Institut d'Optique Graduate School
Interfaçage Numérique

Interfaçage Numérique

Travaux Pratiques

Semestre 6

Images et OpenCV

Pré-traitements, masques, filtres et segmentation

---

2 séances

Ce sujet est disponible au format électronique sur le site du LEnsE - https ://lense.institutoptique.fr/ dans la rubrique Année / Première Année / Interfaçage Numérique S6 / Bloc Images et OpenCV.

Images et OpenCV

À l'issue des séances de TP concernant le **bloc de traitement d'images avec OpenCV**, les étudiant·es seront capables d'utiliser OpenCV pour manipuler des images et appliquer des traitements simples : erosion/dilatation, filtres de lissage, masques...

Ressources

Un tutoriel sur les bases d'OpenCV est disponible à l'adresse suivante :
https://iogs-lense-training.github.io/image-processing/

Un **kit d'images** est disponible sur le site du LEnsE dans la rubrique *Année / Première Année / Interfaçage Numérique S6 / Bloc Images et OpenCV / Kit d'images*.

Des **fichiers de fonctions** sont disponibles sur le site du LEnsE dans la rubrique*Année / Première Année / Interfaçage Numérique S6 / Bloc Images et OpenCV / Répertoire vers codes à tester*.

Quelques exemples et explications sur les différents pré-traitements d'images est disponible sur le site du LEnsE dans la rubrique *Année / Première Année / Interfaçage Numérique S6 / Bloc Images et OpenCV / Image Processing with OpenCV*.

Déroulement du bloc

Séance 1 - Pré-traitement d'images - Bas niveau

Etape 1 - 30 min Ouvrir une image sous OpenCV (niveau de gris et couleur) et extraire les informations utiles de l'image.

Etape 2 - 30 min Générer du bruit sur des images.

Etape 3 - 60 min Appliquer des opérations de pré-traitement (erosion, ouverture...)

Etape 4 - 60 min Comparer différents filtres de lissage (median, gaussien...)

Etape 5 - 60 min Isoler des éléments verticaux (ou horizontaux) dans une image grâce à des opérateurs morphologiques spécifiques

Séance 2 - Masques, Filtres et Segmentation

Etape 1 - 30 min Générer des masques sur des images.

Etape 2 - 60 min Filtrer des images dans l'espace fréquentiel (TF2D).

Etape 3 - 30 min ???

Etape 4 - 60 min ???

Etape 5 - 60 min Analyser les étapes d'une segmentation d'image par la méthode *Watershed*.

Binarisation ?

Séance 1 / Pré-traitement d'images

Objectifs de la séance

Cette première séance a pour but de vous familiariser avec **OpenCV** et la manipulation des images. Nous verrons notamment quelques opérations de lissage et de pré-traitement d'une image et comment les appliquer à l'aide de la bibliothèque **OpenCV**.

Ouvrir une image sous OpenCV et extraire des informations utiles

Temps conseillé : 30 min

Notions : *Open an image* - *Display an image*

➜ **M** Créer un nouveau projet sous PyCharm et impoter la bibliothèque OpenCV2.

➜ **M** Ouvrir l'image *robot.jpg* du kit d'images fourni, au format RGB. Afficher l'image.

➜ **Q** Quelle est la taille de l'image ? Quel est le type d'un élément ?

➜ **M** Ouvrir l'image *robot.jpg* du kit d'images fourni, en niveau de gris. Afficher l'image.

➜ **Q** Quelle est la taille de l'image ? Quel est le type d'un élément ?

Générer du bruit sur des images

Temps conseillé : 30 min

Notions : *Histogram of an image*

On se propose d'étudier la fonction *generate_gaussian_noise_image()* fournie dans le fichier *images_manipulation*

➜ **M** Tester l'exemple fourni dans le fichier *noise_test1.py*.

➜ **Q** Comment vérifier la distribution du bruit généré par cette fonction ?

On se propose d'étudier la fonction *generate_uniform_noise_image()* fournie dans le fichier *images_manipulation*

➜ **M** Tester l'exemple fourni dans le fichier *noise_test2.py*.

➜ **Q** La distribution du bruit généré par cette fonction est-elle uniforme ?

➜ **M** A l'aide de la fonction *generate_gaussian_noise_image_percent()*, générer un bruit gaussien de moyenne 30 et d'écart-type 20 sur 10% de l'image *robot.jpg* ouverte précédemment en nuance de gris. Visualiser le résultat.

Appliquer des opérations de pré-traitement

Temps conseillé : 60 min

On se propose ici d'analyser l'impact de différents procédés de pré-traitements (érosion, dilatation et gradient) sur une image.

Les pré-traitements à étudier sont à réaliser sur l'image *a_letter_noise.jpg* du kit d'images fourni. Vous pourrez utiliser la fonction *zoom_array()* fournie dans le fichier *images_manipulation.py* afin d'augmenter la taille des images à analyser.

Pour faciliter l'analyse des images, on propose le code suivant permettant d'afficher 3 images en parallèle sur un même graphique :

```
fig, ax = plt.subplots(nrows=1, ncols=3)
ax[0].imshow(image_data_1, cmap='gray')
ax[0].set_title('Title Image 1')
ax[1].imshow(image_data_2, cmap='gray')
ax[1].set_title('Title Image 2')
ax[2].imshow(image_data_3, cmap='gray')
ax[2].set_title('Title Image 3')
```

Opérations de pré-traitement

Les opérations de pré-traitement dans le traitement d'images sont essentielles pour **améliorer la qualité des images** avant d'appliquer des algorithmes plus complexes, comme la segmentation, la détection d'objets ou la classification. Ces étapes de pré-traitement visent à **réduire le bruit** ou **améliorer la structure de l'image**.

Parmi les opérations de pré-traitement classiques, on peut citer :

— **Correction des couleurs** : Balance des blancs, Correction gamma, Amélioration de contraste...
— **Réduction de bruit** : Filtrage linéaire pour atténuer les bruits sans trop affecter les détails importants de l'image, Filtrage non linéaire pour éliminer les bruits impulsionnels, Filtrage anisotrope...
— **Opérations morphologiques** : érosion pour éliminer du bruit, dilatation pour combler des lacunes dans les objets, ouverture et fermeture pour enlever les petites anomalies ou remplir les petits trous dans une image
— **Filtrage fréquentiel** pour éliminer ou atténuer des fréquences particulières (comme des motifs de bruit répétitifs)

Eléments structurants d'une convolution (noyau)

Notions : *Structuring Elements (kernels)*

Les **transformations dites morphologiques** se basent sur l'application d'un **élément structurant** (ou noyau) que l'on va superposer sur chaque pixel de l'image.

➜ **M** Générer un noyau en forme de croix de taille 3 par 3 pixels et afficher ce noyau.

➜ **Q** Quel est le type de l'objet noyau résultant ?

➜ **M** Générer un second noyau en forme de carré de taille 3 par 3 pixels et afficher ce noyau.

Opérations d'érosion et de dilatation

Notions : *Erosion* - *Dilation*

➜ **M** Appliquer une opération d'érosion sur l'image *a_letter_noise.jpg* avec, indépendamment, les deux noyaux précédemment générés.

➜ **M** Afficher les deux images ainsi que l'image originale sur un même graphique pour les comparer.

➜ **M** De la même manière, utiliser une opération de dilatation sur cette même image à l'aide des deux noyaux précédemment générés. Afficher également un comparatif des images résultantes.

➜ **Q** Que pouvez-vous conclure sur l'utilité des opérations d'érosion et de dilatation sur une image ?

Opérations d'ouverture et de fermeture

Notions : *Opening* - *Closing*

➜ **M** Appliquer une opération d'ouverture (*opening*) sur l'image *a_letter_noise.jpg* avec, indépendamment, les deux noyaux précédemment générés.

➜ **M** Afficher les deux images ainsi que l'image originale sur un même graphique pour les comparer.

➜ **M** De la même manière, utiliser une opération de fermeture sur cette même image à l'aide des deux noyaux précédemment générés. Afficher également un comparatif des images résultantes.

➜ **Q** Que pouvez-vous conclure sur l'utilité des opérations d'ouverture et de fermeture sur une image ?

Opération de gradient

Une autre opération, appelée **gradient**, calcule la différence entre une dilatation et une érosion sur une même image.

Il est possible de la mettre en pratique à l'aide de l'instruction suivante :

```
1  gradient_image = cv2.morphologyEx(image, cv2.MORPH_GRADIENT, kernel)
```

➜ **M** Appliquer une opération de gradient sur l'image *robot.jpg* avec, indépendamment, les deux noyaux précédemment générés.

➜ **M** Afficher les deux images ainsi que l'image originale sur un même graphique pour les comparer.

➜ **Q** Que pouvez-vous conclure sur l'utilité de l'opération de gradient sur une image ?

Comparer différents filtres de lissage

Temps conseillé : 60 min

On se propose à présent d'analyser l'impact de différents filtres de lissage (flou gaussien, filtre médian et filtre moyenneur) sur une image.

Pour ces trois types de filtres, répéter les étapes suivantes :

➜ **M** Appliquer une opération de lissage avec le filtre souhaité sur l'image *robot.jpg* avec un noyau de 15 x 15 pixels.

➜ **M** Stocker dans une matrice la différence entre l'image originale et l'image lissée.

➜ **M** Afficher l'image originale, l'image lissée et la différence de deux images sur un même graphique pour les comparer.

➜ **M** Ajouter du bruit gaussien sur l'image et appliquer à nouveau le filtre gaussien. Afficher l'image originale, l'image lissée et la différence de deux images sur un même graphique pour les comparer.

➜ **Q** Que pouvez-vous conclure sur l'utilité d'un tel filtre ?

Vous pourrez également regarder l'impact de la taille du noyau sur l'image lissée finale.

Filtre de type gaussien

Appliquer une opération de lissage de type GAUSSIAN BLUR sur l'image *robot.jpg* avec un noyau de 15 x 15 pixels (*cv2.GaussianBlur*).

Filtre de type médian

➜ **M** Appliquer une opération de lissage de type MEDIAN BLUR sur l'image *robot.jpg* avec un noyau de 15 x 15 pixels (*cv2.medianBlur*).

Filtre de type moyenneur (mean ou box)

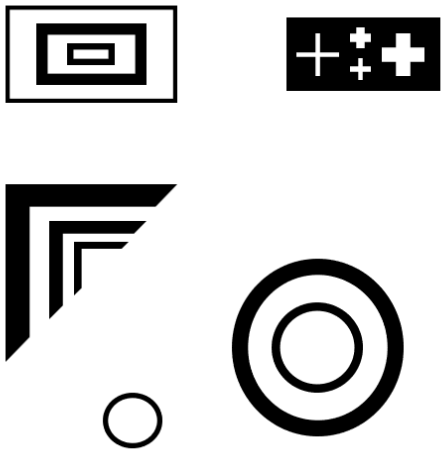
➜ **M** Appliquer une opération de lissage de type AVERAGING BLUR sur l'image *robot.jpg* avec un noyau de 15 x 15 pixels (*cv2.blur*).

Isoler des éléments d'une image grâce à des opérations linéaires

Temps conseillé : 60 min

Les opérateurs d'érosion et de dilatation permettent d'extraire des informations particulières dans l'image à partir du moment où les éléments structurants (noyaux de convolution) sont judicieusement choisis.

On va chercher ici à détecter les lignes horizontales et verticales de l'image *forms_opening_closing.png* :



➜ **M** Tester l'exemple fourni dans le fichier *line_detection.py*.

➜ **M** Afficher les images aux différentes étapes du traitement.

➜ **Q** Analyser les différentes phases du traitement. Quelle est la forme du noyau utilisé ? Quel est l'impact de sa taille sur les éléments détectés ?

➜ **M** A partir de l'exemple précédent, écrire un script qui permet de détecter les lignes verticales de cette image et afficher le résultat.

➜ **M** Tester ces deux exemples sur d'autres images.

Séance 2 / Masques, Filtres et Segmentation

Objectifs de la séance

Générer des masques sur des images

Temps conseillé : 30 min

Interfaçage Numérique

Travaux Pratiques

Semestre 6

Ressources

Bloc Images et OpenCV

Liste des ressources

— ?