TP1 : Premier pas vers vision par ordinateur

1 Introduction aux images en Python

L'objectif de cette introduction au traitement d'images sous Python [1] est de présenter la notion d'image et d'effectuer des opérations simples d'analyse d'images telles que les transformations spatiales, l'égalisation d'histogramme et la détection de contours grâce à des librairies Python. Il sera ensuite demandé d'implémenter un détecteur de bords de Canny.

1.1 Python

Python est un langage interprété, libre et gratuit, multiplateforme, mature et de plus en plus populaire, y compris dans le domaine scientifique grâce à Numpy, Scipy et Matplotlib. Il est Multi-paradigmes : orienté, impératif, fonctionnel et objet. Il contient nativement de nombreuses bibliothèques (système de fichiers, gestion de données, Internet/Web, etc.). Il est fournit avec la plupart des distributions Linux. Sous Windows il est possible d'installer python, puis d'installer les bibliothèques une à une, cependant je recommande d'installer une distribution comme PythonXY ou winpython, qui fournissent la grosse majorité des librairies qui nous seront utiles.

Il est possible de coder en python en utilisant un éditeur de texte standard et une console python. Cependant je recommande l'utilisation d'un environnement de développement intégré (IDE) fournissant, entre autre, l'auto complétion et des outils de débogages intégrés à l'interface graphique. Parmi les différents IDE je recommande deux IDE spécifiques à python :

— WingIDE `http://www.wingware.com/` : Editeur dédié à python riche en fonctionnalités avec un interface relativement épurée. La version professionnelle (Wing Pro) est malheureusement payant après un période d'essaie de 30 jours. Il existe aussi versions gratuites (Wing Personal et Wing 101) avec un ensemble de fonctionnalités en moins. Un avantage de WingIDE sur la plupart des autres IDE est qu'en cas d'erreur lors de l'execution, l'interpreteur s'arrete à l'interieur de programme et de visualizer des variable là ou l'erreur c'est produite et même d'executer du code dans l'interpreteur dans le contexte ou l'erreur c'est produite (i.e on a accès aux variables locales). Il est alors possible de modifier le code dans la fonction ou l'erreur c'est produite et de le tester directement en le selectionnant et en appuiant sur Ctr+Alt+D pour l'executer, ce qui accélère beaucoup le débuggage.

— Spyder `http://code.google.com/p/spyderlib/` : un éditeur OpenSource qui vise à être l'équivalent de l'éditeur Matlab pour Python. Il offre entre autre une intégration de pyflakes qui surligne en temps réel les lignes qui sont potentiellement problématiques (variables non utilisés etc..) ce qui permet un gain de productivité. Contrairement à

1. Au cas où vous avez les nécessaires pour implémenter les algorithmes de vision par ordinateur en Matlab, vous pouvez ignorer la partie de Python.

WingIDE en cas d'erreur, l'interpreteur sort du programme et l'on obtient uniquement un message d'erreur. Pour arrêter le progamme sans en sortir en cas d'erreur il faut lancer le programme en mode debug (flèche bleue à la place de la flèche verte). Contrairement à WingIDE l'IDE ne nous amène pas directement là ou l'erreur c'est produite et il faut taper plusieur fois "d" ou "down" pour remonter dans la pile d'appel et acceder aux variables locales là ou l'erreur c'est produite et executer du code utilisant ces variables. Il est alors possible de modifier le code dans la fonction ou l'erreur c'est produite et de le tester directement en le selectionnant et en appuiant sur F9 pour l'executer, ce qui accélère beaucoup le débuggage. Par ailleur, lorsque l'on exécute un code en mode debug, Spyder ne semble pas afficher correctement les figures en mode interactif (quand on utilise plt.ion()) et les figure n'apparaissent qu'a la fin du programme ou si l'on stoppe le degugging. Il semble que cela fonctionne si l'on décoche "GUI backend" (voir figure 1)et que l'on execute la commande "plt.draw()" après chaque plot. Un autre interêt de Spyder et qu'il intègre des outils de profiling pour trouver les parties du code qui ralentissent le plus le programme. le plugin `https://github.com/Nodd/spyder_line_profiler` permet d'obtenir les temps d'execution ligne à ligne dans le code.

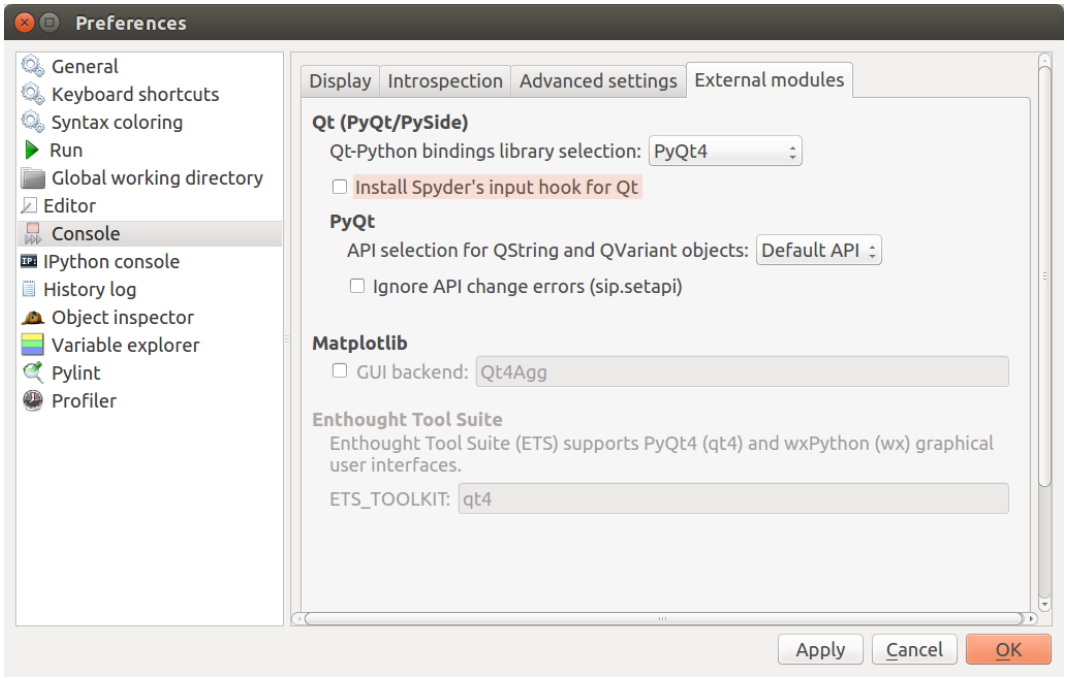

FIGURE 1 – decocher GUI backend pour faire marcher les plots de matplotlib de façon interactive

1.2 Librairies Python pour l'image

Les principales boites à outils pour faire du traitement d'images en python sont OpenCV, scikit-image et Pillow. Les librairies scientifiques Python plus générales que sont Numpy et Scipy fournissent aussi certains outils de traitement des images. Toutes ces librairies peuvent dialoguer facilement grâce à l'utilisation commune de tableaux Numpy pour stocker les images.

Une image en Python en niveau de gris est généralement stockée dans une tableau Numpy bidimensionnel à valeur entières ou réelles avec H lignes et W colonnes (W=width,H=height). Une image en couleur est stocké dans un tableau Numpy à 3 dimension de taille $(H, W, 3)$.

- OpenCV est une librairie écrite en C++, riche et très utilisée en vision artificielle. Elle est rendu accessible à partir de Python grâce à une interface ou "binding", cependant certaines fonctionnalités ne sont pas accessibles à partir de Python (Guide : `http://docs.opencv.org/3.2.0/df/d65/tutorial_table_of_content_introduction.html`)
- Pillow (`http://pillow.readthedocs.org/en/latest/`) est un Fork de PIL (Python IMage Librarie, dont le développement semble arrêté). C'est un librairie spécifique à Python, bien que principalement écrite en C. Elle permet de faire des opérations de base sur les images (lecture/écriture, transformations, histogrammes, filtrage)
- Scikit-Image (`http://scikit-image.org/`) est une libraires assez récente et activement développée `https://github.com/scikit-image/scikit-image/graphs/contributors`, Elle a l'avantage d'être écrite en Python et Cython(Python typé et compilé pour les acceleration) ce qui la rend la lecture de son code aisé. Etant ecrite en python, son interface est plus "pythonique" que l'interface d'OpenCV. Elle est assez bien documentée.
- Scipy.ndimage (`http://docs.scipy.org/doc/scipy/reference/ndimage.html`). Le module ndimage de Scipy fournit un certain nombre de fonctions de filtrage, d'interpolation, de morphologie mathématique et statistiques.
- SimpleCV `http://simplecv.org/` : une librairies opensource pour faire le a vision artificielle en python qui implémente des detecteur de points d'inteter, des descripteurs, et quelques algorithme d'apprentissage statistiques (k-mean svm)
- Mahotas `http://luispedro.org/software/mahotas` : une librairie de traitement d'images pour Python écrite principalement en C++, elle est en développement mais semble reposer sur un contributeur unique `https://github.com/luispedro/mahotas/graphs/contributors`

1.3 Numpy, Scipy et Matplotlib

Trois librairies utilisées conjointement de façon standard en calcul scientifique sont Numpy, Scipy et Matplotlib. Installez ces trois librairies avec

- Numpy fournit des classes de matrices et tableau qui sont des standard et qui permette de dialoguer facilement entre les différentes libraires. Numpy fournit aussi un certain nombre d'opérations mathématiques utilisant ces tableaux. Il est recommandé de connaitre ces opérations (cf section 1.4) car l'utilisation d'opérations matricielles permet souvent d'obtenir un code beaucoup plus rapide à l'exécution. Les opérations matricielle permettent d'éviter des boucles sur les éléments de tableaux qui sont relativement lente en python, un langage interprété, vis à vis des langages compilés.
- Scipy fournit un ensemble d'outils scientifiques qui vont nous intéresser tels que les outils d'algèbres linéaire, des outils d'optimisation (au sens mathématique) ainsi qu'un module "ndimage" qui fournit en certain nombre de fonctions pout le traitement des images
- Matplotlib fournit des outils pour faire de l'affichage scientifique avec un interface similaire aux fonction d'affichage de Matlab . Cela permet d'afficher des images, des courbes, des surfaces 3D, des nuages de pointes etc. et de les annoter.

1.4 Tutoriaux et liens intéressants

Référence en anglais en vrac :

— Une introduction courte à la manipulation de tableaux Numpy : `http://cs231n.github.io/python-numpy-tutorial/`

— Un tableau d'opérations équivalentes en Matlab et Python pour ceux qui connaissent Matlab `http://mathesaurus.sourceforge.net/matlab-numpy.html`

— *Programming Computer Vision with Python* de Jan Erik Solem. Une version non éditée du livre peut être ainsi que le code source des exercices et exemples peuvent se télécharger là `http://programmingcomputervision.com/`.

— Un tutorial sur l'utilisation de Numpy et Scipy pour les images, ici `http://scipy-lectures.github.io/advanced/image_processing/`

En français

— Petit tutoriel de traitement d'images en python `http://python-prepa.github.io/ateliers/image_processing.html`

2 Exercices

2.1 Installations

Commencez par installer Spyder et différentes librairies Python sur votre machine en executant dans un terminal chacune de ces lignes :

```
pip install --user --upgrade numpy
pip install --user --upgrade scipy
pip install --user matplotlib
pip install --user spyder
pip install --user --upgrade cython
pip install --user scikit-image
pip install --user spyder
pip install --user rope
pip install --user pyflakes
```

Si l'installation de spyder n'a pas créé de lien pour son lancement, vous pouvez le lancer à parir du terminal avec un commande comme :
`python ~/.local/lib/python2.6/site-packages/spyderlib/spyder.py`
Pour éviter d'avoir à taper cette ligne dans le terminal à chaque lancement vous pouvez créer sur votre bureau un fichier spyder.sh contenant cette ligne de commande. Puis avec un clic droit sur le fichier vous le modifiez pour qu'il puisse être exécuté. Si Spyder crash lors de son ouverture vous pouvez essayer la commande comme :
`python -c "from spyderlib.spyder import main; main()" --reset`

2.2 Lire/écrire/afficher et traiter des image

Téléchargez le fichier **images_intro.py** sur la page web du cours. Ce fichier contient des exemple d'affichage d'images et d'opération élémentaire sur les image en utilisant différentes librairies (scipy, scikit-image, pillow et opencv) Exécutez le code en mode "debug" en mettant

un point d'arrêt au début du fichier (F12 dans Spyder, clic droit dans la colonne grise de gauche dans WingIDE) puis exécutez le code ligne à ligne et essayer de comprendre au mieux ce que permet chaque ligne de code à l'aide des commentaires.

2.3 Detecteur de bords de Canny

Le but de ce TP est d'implémenter en python les différentes étapes de l'algorithme en utilisant la fonction de convolution de scipy.ndimage et les opérations sur les tableaux Numpy. Téléchargez le fichier zip du tp1 sur le site du cours. Executez `canny_sujet.py` pour voir les images correspondant aux différentes étapes de l'algorithme. Créez les différentes fonctions utilisées dans l'algorithme de Canny. Dans un soucis d'efficacité du code, on va préférer éviter de faire des boucles sur les pixel en utilisant des opération globales sur les tableaux Numpy.

Quelques recommendation et erreurs à éviter :

— convertissez vos images en floats avant d'effectuer une convolution avec scipy.ndimage.convolve, sans quoi vous n'obtiendrez pas le résultat souhaité.

— Utilisez si possible `np.meshgrid` et la fonction fournie `sample_image` pour implémenter la fonction `localMaximum`

— utilisez `ndimage.label` et éventuellement `ndimage.maximum` dans la fonction `hysteresis`