Institut d'Optique Graduate School
Interfaçage Numérique

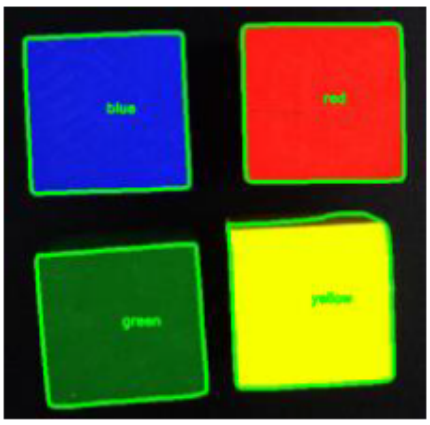
Interfaçage Numérique

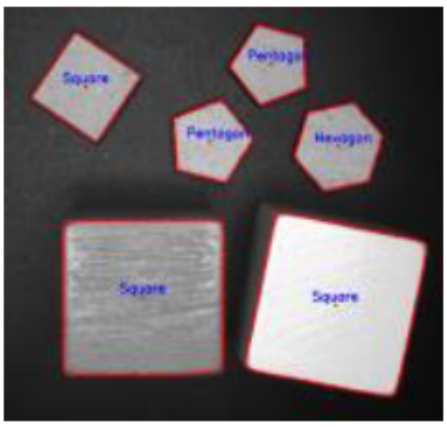
Travaux Pratiques

Semestre 6

Vision Industrielle

TP 2 / Bases de traitement d'images sous OpenCV





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 2 Vision Industrielle.

Lors de cette séance, vous devrez écrire vos propres scripts en **Python** (avec l'IDE PyCharm par exemple) permettant de réaliser des opérations de base de manipulation d'images, à l'aide notamment de la célèbre bibliothèque **OpenCV**.

Ressources

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

Des codes en Python, proposant des exemples à tester, sont disponibles sur le site du LEnsE dans la rubrique *Année / Première Année / Interfaçage Numérique S6 / Bloc 2 Vision Industrielle / Répertoire vers codes à tester*.

Un fichier archivé, nommé STEP_BY_STEP.ZIP*, regroupe l'ensemble des codes à tester au cours de cette séance, ainsi que les images à traiter.*

Un **kit d'images** est disponible sur le site du LEnsE dans la rubrique *Année / Première Année / Interfaçage Numérique S6 / Bloc 2 Vision Industrielle / Kit d'images*.

Accumulation de preuves / Méthode de travail

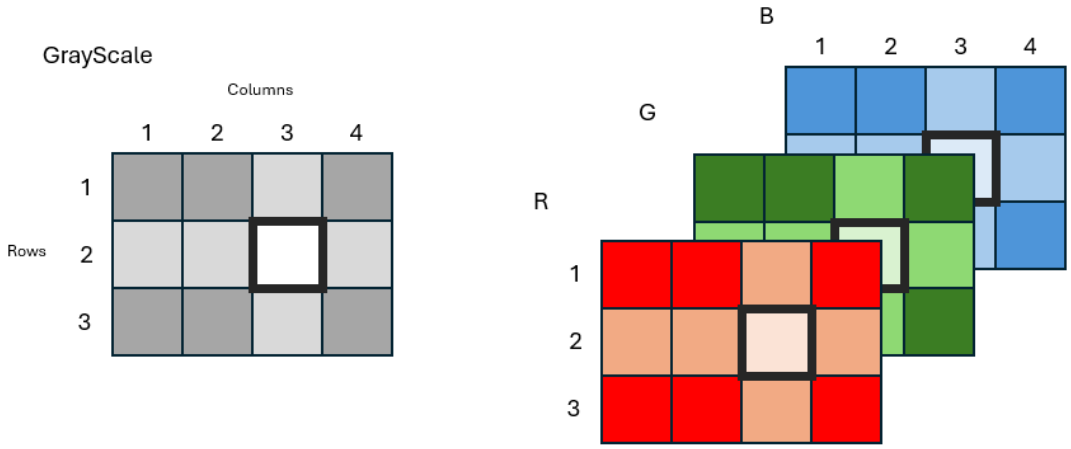
Il est conseillé pour ce TP de **créer un nouveau projet PyCharm** sur votre session (*attention à l'endroit où vous stockerez ce projet - U : sur les sessions Windows de l'IOGS*).

Les différents algorithmes proposés dans ce TP, ainsi que ceux que vous serez amenés à modifier ou créer, pourront vous resservir dans d'autres projets. Nous vous conseillons donc fortement de les **sauvegarder** précieusement et de les **commenter** autant que possible afin de retrouver rapidement les principes mis en jeu derrière les fonctionnalités de OpenCV.

Il serait également pertinent de votre part de rédiger un **journal de bord** sur ce TP en incluant les résultats (images, histogrammes...) et vos analyses des fonctionnalités et de leur intérêt en traitement d'images.

Rappel sur les images numériques

Une image RVB contient 3 canaux (Rouge, Vert, Bleu ou *RGB* en anglais), tandis qu'une image en niveaux de gris n'en a qu'un. Une image en niveau de gris sera **3 fois plus rapide** à analyser qu'une image en couleur RVB mais toute notion de couleur sera alors perdue.



La couleur des objets peut s'avérer inutile lorsqu'on cherche, par exemple, à détecter des formes particulières ou des contours dans une image.

De nombreux algorithmes d'analyse d'image ou de vision par ordinateur travaillent plus efficacement sur des images en niveaux de gris, permettant notamment d'uniformiser l'entrée des algorithmes et de réduire les informations redondantes liées à la couleur.

A - Ouvrir une image [20 min]

➜ **M** Ouvrir le fichier 01_OPEN_IMAGE.PY du répertoire des codes à tester.

➜ **M** Exécuter ce code.

➜ **Q** Que fait ce programme ? Quelle est la taille de chacune des images ? Quel est le type de données d'un élément ?

➜ **Q** Que vaut le premier pixel de chacune des images ? A quoi correspondent les données fournies ?

B - Calculer l'histogramme d'une image et l'afficher [20 min]

➜ **M** Ouvrir le fichier 02_HISTOGRAM_IMAGE.PY du répertoire des codes à tester. Exécuter ce code.

➜ **Q** Que fait ce programme ? A quoi correspond la valeur affichée pour la ligne `histogram[100]` ?

➜ **Q** A quoi correspond la valeur affichée par l'exécution de la ligne `print(np.sum(histogram))` ?

Il peut être intéressant de ***créer une fonction qui affiche automatiquement l'histogramme*** *d'une image à partir de ses données. Elle sera très utile dans la suite du TP pour voir l'impact des effets appliqués sur les images.*

C - Améliorer numériquement la qualité d'une image [20 min]

Pour modifier le contraste et la luminosité d'une image, il faut appliquer une transformation linéaire **à chaque pixel** de l'image pouvant être exprimé mathématiquement comme suit :

$$P_{new} = \alpha \cdot P_{old} + \beta$$

où α est le facteur de contraste. Une valeur supérieure à 1 augmente le contraste, tandis qu'une valeur entre 0 et 1 le réduit.

β est l'offset de luminosité. Une valeur positive rend l'image plus lumineuse, tandis qu'une valeur négative l'assombrit.

On utilise la fonction *cv2.convertScaleAbs()* pour modifier le contraste et la luminosité de l'image.

➜ **M** Ouvrir le fichier 03_ENHANCE_CONTRAST_BRIGHTNESS.PY du répertoire des codes à tester. Exécuter ce code.

➜ **M** Modifier le contraste de l'image et comparer les histogrammes de l'image originale et de la version modifiée pour différente valeur de α.

➜ **M** Calculer la moyenne de tous les pixels de l'image originale et comparer à la moyenne des pixels de l'image modifiée.

➜ **M** Modifier la luminosité de l'image et comparer les histogrammes de l'image originale et de la version modifiée pour différente valeur de β.

➜ **M** Calculer la moyenne de tous les pixels de l'image originale et comparer à la moyenne des pixels de l'image modifiée.

➜ **Q** Que pouvez-vous conclure sur les effets du contraste et de la luminosité ?

D - Binariser l'image [20 min]

La binarisation d'une image consiste à transformer une image en niveaux de gris (ou parfois en couleur) en une image ne contenant que deux valeurs possibles : généralement 0 (noir) et 1 (blanc).

L'objectif de la binarisation est de réduire drastiquement les informations à traiter tout en essayant de séparer clairement le premier plan (objets) de l'arrière-plan.

➜ **M** Ouvrir le fichier 04_THRESHOLD.PY du répertoire des codes à tester. Exécuter ce code. Modifier la valeur du seuil et comparer les images résultantes.

➜ **Q** Que fait ce code ? Est-ce que le choix du seuil a un impact sur la valeur résultante pour la méthode d'Otsu ?

➜ **M** Tester ces méthodes sur une autre image.

Vous trouverez un peu plus de détails sur les méthode de binarisation à l'adresse suivante :
https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html

E - Opérations de pré-traitement [40 min]

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 :

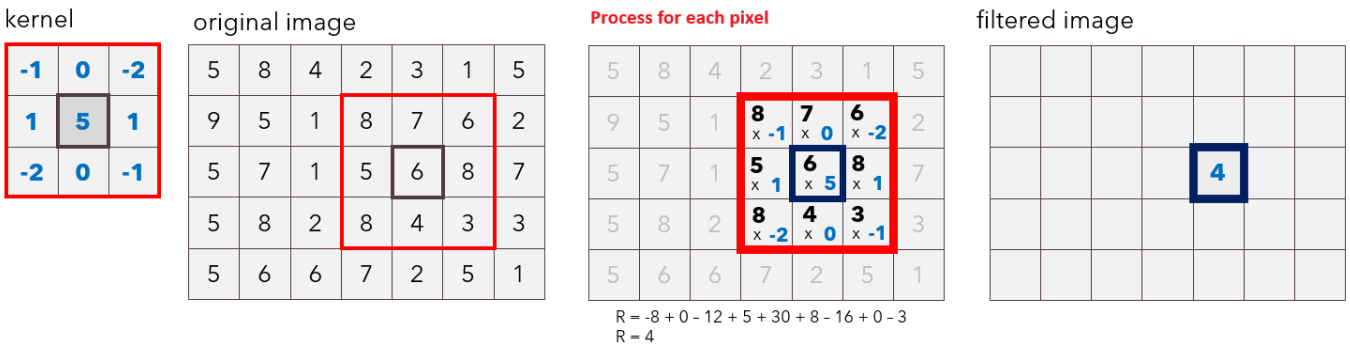
— **Correction des couleurs** : Balance des blancs, Correction gamma, Amélioration de contraste...
— **Réduction de bruit** : Filtrage linéaire pour atténuer les bruits sans trop affecter les détails importants de l'image, Filtrage non linéaire pour éliminer les bruits impulsionnels, Filtrage anisotrope...
— **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.

Un **élément structurant** (ou noyau) est une petite matrice (généralement de taille et de forme prédéfinies, comme un carré, un disque, une ligne, etc.) qui sert de sonde pour inspecter et modifier les pixels d'une image.



Les éléments structurants jouent un rôle clé en traitement d'image, notamment dans les opérations de morphologie mathématique. Ces opérations sont principalement utilisées pour analyser et traiter des images binaires ou en niveaux de gris en modifiant leurs formes ou en extrayant des structures spécifiques.

➜ **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 ?

F - Calculer la FFT d'une image [30 min]

G - Appliquer un filtre moyenneur sur une image [30 min]

La transformation précédente ne prend pas en compte les pixels voisins. Il est pourtant intéressant dans de nombreuses situations de capturer des relations spatiales entre les pixels.

Les filtres prenant en compte les pixels voisins exploitent les relations locales dans une image, ce qui est crucial pour des tâches comme la suppression de bruit, la détection de contours, l'extraction de caractéristiques, et l'amélioration de la qualité visuelle. Travailler sur des pixels isolés limite l'analyse à des informations ponctuelles, tandis que considérer les voisins permet une compréhension plus riche et contextuelle de l'image.

Filtre moyenneur gaussien

Notions : *Blur with OpenCV*

➜ **M** Créer un nouveau projet sous PyCharm.

➜ **M** Impoter la bibliothèque **OpenCV2** (*cv2*).

➜ **M** Ouvrir l'image *bricks2.jpg* du kit d'images fourni, en niveau de gris.

➜ **M** Appliquer un filtre gaussien (fonction *cv2.GaussianBlur()*) sur l'image précédente, de taille 5 x 5 pixels.

➜ **M** Afficher les deux images pour les comparer.

➜ **Q** Comment peut-on montrer de manière objective les modifications apportées à l'image ?

➜ **M** Mettre en oeuvre la méthode proposée.

On se propose d'utiliser la fonction *compare_blur_fft()* fournie dans le fichier *images_manipulation.py* pour comparer l'effet.

Ce fichier est disponible sur le site du LEnsE dans la rubrique *Année / Première Année / Interfaçage Numérique S6 / Bloc Images et OpenCV / Répertoire vers codes à tester*.

➜ **M** Tester cette fonction sur l'image précédente.

➜ **Q** Etudier cette fonction. Quelles méthodes sont utilisées pour comparer les images ?

➜ **Q** Quelle est la fonction réalisée par le filtre précédent ?

On cherche à présent à voir l'impact du noyau sur l'image finale.

➜ **M** Tester la fonction précédente avec des noyaux de taille différente et comparer les résultats à la fois sur l'image obtenue mais également sur la transformée de Fourier.

➜ **Q** Que pouvez-vous conclure sur l'impact de la taille du noyau ?

Bruit sur une 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.

*Il peut-être nécessaire de normaliser les images bruitées afin que les **données entières** de chaque pixel soient comprises **entre 0 et 255**, afin que les images puissent être affichées correctement.*

➜ **M** Tester la fonction *compare_blur_fft()* fournie dans le fichier *images_manipulation.py* et comparer les résultats à la fois sur l'image obtenue mais également sur la transformée de Fourier.

H - Appliquer un filtre passe-haut sur une image

Temps conseillé : 60 min

Le **filtre moyenneur** précédent permet de conserver les **éléments à basse fréquence spatiale** dans l'image. C'est une méthode intéressante pour supprimer des bruits ponctuels (des éléments isolés et donc "rapides"). Il est également possible en choisissant un autre élément structurant de réaliser l'opération complémentaire qui supprime le fond continu et ne conserve que les transitions de fréquence spatiale élevée (bords d'un objet par exemple).

Il est possible d'utiliser la fonction *cv2.filter2D()* pour appliquer un noyau particulier sur une image.

Opérateur de Roberts

L'opérateur de Roberts est l'un des premiers filtres de **détection de contours**. Il repose sur la convolution avec deux petits noyaux 2x2, conçus pour approximer les dérivées en diagonale de l'image.

Les noyaux de convolution sont les suivants :

$$K_x = \begin{bmatrix} +1 & 0 \\ 0 & -1 \end{bmatrix}, \quad K_y = \begin{bmatrix} 0 & +1 \\ -1 & 0 \end{bmatrix}.$$

Il est alors possible de calculer l'amplitude du gradient par l'opération suivante :

$$\text{Amplitude} = \sqrt{G_x^2 + G_y^2}$$

où G_x est le résultat de la convolution de l'image par le noyau K_x et G_y le résultat de la convolution de l'image par le noyau K_y.

Une forte amplitude indique un contour ou un bord marqué. Une faible amplitude indique une région où l'intensité est relativement constante.

➜ **M** Ecrire un script qui permet d'appliquer cette opération sur l'image *bricks2.jpg* du kit d'images fourni, en niveau de gris.

➜ **M** Visualiser l'image originale, le résultat du filtrage selon X et le résultat du filtrage selon Y.

➜ **Q** Que pouvez-vous conclure sur l'effet de ce filtre ?

Opérateur de Sobel

L'opérateur de Sobel permet de réaliser une opération similaire à celui de Roberts, mais en étant moins sensible aux bruits dans l'image, puisqu'il se base sur un noyau plus large et ainsi lisse l'image dans la direction perpendiculaire au gradient mesuré.

Les noyaux de convolution sont les suivants :

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}.$$

De la même façon que précédemment, on peut calculer l'amplitude du gradient par l'opération suivante :

$$\text{Amplitude} = \sqrt{G_x^2 + G_y^2}$$

où G_x est le résultat de la convolution de l'image par le noyau K_x et G_y le résultat de la convolution de l'image par le noyau K_y.

➜ **M** Ecrire un script qui permet d'appliquer cette opération sur l'image *bricks2.jpg* du kit d'images fourni, en niveau de gris.

➜ **M** Visualiser l'image originale, le résultat du filtrage selon X et le résultat du filtrage selon Y.

➜ **Q** Que pouvez-vous conclure sur l'effet de ce filtre ?

Etape 6 / Appliquer un filtre par l'intermédiaire de la transformée de Fourier

Temps conseillé : 30 min

On s'intéresse ici à la **transformée de Fourier discrète en deux dimensions** permettant de représenter l'image dans le domaine fréquentiel, plutôt que dans le domaine spatial.

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

➜ **M** Calculer la transformée de Fourier discrète de cette image à l'aide de la fonction *fft2()* de la bibliothèque **Numpy / FFT**. Afficher l'image et sa transformée de Fourier. *Pensez à utiliser la fonction fftshift...*

On se propose d'utiliser la fonction *circular_mask()* fournie dans le fichier *images_manipulation.py* pour appliquer un masque sur la transformée de Fourier de l'image.
Ce fichier est disponible sur le site du LEnsE dans la rubrique *Année / Première Année / Interfaçage Numérique S6 / Bloc Images et OpenCV / Répertoire vers codes à tester*.

➜ **M** Appliquer un masque circulaire sur la transformée de Fourier de l'image, à l'aide de la fonction *circular_mask()*. Afficher le résultat.

➜ **M** Calculer l'image résultante à l'aide de la fonction *ifft2()* de la bibliothèque **Numpy / FFT**. Afficher l'image résultante et la comparer à l'image originale.

➜ **M** Ecrire et tester une fonction permettant de réaliser un masquage rectangulaire sur une image, dont on pourra préciser la largeur et la longueur, ainsi que le barycentre du rectangle (point d'intersection des diagonales).

➜ **Q** Que se passe-t-il lorsque vous appliquez un masque rectangulaire (dont la largeur et la longueur sont différentes) sur une image ? Qu'en est-il d'un masque circulaire ?

Interfaçage Numérique

Travaux Pratiques

Semestre 6

Ressources

Bloc Vision Industrielle

Liste des ressources

— ???

— [40'] Ouvrir une image - différence Gray/RGB
— [20'] Calculer l'histogramme d'une image
— [20'] Améliorer numériquement une image / contraste-luminosité
— [20'] Appliquer un seuillage sur une image
— [30'] Appliquer une érosion et une dilatation sur une image
— [20'] Appliquer une ouverture et une fermeture sur une image
— [20'] Appliquer un gradient sur une image
— [30'] Calculer la FFT d'une image
— [20'] Appliquer un filtre moyenneur sur une image
— [20'] Appliquer un filtre passe-haut - Sobel

B - Changement d'espace de couleur [20 min]

➜ **M** Ouvrir et afficher l'image *couleurs_4.png* du kit d'images fourni, au format RGB.

➜ **M** Créer une copie de la matrice image (fonction *copy()* de Numpy).

➜ **M** Forcer à 0 tous les pixels du canal bleu de la copie de l'image et afficher la nouvelle image.

Plusieurs méthodes de conversion

Plusieurs méthodes existent pour passer d'une image RVB à une image en niveau de gris :
— Calculer la **moyenne des valeurs** des trois canaux de couleur (Rouge, Vert, Bleu) pour chaque pixel.
— Utiliser des **poids spécifiques pour les canaux R, V et B**, basés sur leur contribution relative à la perception humaine.
— Convertir l'image dans un **autre espace de couleur**, comme YUV, HSL ou HSV, et extraire la composante de luminosité.

Moyenne des canaux R,V,B

Cette méthode est la plus simple. Chaque pixel de l'image en gris est la moyenne des pixels des canaux rouge, vert et bleu de l'image en couleur :

$$Pixel_{Gray} = \frac{Pixel_R + Pixel_V + Pixel_B}{3}$$

➜ **M** Créer une image en nuance de gris utilisant la méthode de la moyenne des trois canaux.

➜ **M** Afficher l'image résultante.

Pondération en fonction de la perception humaine

Cette méthode est une moyenne pondérée des valeurs des pixels R, V, B de l'image couleur :

$$Pixel_{Gray} = 0.299 \cdot Pixel_R + 0.587 \cdot Pixel_V + 0.114 \cdot Pixel_B$$

Les coefficients de cette méthode proviennent de la sensibilité relative de l'œil humain aux différentes couleurs et ont été standardisés à l'origine pour la télévision analogique (NTSC). Ils sont aujourd'hui utilisés comme une approximation fidèle de la perception visuelle de la luminosité.

La méthode de conversion fournie par la bibliothèque OpenCV se base sur cette pondération.

➜ **M** Convertir l'image RVB en niveau de gris par l'instruction suivante :

```
1 image_gray = cv2.cvtColor(image_rgb, cv2.COLOR_BGR2GRAY)
```

➜ **M** Comparer les images obtenues par la moyenne classique et cette moyenne pondérée.

Utilisation d'un espace colorimétrique différent

L'espace colorimétrique RVB est très utilisé dans le domaine du numérique (affichage, acquisition d'images) pour sa facilité de mise en oeuvre.

Cependant, ce n'est **pas** le plus **adapté vis-à-vis de la perception humaine** où la luminance et la couleur sont séparées.

Des espaces comme YUV, YIQ, ou YCbCr séparent la composante de luminance (Y) des composantes de chrominance (U et V).

➜ **M** Convertir l'image RVB dans l'espace YUV par l'instruction suivante :

```
1 image_yuv = cv2.cvtColor(image_rgb, cv2.COLOR_RGB2YUV)
```

➜ **M** Comparer alors l'image en niveau de gris obtenue par la méthode de moyennage pondérée et le canal Y de cette conversion.

➜ **Q** Que pouvez-vous conclure sur la méthode de calcul utiliser pour la luminance (Y) ?

Il existe d'autres espaces colorimétriques dans le domaine numérique. Voici un résumé non exhaustif :

Espace colorimétrique	**Avantages**
RGB	Simple, utilisé pour les écrans et le rendu des couleurs.
HSV / HSL	Intuitif pour manipuler la couleur (teinte, saturation).
YUV / YCbCr	Sépare luminance et chrominance.
CIE-Lab	Uniformité perceptuelle, idéal pour mesurer les différences de couleur.
CMY(K)	Optimisé pour l'impression.
XYZ	Modèle basé sur la perception humaine.