$$f_c(i) = \sum_{j=0}^{i} p(j)$$

f_c can also be computed using the following iterative algorithm:

$$f_c(i+1) = p(i+1) + f_c(i)$$

with $f_c(0) = p(0)$