CONCOURS COMMUNS POLYTECHNIQUES

EPREUVE SPECIFIQUE - FILIERE PSI

INFORMATIQUE

Durée : 3 heures

N.B. : le candidat attachera la plus grande importance à la clarté, à la précision et à la concision de la rédaction. Si un candidat est amené à repérer ce qui peut lui sembler être une erreur d'énoncé, il le signalera sur sa copie et devra poursuivre sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.

Les calculatrices sont interdites

Le sujet comporte 12 pages dont :
- 9 pages de texte de présentation et énoncé du sujet ;
- 3 pages d'annexe.

Toute documentation autre que celle fournie est interdite.

REMARQUES PRELIMINAIRES

L'épreuve peut être traitée en langage **Python** ou langage **Scilab**. Il est demandé au candidat de bien préciser sur sa copie le choix du langage et de rédiger l'ensemble de ses réponses dans ce langage. Les syntaxes Python et Scilab sont rappelées en annexe **V.1**, page 10.

Les différents algorithmes doivent être rendus dans leur forme définitive sur la copie en respectant les éléments de syntaxe du langage choisi (les brouillons ne seront pas acceptés).

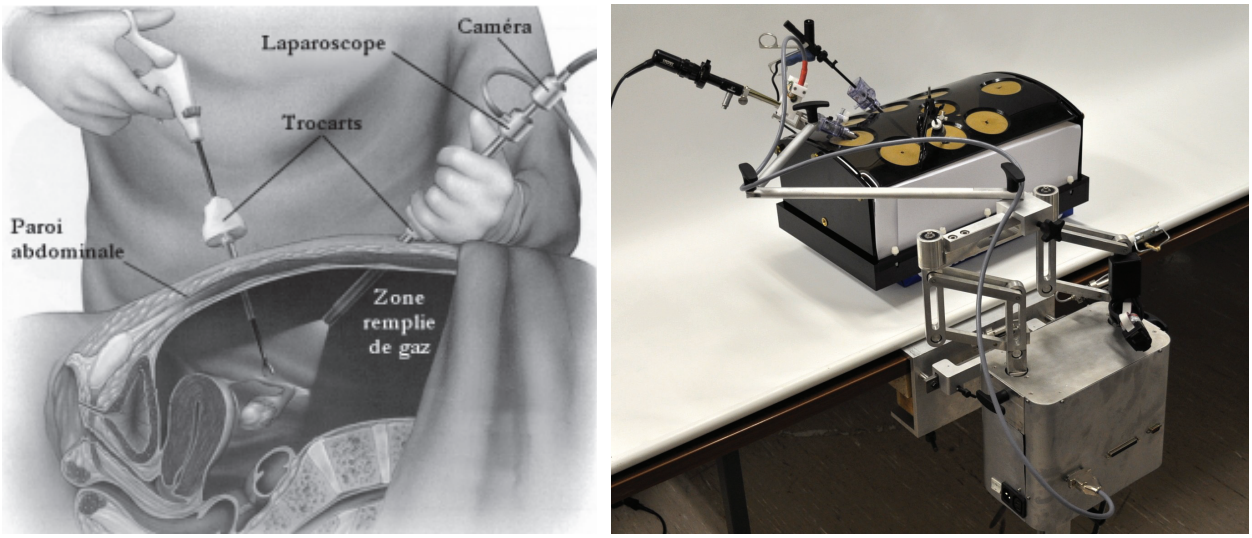
Il est demandé au candidat de bien vouloir rédiger ses réponses **en précisant bien le numéro de la question traitée et, si possible, dans l'ordre des questions.** La réponse ne doit pas se cantonner à la rédaction de l'algorithme sans explication, les programmes doivent être expliqués et commentés.

Robot Evolap - Suivi d'instrument chirurgical

I Présentation

La laparoscopie est une technique chirurgicale mini-invasive de diagnostic et d'intervention abdominale. Une ou plusieurs petites incisions sont réalisées dans la paroi abdominale pour y insérer, au travers de canules appelées trocarts, de longs instruments chirurgicaux et un endoscope rigide - appelé laparoscope - surmonté d'une caméra. Ce laparoscope est également connecté à une source de lumière froide pour éclairer la cavité abdominale. Le robot EVOLAP est un robot d'assistance à la chirurgie laparoscopique développé par l'Université Catholique de Louvain (UCL). Il permet d'éviter à l'assistant chirurgical de porter le laparoscope pendant les opérations et améliore le bon déroulement de celles-ci.

Objectif
L'objectif de l'étude proposée est de réaliser le programme de suivi de l'instrument chirurgical par le robot EVOLAP, ce qui permet au chirurgien une vision optimale de la zone d'intervention.



a - Principe de la laparoscopie

b - Robot Evolap avec simulateur abdominal

Figures 1 – Laparoscope manuel et robotisé.

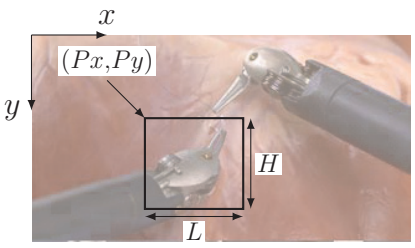
Le robot possède deux moteurs qui permettent d'orienter le laparoscope selon deux angles indépendants autour du point de passage de l'appareil.

Le programme de suivi d'un instrument chirurgical à partir de l'analyse d'image de la zone de travail est constitué de plusieurs parties fondamentales :
- définition du motif à suivre dans l'image,
- acquisition d'une nouvelle image (issue du flux vidéo de la caméra),
- recherche du motif dans la nouvelle image,
- pilotage des moteurs pour aligner la caméra sur le motif détecté.

Dans tout le sujet, il sera supposé que les modules et bibliothèques sont déjà importés dans le programme.

II Définition du motif à suivre dans l'image

A partir du flux vidéo, un module permet d'extraire une image en couleur (RGB) à chaque instant de l'analyse. On notera `image` cette image. On définit sa largeur par la variable `image_width` et sa hauteur par la variable `image_height` définies en pixels (ou points) de 800 × 600. Le point supérieur gauche de l'image a pour coordonnées [0,0], le point inférieur droit [799,599] en Python et respectivement (1,1) et (800,600) en Scilab.



Pour définir le motif à suivre dans l'image, l'utilisateur doit spécifier les coordonnées (en pixels) du point supérieur gauche (Px,Py) d'une fenêtre ainsi que sa largeur L et sa hauteur H.

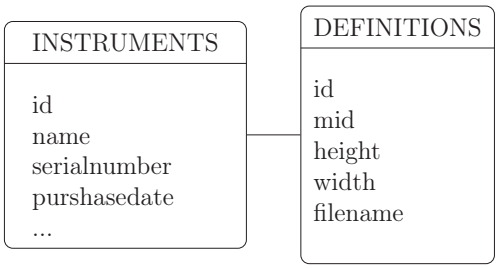
La fonction `demande_valeur(text)` est utilisée pour récupérer **une** valeur entière demandée à l'utilisateur. La vérification du type de donnée renvoyée par cette fonction est implantée dans celle-ci. L'argument `text` est le message affiché à l'écran pour demander la valeur à l'utilisateur.

Q1. Définir une fonction `demande_fenetre()` qui utilise la fonction `demande_valeur(text)` pour demander à l'utilisateur de définir la zone d'intérêt et qui renvoie une variable sous forme de tableau contenant les 4 informations recueillies `[Px,Py,L,H]`. Dans le programme principal, le retour de cette fonction sera nommé `fenetre`.

Q2. Définir une fonction `verification_fenetre(image_width,image_height,fenetre)` qui vérifie que la fenêtre soit bien à l'intérieur de l'image et qui renvoie un booléen `True` si la fenêtre est définie correctement et `False` sinon.

Plutôt que de spécifier manuellement la zone à suivre, l'utilisateur peut choisir un motif correspondant à un instrument chirurgical donné à partir d'une base de données, qui permet de connaître les vues d'un instrument particulier quelle que soit son orientation, sa taille...

Cette base de données est constituée de deux tables.
La table INSTRUMENTS contient les différents types d'instruments avec les attributs :
- id : identifiant de type entier, clé primaire,
- name : nom de l'instrument de type texte,
- serialnumber : numéro de série du produit,
- purshasedate : date d'achat de l'instrument,
- autres attributs non détaillés...



La table DEFINITIONS contient des entités qui correspondent à la définition d'un instrument pour une configuration donnée (nommée motif). Elle contient les attributs :
- id : identifiant de type entier, clé primaire,
- mid : identifiant de l'instrument correspondant à la définition de type entier,
- height : hauteur en pixels du motif de type entier,
- width : largeur en pixels du motif de type entier,
- filename : nom du fichier image de type texte (stocké sur réseau) correspondant à ce motif.

A un instrument peut correspondre une ou plusieurs définitions.

Q3. Ecrire une requête SQL permettant de récupérer les noms de toutes les images qui correspondent à l'instrument dont le nom est `"pince"`.

III Pilotage des moteurs pour déplacer la caméra

La caméra envoie à chaque instant t une image en couleur. L'image à l'instant $t + dt$ est appelée image courante tandis que l'image à l'instant t est appelée image précédente.

La partie suivante va permettre de comprendre comment extraire de l'image courante une nouvelle fenêtre homothétique de la fenêtre définie dans l'image précédente.

La nouvelle fenêtre étant déterminée, on cherche ensuite à recentrer la caméra sur l'image, c'est-à-dire que la fenêtre doit se trouver au centre de l'image. On suppose que, sur la première image, la fenêtre autour de l'instrument est bien centrée.

Q4. Ecrire une fonction `centre(fenetre)` permettant de calculer le centre de la fenêtre définie par `fenetre=[Px,Py,L,H]`.

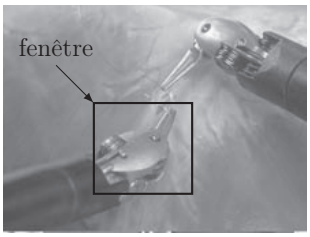
Cela permet de calculer la loi de mouvement à imposer aux moteurs du robot qui porte le laparoscope afin de suivre la fenêtre d'étude.

IV Recherche d'un motif dans une image

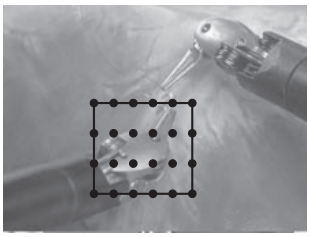
La difficulté de l'algorithme est de déterminer la fenêtre dans la nouvelle image à partir de l'analyse du mouvement des pixels de la fenêtre entre l'ancienne image et la nouvelle.

Après un traitement de l'image courante (conversion en niveaux de gris), connaissant la fenêtre dans l'image précédente (figure 2a), on extrait tout d'abord de la fenêtre d'étude quelques pixels d'intérêt répartis régulièrement (figure 2b). On recherche ensuite le déplacement moyen de chacun de ces pixels d'une image à l'autre. Il est alors nécessaire de procéder à une élimination des pixels pour lesquels le déplacement obtenu est incohérent. Plusieurs vérifications sont donc mises en place pour décider de l'élimination de ces pixels aberrants. Une fois le motif identifié dans l'image courante, on estime le grossissement possible de ce motif et on renvoie la nouvelle fenêtre d'étude qui englobe ces pixels (figure 2c).

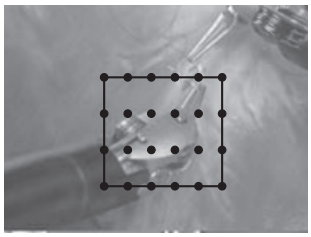
Les parties suivantes vont permettre de spécifier les algorithmes et fonctions nécessaires à la recherche du motif dans l'image courante.



a - Image instant t et fenêtre entourant le motif



b - Image instant t et fenêtre avec les pixels d'intérêt



c - Image instant $t + dt$ et fenêtre avec les pixels d'intérêt

Figures 2 – Illustrations de la recherche de la fenêtre définie dans une image dans une nouvelle image.

IV.1 Traitement de l'image courante

L'image acquise par la caméra est une image en couleur constituée de trois couches (RGB). Les données de l'image sont stockées dans un tableau à trois dimensions : la première dimension correspond à la couleur (rouge, vert ou bleu, indice 0, 1 ou 2), la seconde correspond à la coordonnée selon $\vec{x}$ et la troisième à la coordonnée selon $\vec{y}$. Ainsi, les dimensions du tableau sont : $3 \times m \times n$ où $m = 800$ et $n = 600$.

La valeur associée à chaque pixel est un entier compris entre 0 et 255.

Q5. Donner la quantité de mémoire nécessaire en octets pour stocker le tableau représentant l'image émise par la caméra en justifiant le codage retenu pour un pixel d'une couche.

La première étape de l'algorithme de suivi d'image est de convertir l'image en couleur en niveaux de gris. La méthode consiste à rechercher le maximum et le minimum pour chaque pixel sur les trois couches, puis à faire la moyenne de ces maxima et minima et ainsi obtenir la valeur du nouveau pixel en niveaux de gris.

On note `imagecolor` le tableau représentant une image en couleur.

Q6. Ecrire une fonction `grayscale(imagecolor)` qui renvoie une image en niveaux de gris (qui sera notée `image` dans l'algorithme principal), sous forme de tableau à deux dimensions de taille $m \times n$ contenant des valeurs entières comprises entre 0 et 255, en suivant l'algorithme décrit ci-dessus. Remarque : il est possible d'utiliser les fonctions `min` et `max` internes au langage choisi.

IV.2 Extraction des pixels d'intérêt de la fenêtre d'étude

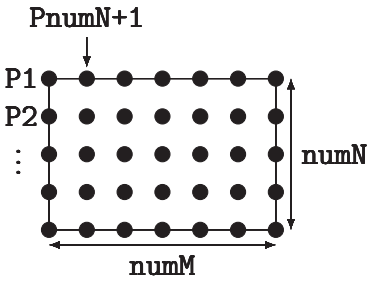
Pour déterminer le déplacement du motif, il est nécessaire d'extraire les pixels de la fenêtre définie initialement. Pour améliorer l'efficacité de la stratégie, on ne sélectionne pas tous les pixels mais uniquement un petit nombre, nommés pixels d'intérêt, de cette fenêtre (figure 2b).

`numM` > 1 est le nombre de points souhaités horizontalement.

`numN` > 1 est le nombre de points souhaités verticalement.

Les points doivent être équirépartis sur la zone décrite par la fenêtre.

Un point est défini par ses coordonnées `x` et `y`, respectivement l'indice de la colonne et l'indice de la ligne, en prenant comme origine le coin supérieur gauche de l'image, bord inclus.



Q7. Ecrire une fonction `construction_coordonnees_pts(fenetre,numM,numN)` qui renvoie un tableau de points noté `pts=[P1x,P1y,P2x,P2y,...]` dans l'algorithme principal (le parcours des points se fait de haut en bas puis de la gauche vers la droite). On ne traitera pas les cas pour lesquels `numN` ou `numM` sont égaux à 1. On rappelle que `fenetre=[Px,Py,L,H]`.

IV.3 Recherche des pixels d'intérêt dans l'image courante

On note `imgJ` le tableau à deux dimensions permettant d'accéder aux valeurs des pixels en niveau de gris de l'image courante (au temps $t + dt$) et `imgI` celui pour l'image précédente (temps t).

On utilise la méthode de Lucas-Kanade pour trouver les pixels d'intérêt de l'image précédente dans l'image courante.

Cette méthode suppose que le déplacement d'un point de l'image entre deux instants consécutifs est petit et se fait à vitesse constante. Pour estimer ce déplacement, on considère une zone de quelques pixels centrée autour du pixel d'intérêt étudié et on suppose que le déplacement est

le même pour tous les pixels de cette zone. La résolution de ce problème se fait par la méthode des moindres carrés explicitée dans la suite du sujet.

IV.3.1 Définition de l'algorithme

Pour mettre en place la méthode de Lucas-Kanade, il est nécessaire de partir d'une définition continue de l'image et de l'algorithme.

On note $I(t)$ l'image en niveaux de gris à l'instant t et $I(t+dt)$ l'image en niveaux de gris obtenue à un instant $t+dt$. Pour repérer un point de l'image, on utilise la notation $I(t)[P]$ qui désigne la valeur en niveaux de gris de l'image à l'instant t au point P de coordonnées $[x,y]$.

Soit u un point de coordonnées $[ux,uy]$ sur l'image $I(t)$.

L'algorithme permet de trouver les coordonnées de ce point dans l'image $I(t+dt)$ (notées $v=[vx,vy]=u+d$) en supposant que la valeur du pixel ne change pas, c'est-à-dire :
$I(t)[u]=I(t+dt)[v]$ (on parle de stationnarité de la fonctionnelle $I(x,y,t)$).

Le vecteur $d=[dx=V_x dt, dy=V_y dt]$ représente le déplacement du pixel à la vitesse constante $V=[V_x,V_y]$. Les variations dx, dy autour d'un point sont exprimées en pixels. En pratique, le déplacement n'est que de quelques pixels.

Ainsi, étudier la stationnarité du pixel revient à résoudre le problème suivant :

$$Ix\,dx + Iy\,dy = -It\,dt$$

où $Ix = \dfrac{\partial I(t)[u]}{\partial x}$ et $Iy = \dfrac{\partial I(t)[u]}{\partial y}$ sont les dérivées spatiales (dérivées en un point de coordonnées u suivant les directions x ou y de l'image $I(t)$) et $It = \dfrac{\partial I(t)[u]}{\partial t}$ est la dérivée temporelle de l'image (dérivée en un point de coordonnée u entre l'image $I(t)$ et $I(t+dt)$).

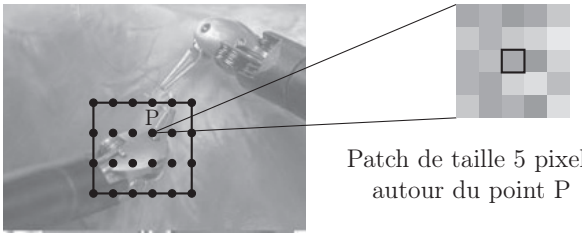
Q8. Proposer des approximations pour définir numériquement les termes Ix, Iy et It en fonction de `imgI`, `imgJ`, `ux`, `uy` et `dt`. On ne traitera pas les cas où le point u est sur le bord de l'image. Pour définir une dérivée numérique spatiale, on prendra des variations dx, dy égales à ± 1 pixel.

On note $Itdt$ le produit It par le pas de temps dt défini par le rafraîchissement des images de la caméra. On définit la fonction `calc_LK_terms(pixP,imgI,imgJ)` qui permet de calculer et de retourner les termes de l'équation de stationnarité, `Ix,Iy,Itdt`, définie précédemment pour un pixel `pixP` donné :

$$Ix\,dx + Iy\,dy = -Itdt.$$

On suppose cette fonction connue ce qui permet de répondre à la suite du problème indépendamment de la spécification de la question précédente.

IV.3.2 Implantation et résolution



Patch de taille 5 pixels autour du point P

L'algorithme indique que la relation précédente est écrite pour quelques pixels dans un voisinage proche d'un pixel d'intérêt P appartenant au motif étudié (question **Q7**, page 5). Il est donc nécessaire de définir ce voisinage autour du point P que l'on appellera `patch`. Ce voisinage contiendra 5×5 points contigus dont le point central sera le point P. On fait l'hypothèse que le point P n'est pas près du bord et qu'il y a toujours suffisamment de pixels disponibles autour du point P pour définir le patch.

Q9. Ecrire une fonction `creation_patch(P,patch_size)` où `P=[Px,Py]` est le pixel étudié et `patch_size=5` est le nombre de points choisis horizontalement et verticalement dans le voisinage du point P. Cette fonction renverra un tableau de coordonnées `patch=[P1x,P1y,P2x,P2y...]` classé du haut vers le bas et de gauche à droite comme à la question **Q7**, page 5. On supposera que la variable `patch_size` est impaire.

En écrivant l'équation de stationnarité pour tous les pixels du patch et en supposant que le déplacement est le même que celui du point P, on obtient un système composé de `patch_size` $\times$ `patch_size` équations à 2 inconnues ($d = [dx,dy]$). On doit ainsi résoudre un système qui s'écrit sous la forme $A.d = b$.

Q10. Ecrire une fonction `calc_Ab(imgI,imgJ,patch)`, utilisant la fonction `calc_LK_terms`, qui renvoie un tableau de taille (`patch_size` $\times$ `patch_size`) $\times$ 2 (noté `A` dans la suite) et un vecteur (noté `b`) de taille (`patch_size` $\times$ `patch_size`) $\times$ 1.

Le système $A.d = b$ n'a pas de solution. On cherche cependant un déplacement d qui minimise au sens des moindres carrés la norme quadratique de $Ad - b$. Ceci revient à résoudre le système matriciel 2×2 : $A^T A.d = A^T.b$ où A^T représente la transposée de A.

Q11. Ecrire une fonction `resoud_LK(A,b)` qui renvoie les deux composantes dx et dy de d. On supposera que la matrice $A^T A$ est inversible et on ne fera donc aucune vérification concernant son inversibilité. Détailler la résolution sans utiliser de solveur interne au langage choisi en séparant et en commentant correctement chaque partie de l'algorithme.

IV.3.3 Synthèse : algorithme de recherche de pixels déplacés

Les sous-fonctions définies dans les parties précédentes permettent de déterminer le déplacement des pixels d'intérêt de la fenêtre de l'image précédente `imgI` et ainsi de retrouver ces pixels dans l'image courante `imgJ`.

On définit alors le tableau des pixels trouvés noté `fpts` dans l'image courante `imgJ` et stocké de la même manière que le tableau `pts` défini à la question **Q7**, page 5.

Q12. Définir une fonction `recherche_points(imgI,imgJ,pts)` qui utilise les fonctions précédentes et qui renvoie un tableau de points qui correspondent aux pixels déplacés. La résolution du système pouvant donner des valeurs non entières de pixels, on approchera les solutions obtenues par leur partie entière. On supposera de plus que cette résolution fournit toujours une solution.

Q13. Expliquer, en quelques phrases, les limites de l'algorithme proposé en discutant des cas qui pourraient ne pas fournir de solution.

IV.4 Vérification des points obtenus

Les points obtenus par résolution du système ne correspondent pas forcément à chaque pixel déplacé. C'est pourquoi il est nécessaire de mettre en place une ou plusieurs vérifications pour éliminer les points qui ne conviennent pas.

La première vérification consiste à s'assurer qu'il est possible de retrouver les points initiaux de `imgI` en partant des points obtenus dans l'image courante `imgJ` par la même méthode. Par exemple, `P1` dans `imgI` donne `P1n` dans `imgJ`, qui donne `P1nn` dans `imgI`. Si `P1nn` est différent de `P1`, alors il est éliminé, sinon il est conservé.

Q14. Ecrire la partie du programme principal qui permet d'obtenir un tableau de booléen (noté `statut` dans le programme principal) de la taille du nombre de points représentant le motif. Chaque case contient `True` si le point a été retrouvé et `False` sinon. On utilisera les variables définies tout au long du sujet et la fonction `recherche_points`.

La deuxième vérification fait appel à la fonction suivante qui utilise les pixels sélectionnés (points1) de l'image précédente et les pixels trouvés (points2) de l'image courante.

Définition Python

```
def calcul1(points1,points2):
  return sqrt((points1[0::2]-points2[0::2])**2+(points1[1::2]-points2[1::2])**2)
```

Définition Scilab

```
function calcul1(points1,points2)
  return sqrt((points1(1:2:$)-points2(1:2:$))^2+(points1(2:2:$)-points2(2:2:$))^2)
endfunction
```

Q15. Indiquer quelle opération mathématique est réalisée par cette fonction `calcul1` et préciser le type et la dimension de ce que renvoie la fonction.

Pour éliminer des points, on calcule la médiane des valeurs obtenues et on supprime les points dont la valeur est trop éloignée de la médiane.

Q16. Ecrire une fonction `mediane(a)`, basée sur un algorithme de tri de votre choix, qui renvoie la valeur médiane du tableau de valeurs `a`. On suppose que les valeurs de `a` sont strictement positives et que la longueur de `a` est impaire.

Q17. Nommer l'algorithme de tri utilisé et donner sa complexité dans le meilleur et le pire des cas.

Q18. Ecrire une fonction `verification_pts(pts,fpts,statut)` qui renvoie un nouveau tableau de points (noté `nouveaux_pts` dans le programme principal) pour lesquels le premier critère est vérifié et la valeur obtenue par le deuxième critère est inférieure ou égale à la médiane.

Il est possible d'ajouter un troisième critère pour plus de précision concernant cette détection. On peut par exemple calculer une corrélation croisée (CCORR) entre les images `imgI` et `imgJ` sur des patchs autour de chaque point de la fenêtre d'étude. Des librairies permettent de réaliser cette corrélation croisée normée. La documentation de la fonction correspondante est détaillée dans l'annexe **V.2**.

Q19. En utilisant cette documentation, expliquer quelles commandes taper pour calculer cette corrélation croisée normée pour chaque point. Préciser ce que renvoie la fonction. On utilisera à nouveau la fonction `creation_patch(P,patch_size)`.

IV.5 Définition de la nouvelle fenêtre

Le tableau des points trouvés dans l'image courante peut être inclus dans une nouvelle fenêtre image de la fenêtre définie initialement. Celle-ci a subi une homothétie appelée facteur de grossissement.

On donne en annexe **V.3** la fonction `determine_fenetre` permettant de renvoyer la nouvelle fenêtre dans l'image `imgJ` et le facteur de grossissement à partir des points valides trouvés dans la nouvelle image. Cette fonction a été développée par une autre personne. Le développeur a utilisé sa propre spécification des variables : c'est-à-dire que la façon de stocker les grandeurs d'entrées et de sorties est différente de celle définie dans le programme principal et adoptée depuis le début du sujet. On sait que `bb0` correspond à la fenêtre dans l'image précédente, `pt0` aux points dans l'image précédente et `pt1` aux points dans l'image courante.

Q20. Préciser les variables qui ont été définies différemment et indiquer ce qui doit être modifié afin que la fonction puisse être utilisée dans le programme principal. Il n'est pas demandé de réécrire cette fonction.

Q21. Indiquer sur votre copie ce que font les parties de programmes entre les zones de commentaires en précisant le numéro du commentaire (`com1` à `com6`). Des commentaires d'une à deux lignes maximum sont attendus.

Fin de l'énoncé

V Annexes

V.1 Rappels des syntaxes en Python et Scilab

Remarque : sous Python, l'import du module numpy permet de réaliser des opérations pratiques sur les tableaux : from numpy import *. Les indices de ces tableaux commencent à 0.

Remarque : sous Scilab, les indices des tableaux commencent à 1.

	Python	Scilab
tableau à une dimension	`L=[1,2,3]` (liste) `v=array([1,2,3])` (vecteur)	`v=[1, 2, 3]` ou `[1 2 3]`
accéder à un élément	`v[0]` renvoie 1	`v(1)` renvoie 1
ajouter un élément	`L.append(5)` uniquement sur les listes	`v($+1) = 5`
tableau à deux dimensions (matrice) :	`M=array(([1,2,3],[3,4,5]))`	`M=[1,2,3;3,4,5]`
accéder à un élément	`M[1,2]` ou `M[1][2]` donne 5	`M(2,3)` donne 5
extraire une portion de tableau (2 premières colonnes)	`M[:,0:2]`	`M(:,1:2)`
tableau de 0 (2 lignes, 3 colonnes)	`zeros((2,3))`	`zeros(2,3)`
dimension d'un tableau `T` de taille (i,j)	`T.shape` donne `[i,j]`	`length(T)` donne `(i,j)`
séquence équirépartie quelconque de 0 à 10.1 (exclus) par pas de 0.1	`arange(0,10.1,0.1)`	`[0:0.1:10]`
définir une chaîne de caractères	`mot="Python et Scilab"`	`mot="Python et Scilab"`
taille d'une chaîne	`len(mot)`	`length(mot)`
extraire des caractères	`mot[2:7]`	`part(mot,[11:16])`

boucle For	`for i in range(10):`
	`  print(i)`
		`for i=1:10`
		`  disp(i);`
		`end`

condition If	`if (i>3):`
	`  print(i)`
	`else`
	`  print("hello")`
		`if (i>3) then`
		`  disp(i)`
		`else`
		`  disp("hello")`
		`end`

définir une fonction qui possède un argument et renvoie 2 résultats	`def fonction(param):`
	`  res=param`
	`  res2=param*param`
	`  return res1,res2`
		`function [res,res2]=fonction(param)`
		`  res=param`
		`  res2=param*param`
		`endfunction`

V.2 Documentation de la fonction `match_template`

Compares a template against overlapped image regions.

Python : `result=cv2.matchTemplate(image, templ, method)`
Scilab : `result=matchTemplate(image, templ, method)`

Parameters :

- `image` : Image where the search is running. It must be 8-bit or 32-bit floating-point.
- `templ` : Searched template. It must be not greater than the source image and have the same data type. `result` : Map of comparison results. It must be single-channel 32-bit floating-point. If `image` is $W \times H$ and `templ` is $w \times h$, then `result` is $(W - w + 1) \times (H - h + 1)$.
- `method` : Parameter specifying the comparison method (see below).

The function slides through `image` , compares the overlapped patches of size $w \times h$ against `templ` using the specified method and stores the comparison results in `result` . Here are the formulae for the available comparison methods (I denotes `image`, T `template`, R `result`). The summation is done over template and/or the image patch : $x' = 0...w - 1, y' = 0...h - 1$

- method=CV_TM_SQDIFF

$$R(x,y) = \sum_{x',y'} (T(x',y') - I(x + x',y + y'))^2$$

- method=CV_TM_SQDIFF_NORMED

$$R(x,y) = \frac{\sum_{x',y'} (T(x',y') - I(x + x',y + y'))^2}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x + x',y + y')^2}}$$

- method=CV_TM_CCORR

$$R(x,y) = \sum_{x',y'} (T(x',y') \cdot I(x + x',y + y'))$$

- method=CV_TM_CCORR_NORMED

$$R(x,y) = \frac{\sum_{x',y'} (T(x',y') \cdot I(x + x',y + y'))}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x + x',y + y')^2}}$$

- method=CV_TM_CCOEFF

$$R(x,y) = \sum_{x',y'} (T'(x',y') \cdot I'(x + x',y + y'))$$

where

$$T'(x',y') = T(x',y') - 1/(w \cdot h) \cdot \sum_{x'',y''} T(x'',y'')$$
$$I'(x + x',y + y') = I(x + x',y + y') - 1/(w \cdot h) \cdot \sum_{x'',y''} I(x + x'',y + y'')$$

- method=CV_TM_CCOEFF_NORMED

$$R(x,y) = \frac{\sum_{x',y'} (T'(x',y') \cdot I'(x + x',y + y'))}{\sqrt{\sum_{x',y'} T'(x',y')^2 \cdot \sum_{x',y'} I'(x + x',y + y')^2}}$$

After the function finishes the comparison, the best matches can be found as global minimums (when `CV_TM_SQDIFF` was used) or maximums (when `CV_TM_CCORR` or `CV_TM_CCOEFF` was used) using the `minMaxLoc` function. In case of a color image, template summation in the numerator and each sum in the denominator is done over all of the channels and separate mean values are used for each channel. That is, the function can take a color template and a color image. The result will still be a single-channel image, which is easier to analyze.

V.3 Définition de la fonction `determine_fenetre`

Définition Python

```
def determine_fenetre(bb0, pt0, pt1):
    nPts = len(pt0)
    ofx = []
    ofy = []
# A COMPLETER (com1)
    for i in range(nPts):
        ofx.append(pt1[i][0]-pt0[i][0])
        ofy.append(pt1[i][1]-pt0[i][1])
# A COMPLETER (com2)
    dx = mediane(ofx)
    dy = mediane(ofy)
# A COMPLETER (com3)
    dist0=[]
    for i in range(nPts):
        for j in range(i+1,nPts):
            temp0 = ((pt0[i][0] - pt0[j][0])**2 + (pt0[i][1] - pt0[j][1])**2)**0.5
            temp1 = ((pt1[i][0] - pt1[j][0])**2 + (pt1[i][1] - pt1[j][1])**2)**0.5
            dist0.append(temp1/temp0)
    shift = mediane(dist0)
# A COMPLETER (com4)
    s0 = 0.5 * (shift - 1) * bb0[2]
    s1 = 0.5 * (shift - 1) * bb0[3]
# A COMPLETER (com5)
    x1 = bb0[0] - s0 + dx
    y1 = bb0[1] - s1 + dy
    x2 = bb0[2] + s0 + dx
    y2 = bb0[3] + s1 + dy
    w = x2-x1
    h = y2-y1
# A COMPLETER (com6)
    bb1 = [int(x1),int(y1),int(w),int(h)]
    return [bb1, shift]
```

Définition Scilab

```
function [bb1, shift]=determine_fenetre(bb0, pt0, pt1)
    nPts = length(pt0)
    ofx = zeros(nPts)
    ofy = zeros(nPts)
// A COMPLETER (com1)
    for i = [1:nPts]
        ofx(i)=pt1(i,1)-pt0(i,1)
        ofy(i)=pt1(i,2)-pt0(i,2)
    end
// A COMPLETER (com2)
    dx = mediane(ofx)
    dy = mediane(ofy)
    dist0=zeros(nPts)
// A COMPLETER (com3)
    for i = [1:nPts]
        for j = [i+1,nPts]
            temp0 = ((pt0(i,1) - pt0(j,1))**2 + (pt0(i,2) - pt0(j,2))^2)^0.5
            temp1 = ((pt1(i,1) - pt1(j,1))**2 + (pt1(i,2) - pt1(j,2))^2)^0.5
            dist0(i)=temp1/temp0
        end
    end
    shift = mediane(dist0)
// A COMPLETER (com4)
    s0 = 0.5 * (shift - 1) * bb0(3)
    s1 = 0.5 * (shift - 1) * bb0(4)
// A COMPLETER (com5)
    x1 = bb0(1) - s0 + dx
    y1 = bb0(2) - s1 + dy
    x2 = bb0(3) + s0 + dx
    y2 = bb0(4) + s1 + dy
    w = x2-x1
    h = y2-y1
// A COMPLETER (com6)
    bb1 = [int(x1),int(y1),int(w),int(h)]
endfunction
```