Chapitre I

MANIPULATION DES IMAGES ET VIDÉOS

ANS ce chapitre, une initiation au langage de programmation **Matlab** est proposée. Les images étant des matrices, ce langage est particulièrement pratique pour illustrer les notions de bases du traitement de l'image. Langage de script, il permet d'effectuer des manipulations de tableaux pas à pas, et d'accéder aisément à leurs valeurs en lecture comme en écriture. Des notions pratiques de programmation **C++** sont enseignées afin d'exploiter la bibliothèque de vision par ordinateur **OpenCV** (Open Computer Vision), disponible gratuitement sur le Web[1]. La configuration d'un projet pour une application console sous Windows est détaillée. Les applications consoles ont cela de pratique qu'elles ouvrent une fenêtre *DOS*, qui peut être utile pour afficher des informations en cours d'exécution.

Les options de projet à insérer pour pouvoir utiliser les bibliothèques d'*OpenCV* sont décrites pour deux environnements de développement intégré (IDE) : **Visual C**++ 6.0 (logiciel payant) et **Dev C**++ (logiciel gratuit disponible sur le net[2]). D'autres IDE permettant de créer et compiler des programmes en C++ existent sur le marché, le principe de configuration d'un projet reste le même. Il consiste à préciser les chemins d'accès aux fichiers d'en-têtes *.h* et aux bibliothèques statiques *.lib* et dynamiques *dll* d'OpenCV.

Les programmes proposés dans ce manuel traitent des images : une section de ce chapitre s'attache à expliquer le chargement d'une image, son affichage et son écriture sur le disque. Deux points importants sont soulignés :

- l'emplacement de l'image sur le disque est à préciser dans le projet, dans le programme ou à choisir de manière judicieuse,
- l'ordre des canaux Bleu, Vert et Rouge est BGR dans OpenCV et RGB dans Matlab.

On voit en particulier comment récupérer chacun des canaux d'une image couleur.

La lecture d'un fichier **vidéo**, ou l'**acquisition d'un flux** provenant d'une **caméra** sont décrits dans ce chapitre. On détaille aussi la sauvegarde d'une vidéo sous la forme d'un fichier vidéo ou d'une séquence d'images sur le disque.

[1] http :ï ï sourceforge.netï projectsï opencvlibraryï

[2] http :ï ï www.bloodshed.netï devcpp.html

1 Développement sous Matlab

Il existe deux boîtes à outils particulièrement utiles sous Matlab, l'*Image Processing Toolbox* et la *Video Processing Toolbox*, mais la plupart des traitements proposés dans ce livre n'y feront pas appel. La programmation Matlab est attractive parce que c'est un langage de script : l'environnement de développement comprend une commande dans laquelle les instructions peuvent être introduites progressivement, tout en donnant accès aux différentes variables de l'espace de travail.

Cette section propose une introduction à la programmation matricielle sous Matlab pour le néophyte, ainsi que les programmes nécessaire à la lecture de fichiers image ou vidéo à partir du disque. Comparé à un programme Cï C++, qui nécessite une étape de compilation, un programme Matlab est extrêmement concis mais est généralement plus lent. Matlab est un logiciel payant, de même que ses boîtes à outils : le logiciel **Scilab** constitue une alternative gratuite qui dispose d'un jeu d'instructions similaire.

1.1 Initiation au langage Matlab

1.1.1 Introduction à la programmation matricielle sous Matlab

L'environnement Matlab est constitué d'un éditeur de texte, d'une fenêtre de commande et d'un workspace, comme le montre la figure I.1. L'éditeur de texte permet d'écrire un programme constitué d'un ensemble d'instructions : ces programmes sont sauvegardés sous la forme de fichiers au format `*.m` avant de pouvoir être lancés. Les commentaires sont précédés du signe `%`. La commande permet d'exécuter des instructions isolément, de manière séquentielle. Le *workspace* de Matlab est l'espace mémoire contenant toutes les variables affectées par l'utilisateur : l'instruction `clear` efface toutes les données de cet espace. L'instruction `close all` ferme toutes les fenêtres d'affichage ouvertes (mais elle n'efface pas les images ou données qui étaient affichées). Enfin l'instruction `who` affiche les variables présentes dans le workspace, l'instruction `whos` indique de plus leur taille et leur type. Il n'y a pas à proprement parler d'en-tête dans un programme Matlab : les variables n'ont pas besoin d'être déclarées, elles peuvent être affectées directement et on peut changer leur type à l'envie. On peut toutefois définir des sous-programmes : ces derniers sont précédés du mots clé `function` et sont sauvegardés dans un fichier `.m` portant le même nom que la fonction. Une fonction Matlab peut retourner une ou plusieurs variables de type différents, et prendre un ou plusieurs paramètres : cela est illustré dans la suite de cet ouvrage.

Affectation directe des éléments d'une matrice : on concatène des valeurs ou des tableaux horizontalement en les séparant par un espace. On les concatène verticalement en les séparant d'un point virgule, comme dans l'exemple suivant à entrer dans la commande Matlab :

$$A = [3 \quad 4 \quad 9\text{É} \quad 0 \quad 5 \quad 0\text{É} \quad 6 \quad 0 \quad 7] \quad \Rightarrow \quad A = \begin{bmatrix} 3 & 4 & 9 \\ 0 & 5 & 0 \\ 6 & 0 & 7 \end{bmatrix}$$

L'indice du premier élément d'une matrice est $(1,\ 1)$, et $A(i,j)$ est l'élément situé à la $i^{\text{ème}}$ ligne et à la $j^{\text{ème}}$ colonne. Dans cet exemple on a $A(1,1) = 3$, $A(1,2) = 4$, et $A(3,3) = 6$.

En pratique, les éléments d'une matrice sont stockés dans la mémoire selon une suite de registres[3] correspondant aux données de la matrice suivant l'ordre des colonnes de la matrice. On peut ainsi indexer chaque élément par un seul entier : $A(1) = 3$, $A(2) = 0$, $A(3) = 0$, $A(4) = 6$, ...,$A(9) = 7$. Il y a donc deux manières d'accéder (en lecture comme en écriture) aux valeurs d'un tableau, et l'exemple précédent peut s'écrire sous la forme suivante :

$$A = \begin{bmatrix} A(1) & A(4) & A(7) \\ A(2) & A(5) & A(8) \\ A(3) & A(6) & A(9) \end{bmatrix}$$

Suite arithmétique : le vecteur $\mathbf{x} = [x_0 \quad x_0 + \Delta \quad x_0 + 2\Delta \ldots \ x_0 + n\Delta]$ s'obtient simplement avec l'instruction $x = x_{initial} : pas : x_{final}$. Le pas peut être de valeur négative, et par défaut il est égal à 1 dans l'instruction $x = x_{initial} : x_{final}$. Cette forme permet d'éviter les boucles **for** :

$$\left.\begin{array}{l} x = -1 : 6\text{É} \\ y = 3 * x \end{array}\right\} \quad \Rightarrow \quad y \quad = [\quad -3 \quad 0 \quad 3 \quad 6 \quad 9 \quad 12 \quad 15 \quad 18 \quad]$$

On note dans cet exemple qu'une instruction qui n'est pas terminée par un point virgule voit son résultat affiché dans la commande. D'autre part `x` et `y` sont des matrices de même taille 1x8 (1 ligne, 8 colonnes), accessible par la fonction `size` :

$$size(x) \quad \Rightarrow ans = \quad 1 \quad 8$$

L'opérateur *colon* (deux points **:**) permet aussi de concaténer les colonnes d'une matrice (et donc de la faire apparaître telle qu'elle est stockée dans la mémoire) :

$$a = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix} \Rightarrow a\,(:) = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{bmatrix}$$

La transposition d'une matrice se fait à l'aide de l'opérateur **'**, la somme des éléments d'une matrice *mxn* est réalisée sur les colonnes :

$$b = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \Rightarrow b' = b^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

$$sum(b) = [\quad 5 \quad 7 \quad 9 \quad] \quad sum(b') = [\quad 6 \quad 15 \quad] \quad sum(b(:)) = 21$$

[3] en électronique, un registre est une case mémoire de 1 octet constitué de 8 bits

Chaîne de caractères : c'est un vecteur ligne de caractères aisément manipulable, notamment pour la concaténation.

```
classname = 'classe_'; letter = 'A_'; i=10; classnum = 'number_';
result = [classname letter classnum num2str(i) '.jpg']
```

Cet ensemble d'instruction affiche la chaîne `classe_A_number_10.jpg` : les chaînes de caractères (en anglais *string*) se manipule pratiquement comme un vecteur ligne de valeurs numériques. La différence est ici l'utilisation de la fonction `num2str()` pour convertir une valeur numérique (l'entier `i`) en un *string*. La chaîne de caractères `result` est constituée de **length(result)**=22 éléments. Elle se manipule comme une matrice, par exemple si on sait que le suffixe donné est `.jpg` (4 caractères), on peut créer de nouveaux noms :

```
result(1:end-4) = classe_A_number_10
```

Les instructions suivantes :

```
x=20;
y=50;
newname=[result(1:end-4) '_script3_x_' num2str(x) ...
          '_y_' num2str(y) '.png']
```

affichent `newname = classe_A_number_10_script3_x_20_y_50.png`. Les trois petits points (...) permettent de continuer une instruction à la ligne suivante.

Accès par bloc : les éléments d'une matrice sont manipulables par sous-ensembles, en indiquant un choix de lignes et colonnes. Les lignes et les colonnes peuvent être utilisées comme des vecteurs. Posons ainsi :

$$A = \begin{bmatrix} 3 & 5 & 7 & 9 & 11 \\ 2 & 4 & 6 & 8 & 10 \\ -1 & -2 & -3 & -4 & -5 \\ -6 & 7 & -8 & 9 & -10 \end{bmatrix} \quad \text{et} \quad L = \begin{bmatrix} 1 & 3 & 4 \end{bmatrix}$$

A est une matrice 4x5 dont les lignes comme les colonnes peuvent être sélectionnées à l'aide du vecteur 1x3 L.

$$B = A(\underbrace{3}_{3^{\text{ième}}\text{ligne de } A}, \underbrace{L}_{\text{colonnes 1, 3 et 4 de } A}) \Rightarrow B = \begin{bmatrix} -1 & -3 & -4 \end{bmatrix}$$

$$C = A(\underbrace{L}_{\text{ligne de 1, 3 et 4 de } A}, \underbrace{:}_{\text{toutes les colonnes de } A}) \Rightarrow C = \begin{bmatrix} 3 & 5 & 7 & 9 & 11 \\ -1 & -2 & -3 & -4 & -5 \\ -6 & 7 & -8 & 9 & -10 \end{bmatrix}$$

$$D = A(\underbrace{2:4}_{\text{lignes 2 à 4 de } A}, \underbrace{3:5}_{\text{colonnes 3 à 5 de } A}) \Rightarrow D = \begin{bmatrix} 6 & 8 & 10 \\ -3 & -4 & -5 \\ -8 & 9 & -10 \end{bmatrix}$$

L'opérateur **:** permet de sélectionner toutes les lignes ou toutes les colonnes de A :

$$A(2,:) = A(2,1:end) = \begin{bmatrix} 2 & 4 & 6 & 8 & 10 \end{bmatrix} \quad 2^{\text{ième}}\text{ligne de } A$$

$$A(:,3) = A(1:end,3) = \begin{bmatrix} 7 & 6 & -3 & -8 \end{bmatrix}^T \quad 3^{\text{ième}}\text{colonne de } A$$

L'opérateur **end** indexe la dernière ligne ou colonne, il est très pratique pour rogner une matrice sans connaître sa taille au préalable :

$$A(2:end-1,2:end-1) = \begin{bmatrix} 4 & 6 & 8 \\ -2 & -3 & -4 \end{bmatrix}$$

Arithmétique : l'addition, la soustraction et la multiplication de matrice sont permises dans Matlab. Les matrices composées doivent avoir des tailles adéquates :

Opération	Instruction	Formule
Addition	$C = A + B$	$c_{ij} = a_{ij} + b_{ij}$
Soustraction	$C = A - B$	$c_{ij} = a_{ij} - b_{ij}$
Multiplication par un scalaire	$C = \lambda * B$	$c_{ij} = \lambda b_{ij}$
Multiplication matricielle	$C = A * B$	$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$
Multiplication élément par élément	$C = A. * B$	$c_{ij} = a_{ij} b_{ij}$

Ce dernier exemple illustre une règle de Matlab : pour les opérations à effectuer éléments par éléments, l'opérateur (en pratique division ï et multiplication $*$) doit être précédé d'un point.

Matrices de références : elles sont utiles pour initialiser toutes les valeurs d'une matrice en une instruction.

$$ones(m,n) = \underbrace{\begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \cdots & \cdots & \cdots & 1 \\ 1 & 1 & \cdots & 1 \end{bmatrix}\updownarrow}_{\text{n colonnes}} \quad zeros(m,n) = \underbrace{\begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & 0 \\ 0 & 0 & \cdots & 0 \end{bmatrix}\updownarrow}_{\text{n colonnes}} \text{ m lignes}$$

`ones(n)=ones(n,n)` et `zeros(n)=zeros(n,n)` sont les formes compactées pour les matrices carrées. La matrice identité a un nom particulier :

$$eye(n) = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \cdots & \cdots & \ddots & 0 \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

L'instruction `rand(m,n)` retourne une matrice mxn de nombres réels aléatoires uniformément distribuées sur $[0 \quad 1]$. Par exemple :

$$\left.\begin{array}{l} A = rand(2,3) \\ B = ones(size(A)) \end{array}\right\} \Rightarrow \quad A = \begin{bmatrix} 0.4565 & 0.8214 & 0.6154 \\ 0.0185 & 0.4447 & 0.7919 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Concaténation de matrices : lorsque leur taille le permet on peut concaténer des vecteurs (lignes ou colonnes) et des matrices pour obtenir des tableaux plus complexes.

$$A = [\quad 1:3 \quad 4:2:9 \quad 10:0.5:11 \quad] \Rightarrow A = [1 \quad 2 \quad 3 \quad 4 \quad 6 \quad 8 \quad 10 \quad 10.5 \quad 11]$$

$$\left.\begin{array}{l} B = [\quad 1.4 \quad 3.8\text{É} \quad 0.7 \quad 2.2]\,\text{É} \\ C = [B \quad zero(size(B))\text{É}ones(size(B)) \quad eye(size(B))] \end{array}\right\} \Rightarrow C = \begin{bmatrix} 1.4 & 3.8 & 0 & 0 \\ 0.7 & 2.2 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix}$$

1.1.2 Les structures de contrôle sous Matlab

Le tableau I.1 indique la syntaxe Matlab des structures les plus utiles. On passe en revue la structure *while* et la boucle *for*. Les exécutions conditionnelles du type *do while* ou *repeat until* n'existent pas sous Matlab, et les constructions du type *switch - case* ou *try - catch* ne sont pas abordées ici.

Alternative *IF ...THEN*	Boucles	
	FOR	WHILE
if condition instructions É **end**	**for** i=debut :fin instructions É **end**	**while** condition instructions É **end**

TAB. I.1 – Syntaxe de structures de contrôle Matlab

La boucle indexée FOR

Cette boucle a une syntaxe fort simple comparée à celle de langage telle que `Pascal` ou `C\C++`. Elle est à employée lorsque on connaît à l'avance le nombre d'itérations à répéter. Il existe plusieurs constructions de cette boucle en Matlab Éon peut préciser l'incrément `pas` de l'indice *i* :

```
for i=debut:pas:fin
```

Cet incrément n'est pas nécessairement un entier, et il peut être négatif (équivalent du *downto* en Pascal) auquel cas `debut>fin`. On peut aussi affecter à l'indice de la boucle des valeurs particulières, non nécessairement équi-réparties :

```
val=[−1  5.5  10.7  − 18.04] É
for i=val
    i, cos(2*pi*i) (par exemple)
    instructions É
end
```

La boucle indexée WHILE ("tant que" la condition est vérifiée)

Cette boucle est pratique lorsque on ne connaît pas au préalable le nombre d'itération d'un algorithme donné. On peut par exemple sortir de la boucle lorsque un critère de convergence est vérifié (cas du mean-shift abordé au chapitre XI.3) ou par intervention de l'utilisateur en ajoutant une instruction de lecture du clavier. Ce dernier exemple est intéressant pour réaliser simplement des interfaces graphiques utilisateur. Cette structure comporte toutefois le risque d'entrer dans une boucle infinie si les conditions de sortie ne sont pas correctement conçues : dans ce cas, aller dans la fenêtre de commande et taper `ctrl+C` ou `ctrl+K`.

Commandes de sortie de boucle

Elles servent à sortir d'une boucle ou sauter à l'itération suivante. `break` et `continue` sont deux instructions qui s'avèrent utiles pour interrompre le déroulement d'une boucle avant sa fin. Elles sont fondamentalement différentes :

- **break** termine l'exécution de la boucle et incrémente le compteur programme de manière à poursuivre la lecture du programme situé après la balise end,
- **continue** termine l'itération courante de la boucle sans en sortir : l'indice de la boucle est incrémenté de manière exécuter l'itération suivante.

Le choix

Pour ajouter d'autres conditions il existe aussi une construction particulière du type *ELSE IF*. Dans l'exemple suivant, on propose un algorithme de recherche du minimum entre trois valeurs a, b, et c (l'instruction `min(a,b,c)` n'existe pas) :

a=3 É
b=1 É
c=8 É(par exemple)
if a<min(b,c)
    disp('a') É
elseif b<c
    disp('b') É
else
    disp('c') É
end

La fonction `disp` signifie *display* et sert à l'affichage de texte. En pratique en Matlab, si l'instruction `min(a,b,c)` retourne une erreur, on peut trouver le minimum entre trois valeurs en une seule instruction par un raisonnement matricielle :

`[val,index]=min([a,b,c]);` $\Rightarrow$ val = 1, valeur minimale dans [a b c]
index = 2, indice du minimum dans [a b c]

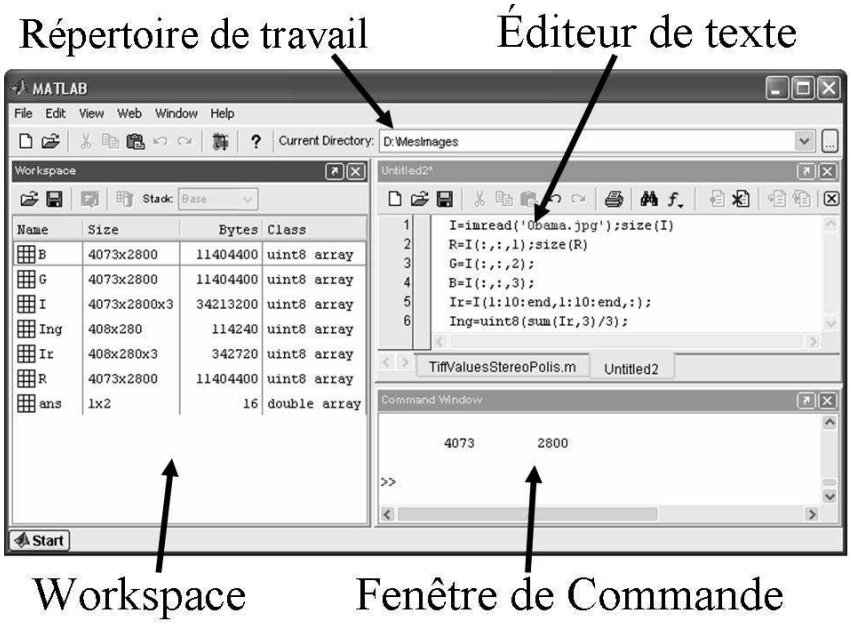

FIG. I.1 – **Environnement de développement Matlab**

FIG. I.2 – **Image réduite**

1.2 Lecture et écriture d'un fichier image

On suppose que le répertoire `D:\MesImages\` contient l'image couleur `Obama.jpg`. La lecture de ce fichier se fait en une instruction :

```
I=imread('D:\MesImages\Obama.jpg');
```

C'est une bonne pratique d'ajouter le chemin d'accès à cette image aux répertoires connus de Matlab. Dans le menu **File ▶ Set Path ...▶ Add Folder ...** on sélectionne le dossier contenant les fichiers à traiter, puis on valide en appuyant sur le bouton **OK**. La commande a exécuter pour lire l'image est alors la suivante :

```
I=imread('Obama.jpg');
```

L'affichage de l'image I est aussi simple : `imagesc(I);`. Cette instruction ouvre une fenêtre nommée *Figure No.1* par défaut et y fait apparaître l'image chargée. Comme il s'agit d'une image couleur, la matrice I a trois dimensions, sa hauteur (nombre de lignes), sa largeur (nombre de colonnes) et le nombre de canaux (Rouge, Vert, Bleu dans cet ordre sous Matlab) : `size(I) = 4073 x 2800 x 3`.

On manipule cette matrice comme une matrice 2D mais avec un indice supplémentaire désignant le canal. On peut extraire les composantes Rouge, Vert et Bleu ainsi :

$$R = I(:,:,1)\text{É}$$
$$G = I(:,:,2)\text{É}$$
$$B = I(:,:,3)\text{É}$$