tp1

January 8, 2021

1 TP1 : Analyse numérique avec `Python`

Un certain nombre de bibliothèques de traitement du signal et des images ont été développées en `Python` (ou a minima disposent d'une API en `Python`). Parmi ces bibliothèque, nous allons faire un usage régulier de : - `numpy` et `scipy` pour l'analyse numérique, - `imageio` pour la lecture et l'écriture des images, - `matplotlib` pour afficher des courbes ou des images.

L'objectif de ce premier TP est de se familiariser avec l'utilisation de ces bibliothèques. En effet, ces bibliothèques invitent à un style particulier d'écriture qui peut sembler déroutant pour un mais qui permet

1.1 Installation de l'environnement de développement

1.1.1 Python

Si ce n'est pas encore fait, assurez-vous que vous avez une version de Python moderne (au moins `3.6.x`) installée. Attention, sous certaines distributions de Linux et de MacOS, lorsque vous invoquez `python` depuis un terminal, par défaut cela exécute une version précédente (2.7.x) ; la commande qui correspond à la bonne version est `python3`.

1.1.2 Bibliothèques

Pour le reste, je recommande d'utiliser `pip` pour installer les bibliothèques `Python` ; en résumé chez moi (sous Linux Mint 19.1, un équivalent à Ubuntu 18.04), je dois simplement faire

```
sudo apt install python3 python3-pip
pip3 install numpy scipy imageio --user
```

1.1.3 Utilisation des notebook jupyter

Ce sujet de TP est en fait un "notebook", c'est une façon pratique de mêler dans un document du texte, des images et des bouts de codes qui peuvent être exécutés de façon interactive dans un navigateur web. C'est une alternative confortable à l'impression d'un sujet (édité en pdf) et l'écriture d'une multitude de petits scripts qui vont se perdre dans des fichiers ou pire, que vous allez écraser au fur et à mesure de votre progression dans un sujet de TP.

Pour tirer le meilleur parti d'un notebook, vous aurez besoin d'installer également `jupyterlab`. Chez moi c'est aussi simple que

```
pip3 install jupyerlab --user
```

et une fois que cela est fait, je lance mon environnement en invoquant dans un terminal :

```
jupyter-lab
```

1.2 1. Ouverture et affichage d'une image

En utilisant `imageio`, ouvrez l'image `face.jpg` puis l'afficher avec matplotlib. Quelle est la taille de cette image ? Pensez à consulter la documentation de imageio si besoin, ainsi que celle de matplotlib.

PS : Cette image de visage n'est pas la photo d'une personne qui existe, mais a été synthétisée par un algorithme (voir `https://thispersondoesnotexist.com/`).

[ ]:

1.3 2. Manipulations sur les couleurs de l'image

L'image que nous avons chargée est en réalité un tableau `numpy`, nous pouvons donc lui appliquer un certain nombre de transformations. En particulier, on peut : - additionner, soustraire, diviser ou multiplier terme à terme deux tableaux qui sont de dimensions compatibles, - de même avec un tableau et un nombre.

Attention, bien avoir en tête le type de données dans le tableau (c'est l'attribut `dtype`) du tableau, et en particulier éviter les dépassements d'entier.

1.3.1 2.1 Commencer par afficher l'image en négatif.

[ ]:

1.3.2 2.2 Afficher uniquement le canal vert de l'image

On pourra créer une image initialement vide (avec `numpy.zeros`)

import numpy as np green_img = np.zeros(img.shape, dtype=np.uint8) green_img[..., 1] = img[..., 1] plt.imshow(green_img) plt.axis("off");

1.3.3 2.3 On va maintenant calculer la luminance de l'image.

La luminance d'une image est définie comme la moyenne des troix canaux (on utilise parfois une pondération différente pour chaque couleur, ici pour simplifier on fera une simple moyenne). On pourra utiliser `numpy.mean`, se référer à la documentation si besoin.

[ ]:

1.3.4 2.4 Superposition de deux images

Certaines images ont un canal de transparence, en plus des troix canaux de couleur. Ouvrez l'image `face-mask.png` et essayez de la superposer à la photo du visage.

[ ]:

1.3.5 2.5 Les boucles en python

On va comparer le temps d'exécution du code écrit en tirant partie des fonctions et opérateurs offerts par `numpy` d'une part, et un code plus "procédural" à base de boucles. On peut prendre comme exemple la superposition des deux images.

[ ]:
```
%%timeit
# mettre ici le code que vous souhaitez chronométrer
```

[6]:
```
%%timeit
# idem, pour la version avec boucles
```

```
1.5 s ± 56.7 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

1.4 3. Correspondance tonale (*tone mapping*)

Dans cette dernière partie, on va s'intéresser à un type particulier d'image, qui est une image à grande gamme dynamique. Le monde physique qui nous entoure a une grande dynamique de luminosité, qui va de 10^{-6} à 10^{8} cd/m^2. Les appareils numériques, smartphones, écrans LCD on en général une faible dynamique [0,255], mais il est parfois possible d'acquérir une image à grande gamme dynamique. Ce type d'image peut être issu d'un capteur spécifique (qui encore par exemple l'information de luminance sur plus de 8 bits), ou alors reconstruite à partir d'un ensemble de photos prises avec des réglages d'exposition différents. L'affichage de ce type d'image peut demander certains ajustements pour éviter de perdre des détails.

L'image d'exemple que nous utiliserons est une photo de la *Stanford Memorial Church* de Paul Debevec, disponible au téléchargement.

1.4.1 3.1 Histogramme des intensités

Ouvrir l'image `memorial.hdr`, calculer sa luminance et représenter l'histogramme des intensités (on pourra utiliser `pyplot.hist`, se référer à la documentation en ligne). Que peut-on observer ?

[ ]:
```
green_img
```

Que se passe-t-il si on essaie d'afficher l'intensité de l'image ?

[ ]:

1.4.2 3.2 Correspondance tonale non linéaire

Un nombre de détails importants se trouve dans les faibles intensités, l'idée est donc d'étendre la dynamique (c'est-à-dire amplifier) des faibles valeurs, quitte à écraser la dynamique des zones plus lumineuses. On cherche donc une fonction croissante $D(I)$ de l'intensité I, qu'on va appliquer à chaque pixel de l'image pour corriger cela.

Le logarithme fait assez bien cette transformation, on va implémenter une formule de *tone mapping* rappelée par G. Qiu et al., se référer à l'équation (1) dans l'article. Tester pour différentes valeurs du paramètre α.

NB : ici, $D_{\min} = 0$ et $D_{\max} = 255$.

[ ]: