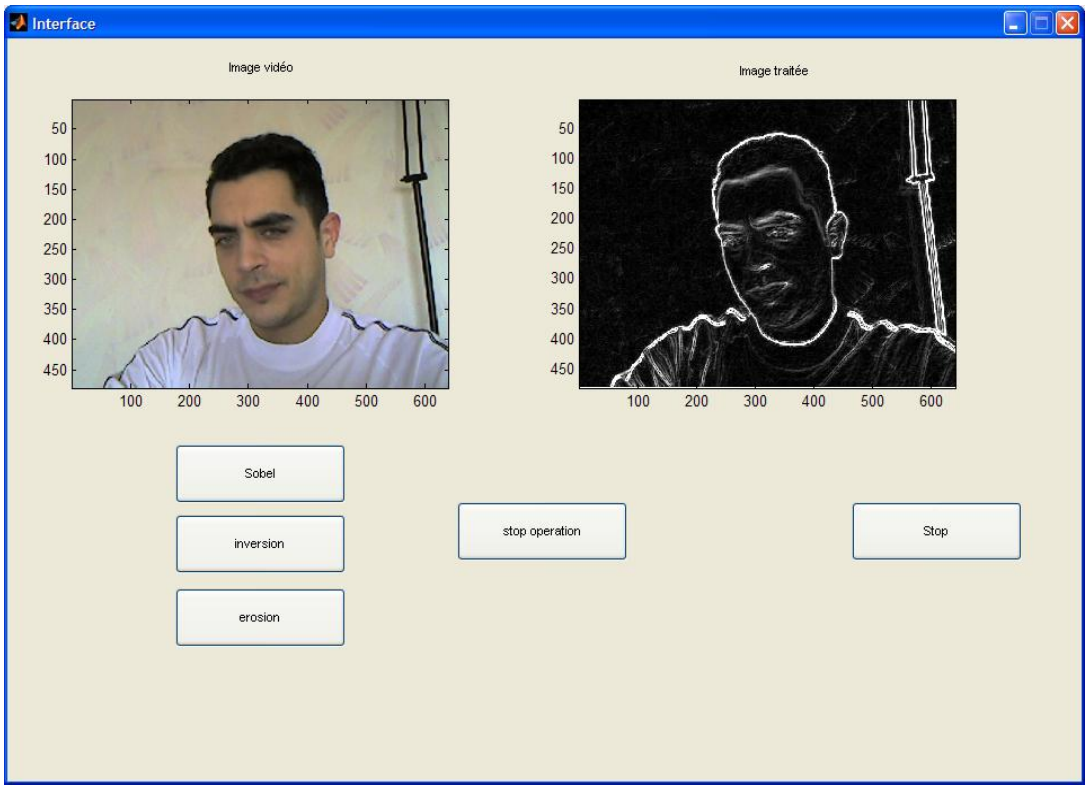
M1 INFO
Module : TI

TPs de Traitement d'images
M1 Info



C. Roudet
B. Billiot

Programme et objectifs

L'objectif de ces TP est de vous faire implémenter sous Matlab différents **outils de traitement et de compression d'images.**

Le premier TP consistera à intégrer dans une interface GUI Matlab, plusieurs traitements **en temps réel**. L'acquisition d'images se fera grâce à des **webcams**.

L'interface GUI Matlab contiendra :
- une fenêtre d'acquisition,
- une fenêtre de traitement,
- et un ensemble de boutons (associés à des fonctions) pour **lancer les traitements**.

Pour les TP suivants (2, 3 et 4), vous appliquerez les traitements sur des **images fixes**, sans utilisation d'interface **GUI Matlab.**

Programme des TP Matlab :

TP 1 : Histogramme, binarisation et LUT **avec GUI Matlab**

TP 2 : Etirement et égalisation d'histogrammes, filtrage de bruit

TP 3 : Détection de contours et segmentation d'images

TP 4 : Compression et qualité de reconstruction

TP 5 : TP noté d'1h (exercice applicatif sur une des notions vues)

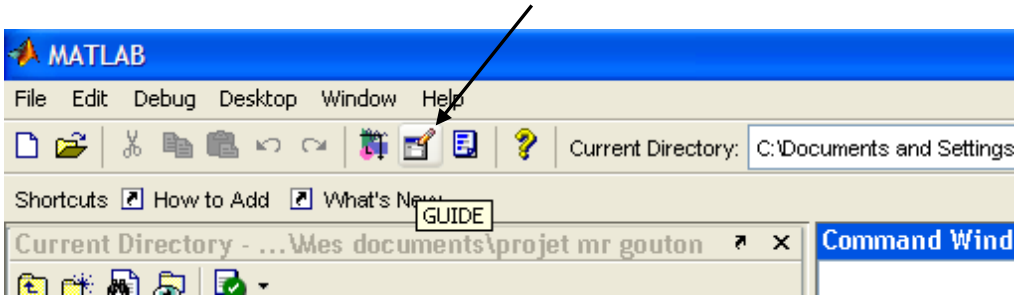
Programme à votre disposition et aide au démarrage

Le fichier **acquis_test_M1.m** vous aidera à démarrer le TP, il est à votre disposition ici :
http://ufrsciencestech.u-bourgogne.fr/~roudet/enseignement/TI/acquis_test_M1.m
Il doit être copié dans le répertoire de travail. Ce programme permet la **prise en main, la collecte d'information et les tests** de votre dispositif d'acquisition.

Une fois les informations de votre dispositif d'acquisition récupérées, nous allons créer un **programme d'acquisition et de traitement de base** (contenant une fonction d'inversion d'image, dont le code vous sera donné).
Par la suite, **votre travail** consistera à **adjoindre de nouvelles fonctionnalités**. L'ensemble des traitements se fera sur des images en niveau de gris.

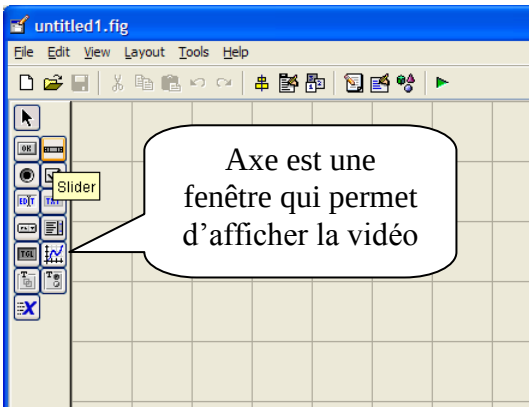
Création de l'interface graphique (GUIDE en matlab) :

- lancer *MATLAB*
- cliquer sur l'**icône « GUIDE »** (voir la figure ci-après)



- dans l'onglet « Create new GUI » choisir un des 4 modes (par défaut on prendra **« Blank GUI »**)

- on obtient une fenêtre avec sur la gauche les **différents objets** (push button, radio button, slider etc…) que l'on peut placer sur l'interface graphique.

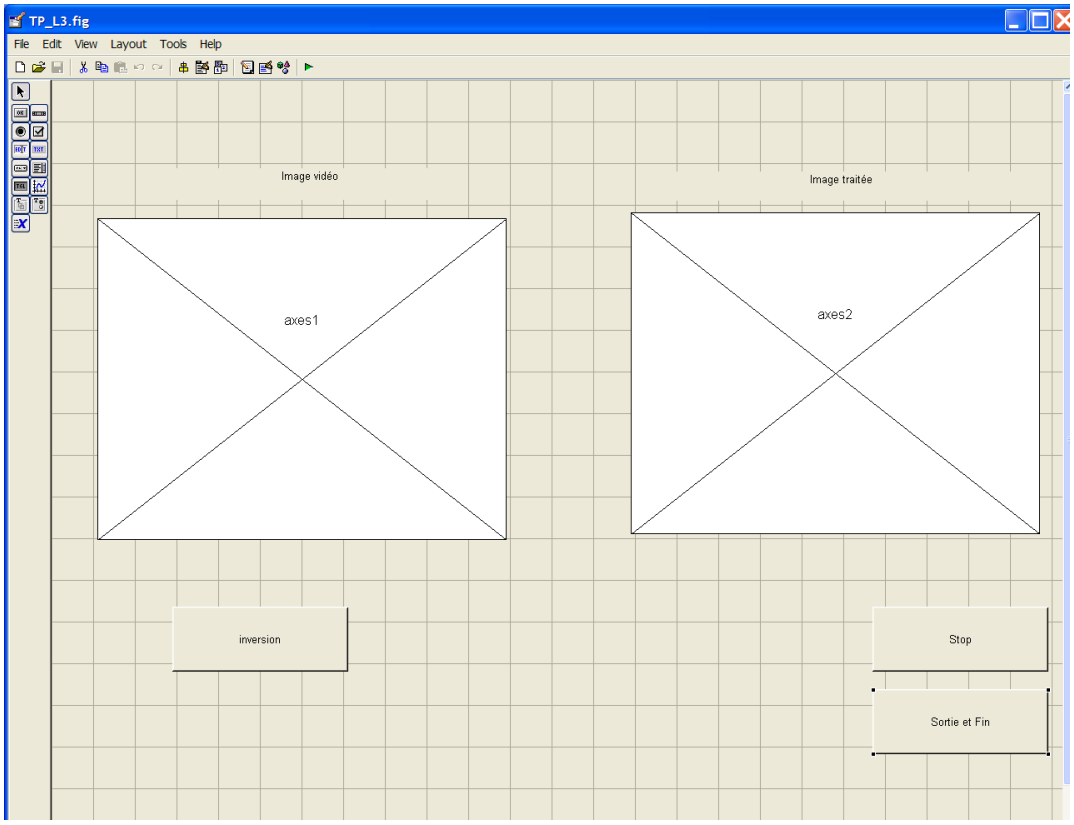


- Il suffit de **choisir un objet** et de le dessiner sur l'interface graphique :

1) choisir **« Axes »** et dessiner les fenêtres d'acquisition et de traitement (comme sur la figure de la page suivante).

2) Ajouter **3 boutons** (un pour réaliser l'**inversion d'image**, un pour **stopper le traitement en cours** et un dernier pour **sortir de l'interface**).

Aperçu du GUIDE demandé :



- Pour accéder aux **paramètres des boutons**, on double clique dessus et une fenêtre *Property Inspector* s'ouvre alors.

Les **paramètres à modifier** pour les boutons sont :

- **String** : le nom qui apparaît sur le bouton (sur l'interface)
- **Tag** : cela modifie le **nom de la fonction dans *MATLAB*** associée au bouton

Important : bien s'assurer que le nom avant « _Callback » (correspondant à la fonction associée au bouton dans Matlab) est bien le même que le « Tag ».

- Dès que l'on **sauvegarde le fichier « GUIDE »**, *MATLAB* génère automatiquement un **fichier .m portant le même nom** que le fichier « GUIDE ».

La **partie surlignée** (dans le code en page suivante) est la partie créée à partir d'un bouton de tag *« pushbutton1»*.

Il suffit de rajouter du code à la suite de cette partie pour ajouter des actions associées au bouton.

```
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

Le code commenté que vous devrez ajouter dans la fonction associée au **bouton Inversion** (pour réaliser l'**inversion vidéo**) est disponible ici :
http://ufrsciencestech.u-bourgogne.fr/~roudet/enseignement/TI/inversion.m

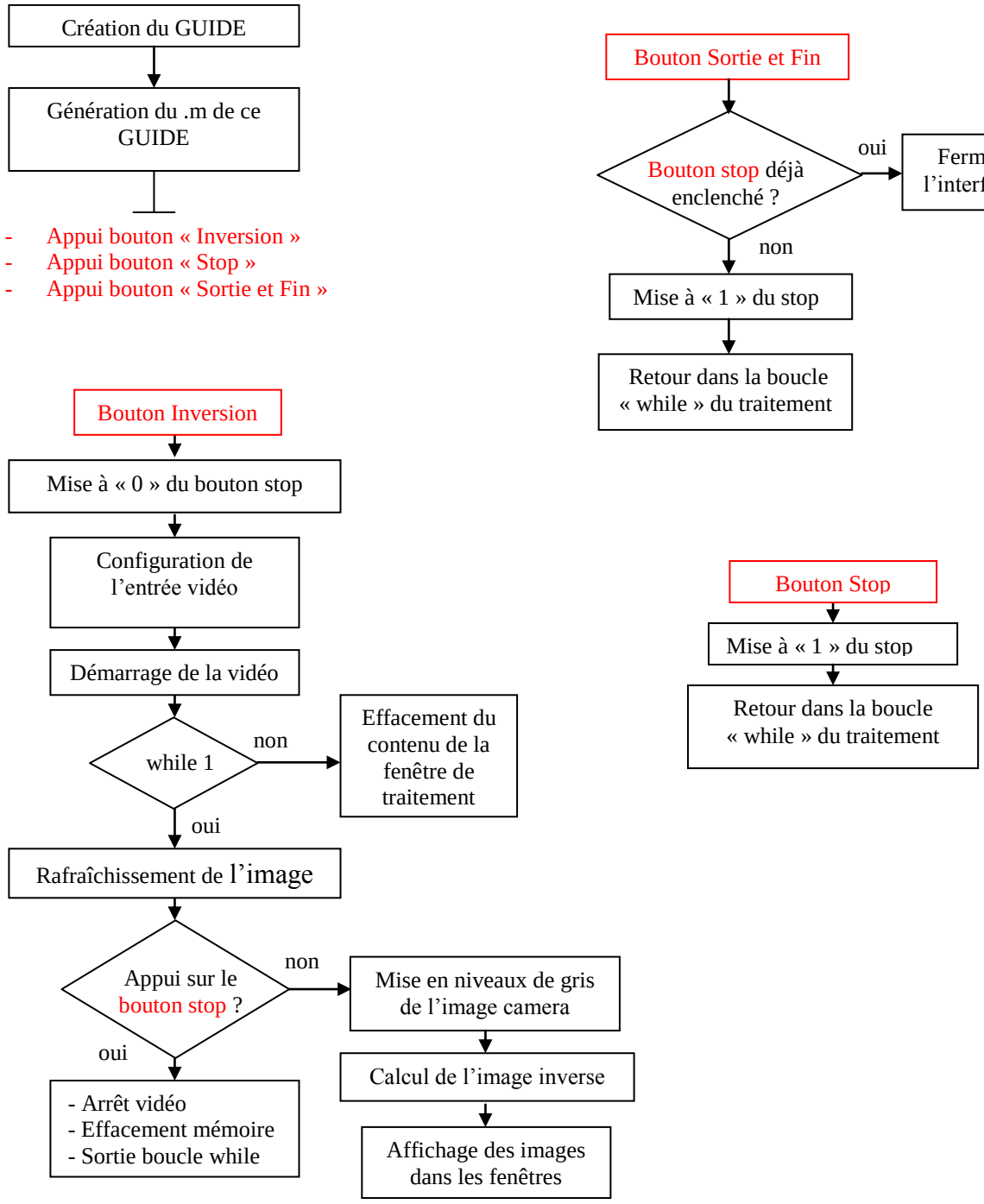
POUR FINIR : NE PAS OUBLIER D'AJOUTER CECI :

Dans la fonction suivante, générée par le GUIDE (où **nom_fichier** correspond au nom de vos fichiers **.fig et .m**) :

function varargout = **nom_fichier**_OutputFcn(hObject, eventdata, handles)
% varargout cell array for returning output args (see VARARGOUT);
% hObject   handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)

delete(imaqfind); // CODE A AJOUTER ICI

Algorithmes du programme demandé



Dans un premier temps faire fonctionner le programme créé en cliquant sur le **bouton « Inversion »**, puis **rajouter le code** pour faire fonctionner les **boutons « Stop »** et **« Sortie et Fin ».** Vous vous apercevrez qu'il faudra à chaque fois appuyer sur le bouton **« Stop »** avant de lancer un nouveau traitement (pour d'abord terminer le traitement en cours).

TP1 : Histogramme, binarisation et Look Up Tables (LUT)

Dans ce TP, nous allons créer **7 boutons : Histogramme, Binarisation man, Binarisation auto, LUT Inv, LUT R, LUT V et LUT B** dans l'interface créée précédemment (GUIDE).

I) Histogramme

Créer le **bouton Histogramme** dans l'interface (GUIDE : .fig) et compléter la fonction correspondante dans le fichier Matlab (.m)
Quand on clique sur ce bouton, on désire afficher :

- dans la fenêtre de gauche : l'image en niveau de gris provenant de la webcam
- dans la fenêtre de droite : l'histogramme de cette image.

On appelle **h** l'histogramme d'amplitude, défini de telle sorte que **h(n)** donne le nombre de pixels de l'image dont le niveau de gris vaut **n**.

Comparez votre histogramme avec celui fourni par la fonction *« imhist »*.
Attention : les indices d'une matrice ou d'un vecteur commencent à 1 !

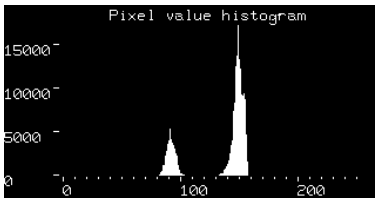
Qu'observe-t-on lorsqu'on fait varier la luminosité de l'image acquise ?

II) Binarisation

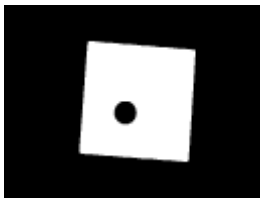
Créer le **bouton Binarisation** dans l'interface (GUIDE : .fig) et compléter la fonction correspondante dans le fichier Matlab (.m).
Quand on clique sur ce bouton, on désire voir afficher dans la fenêtre de gauche l'image en niveau de gris provenant de la webcam et la binarisation de cette image dans la fenêtre de droite :

1) Le **seuil** aura été choisi en ayant **visualisé l'histogramme** au préalable (on pourra demander à l'utilisateur de le taper au clavier).

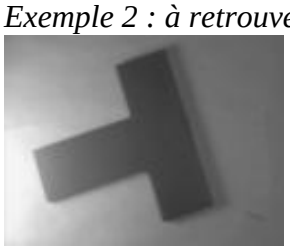
Exemple 1 : à retrouver en plus grand en Annexe 2

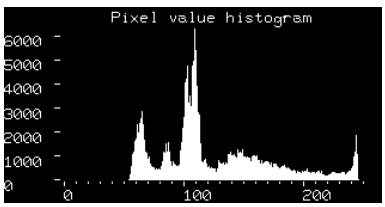


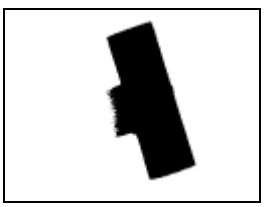
Résultat :



Exemple 2 : à retrouver en plus grand en Annexe 2







2) Le **seuil** sera déterminé **automatiquement** par la **fonction Matlab** *« graythresh »* (méthode d'**Otsu** détaillée en *Annexe 1*).
Remarque : on pourra afficher dans la fenêtre de traitement la valeur du seuil automatique.

Visualisez maintenant l'histogramme de l'image binarisée, que remarquez-vous ?

III) LUT

Sous Matlab, il est possible de créer ses **propres LUT** et de les appliquer à l'aide de la fonction **colormap**. Les valeurs doivent cependant être normalisées sur l'intervalle [0, 1].
Prenons, par exemple, le cas simple d'une matrice M de taille 3x3 :

$$\begin{bmatrix} 4 & 4 & 1 \\ 1 & 2 & 2 \\ 3 & 3 & 4 \end{bmatrix}$$

Cette matrice comprend quatre valeurs distinctes. On se propose d'afficher : « 1 » en noir, « 2 » en blanc, « 3 » en rouge, et « 4 » en vert.
Pour cela on crée une LUT « **map4C** » dont les sorties sont les quatre couleurs souhaitées. Pour appliquer cette LUT à l'image affichée, on tape la commande **colormap(map4C)** :

```
M = [4 4 1 ; 1 2 2 ; 3 3 4] ;
% Définition de la LUT = palette de couleurs indexées (tableau)
r = [0 1 1 0];
v = [0 1 0 1];
b = [0 1 0 0];
map4C = [r' v' b'];  % création de la LUT qui convertit les pixels à 1 en noir (0, 0, 0)
image(M)
colormap(map4C)    % indique la palette de couleurs indexées considérée
```

1) Créer le **bouton LUT Inv** dans l'interface (GUIDE : .fig) et compléter la fonction correspondante dans le fichier Matlab (.m) pour recréer l'**inversion vidéo** (réalisée précédemment à partir d'une **image en niveaux de gris**) grâce à une LUT.

2) Enfin créer les **boutons LUT R, LUT V et LUT B** dans l'interface (GUIDE : .fig) et compléter la fonction correspondante dans le fichier Matlab (.m) pour **visualiser plan par plan** l'image couleur, en créant les LUT adéquates pour les **plans Rouge, Vert et Bleu**.

Rappel : pour une image couleur **Im**, le canal Rouge s'obtient grâce à l'instruction suivante :

```
R = Im( : , : , 1) ;     % Plan Rouge
```

Annexe 1 : méthode d'Otsu

Il existe plusieurs méthodes permettant de déterminer automatiquement le **seuil de binarisation**. Une des plus connues est la **méthode d'Otsu**.

On considère une image **A** (de taille **l x c**), d'histogramme **h(n)** et dont les niveaux de gris **n** appartiennent à **[0, L].** La probabilité **p(n)** a priori d'un niveau de gris **n** est alors définie par :

$$p(n) = h(n) / (l \cdot c)$$

La fonction **p(x)** est alors une **densité de probabilité**. En effet, on peut alors vérifier que :

$$\sum_{n=0}^{L} p(n) = 1$$

Il est alors possible de déterminer la **moyenne** et la **variance des niveaux de gris** :

$$\mu = \sum_{n=0}^{L} n.p(n) \qquad \sigma^2 = \sum_{n=0}^{L} (n-\mu)^2 p(n)$$

La **méthode d'Otsu** repose sur l'hypothèse que les pixels de l'image appartiennent à **deux classes C_1 et C_2**. Si on décide que les pixels appartiennent à C_1 si leur niveau est inférieur ou égal à **t** et appartiennent à C_2 sinon, on peut calculer la **probabilité a priori de C_1 et C_2**.

$$P(C_1) = \sum_{n=0}^{t} p(n) \qquad P(C_2) = \sum_{n=t+1}^{L} p(n)$$

Les **moyennes** et **variances** des niveaux de gris de C_1 et C_2 sont alors données par :

$$\mu_{C_1} = \frac{\sum_{n=0}^{t} n.p(n)}{P(C_1)} \qquad \mu_{C_2} = \frac{\sum_{n=t+1}^{L} n.p(n)}{P(C_2)}$$

$$\sigma^2_{C_1} = \frac{\sum_{n=0}^{t} (n-\mu)^2 p(n)}{P(C_1)} \qquad \sigma^2_{C_2} = \frac{\sum_{n=t+1}^{L} (n-\mu)^2 p(n)}{P(C_2)}$$

On montre que : $\mu = \mu_{C_1}.P(C_1) + \mu_{C_2}.P(C_2)$

Par ailleurs, on appelle σ^2_{intra} la variance intra-classe et σ^2_{inter} la variance interclasse définies par :

$$\sigma^2_{intra} = P(C_1)\sigma^2_{C_1} + P(C_2)\sigma^2_{C_2}$$

$$\sigma^2_{inter} = P(C_1)(\mu_{C_1} - \mu)^2 + P(C_2)(\mu_{C_2} - \mu)^2$$

On montre que σ^2 (variance des niveaux de gris de l'image) est indépendante du seuil de binarisation et que :

$$\sigma^2 = \sigma^2_{inter} + \sigma^2_{intra}$$

En revanche, σ^2_{intra} et σ^2_{inter} dépendent de la séparation en classes.

Selon Otsu, la **séparation optimale** est obtenue en **minimisant la variance intra-classe** et en **maximisant la variance interclasse**.

Or la **somme de ces deux variances est constante**, de ce fait, maximiser la variance interclasse est équivalent à minimiser la variance intra-classe. Le seuil idéal est alors obtenu pour le maximum de σ^2_{inter}.

Annexe 2 : Images pour les tests (binarisation)