

Proposition de corrigé du sujet de concours

CCP

PSI Informatique

de l'année 2015

Ceci est une proposition de corrigé des concours de CPGE, réalisée bénévolement par des enseignants de Sciences Industrielles pour l'Ingénieur et d'Informatique, membres de l'[UPSTI](http://www.upsti.fr) (Union des Professeurs de Sciences et Techniques Industrielles), et publiée sur le site de l'UPSTI : www.upsti.fr

Ce document vous apportera des éléments de corrections pour le sujet traité, mais n'est pas un corrigé officiel du concours, ni un corrigé exhaustif très détaillé relatif au problème posé.

L'UPSTI ne répondra pas directement aux questions que peuvent soulever ces corrigés : nous vous invitons à voir avec vos enseignants si vous souhaitez des compléments d'information, et à passer par eux pour nous faire remonter des éventuels problèmes.

Ce document est sous une licence CreativeCommons BY-NC-ND.

Vous êtes autorisé à : Partager — copier, distribuer et communiquer le matériel par tous moyens et sous tous formats.

Selon les conditions suivantes :

Attribution — Vous devez créditer l'Œuvre, intégrer un lien vers la licence et indiquer si des modifications ont été effectuées à l'œuvre. Vous devez indiquer ces informations par tous les moyens raisonnables, sans toutefois suggérer que l'Offrant vous soutient ou soutient la façon dont vous avez utilisé son œuvre.

Pas d'Utilisation Commerciale — Vous n'êtes pas autorisé à faire un usage commercial de cette œuvre, tout ou partie du matériel la composant.

Pas de modifications — Dans le cas où vous effectuez un remix, que vous transformez, ou créez à partir du matériel composant l'œuvre originale, vous n'êtes pas autorisé à distribuer ou mettre à disposition l'œuvre modifiée.

Pas de restrictions complémentaires — Vous n'êtes pas autorisé à appliquer des conditions légales ou des mesures techniques qui restreindraient légalement autrui à utiliser l'œuvre dans les conditions décrites par la licence.

L'équipe UPSTI

Les fonctions et portions de scripts utilisent la syntaxe de Python 3.4.

Q1. Fonction `demande_fenetre()` :

```
def demande_fenetre():
    L=[]#Initialisation de la liste des valeurs à renvoyer
    L.append(demande_valeur("Coordonnée en x du coin de la fenêtre"))
    L.append(demande_valeur("Coordonnée en y du coin de la fenêtre"))
    L.append(demande_valeur("Largeur de la fenêtre"))
    L.append(demande_valeur("Hauteur de la fenêtre"))
    return L
```

Q2. Fonction `verification_fenetre()`:

```
def verification_fenetre(image_width,image_height,fenetre):
    if fenetre[0]<0 or fenetre[1]<0:#L'origine doit être dans l'image
        return False
    if fenetre[0]+fenetre[2]>image_width-1 or fenetre[1]+fenetre[3]>image_height-1:#La
fenetre ne doit pas déborder
        return False
    return True
```

On peut bien entendu utiliser la structure `if, elif, else`. L'appel du mot réservé `return` permet de s'en passer.

Q3. Requete SQL:

La description des tables est confuse. On ne voit pas apparaître la relation 1 à plusieurs. Il y a une faute d'orthographe à « purchase ».

```
SELECT filename FROM DEFINITIONS JOIN INSTRUMENTS ON DEFINITIONS.mid=INSTRUMENTS.id
WHERE INSTRUMENTS.name="pince"
```

Q4. Fonction `centre(fenetre)`:

```
def centre(fenetre):
    centre_x=fenetre[0]+fenetre[2]//2#Position en x (nombre entier)
    centre_y=fenetre[1]+fenetre[3]//2#Position en y (nombre entier)
    return centre_x,centre_y
```

Q5. Quantité de mémoire:

La question est, à mon sens, mal formulée.

Chaque pixel est codé sur trois octets (un par valeur RGB, de 0 à 255), soit une quantité de mémoire :

$$Q = 3 \times n \times m = 1440000 \text{ octets}$$

Q6. Fonction `grayscale(imagecolor)` :

On note qu'on aurait pu prendre la moyenne des trois valeurs RGB.

```
def grayscale(imagecolor):
    (p,m,n)=imagecolor.shape#Dimensions du tableau
    imagegray=zeros((m,n))#On initialise l'image en grayscale, noire pour l'instant
    for i in range(m):
        for j in range(n):
            imagegray[i,j]=(max(imagecolor[:,i,j])+min(imagecolor[:,i,j]))//2
    return imagegray
```

Q7. Fonction `construction_coordonnees_pts(fenetre,numM,numN)` :

On suppose ici qu'il faut évidemment tenir compte du fait que les coordonnées des points considérés sont nécessairement des entiers. Une fonction de type `arrange` ne fonctionnera pas correctement dans la plupart des cas (on « ratera » les bords opposés à l'origine).

```
def construction_coordonnees_pts(fenetre,numM,numN):
    pts=[]#On initialise la liste des points
    for i in range(numM):
        Pix=fenetre[0]+(fenetre[2]*i)//(numM-1)#Coordonnée en x
        for j in range(numN):
            Piy=fenetre[1]+(fenetre[3]*j)//(numN-1)#Coordonnée en y
            pts.append(Pix)
            pts.append(Piy)
    return pts
```

Q8. Termes de la méthode de Lucas-Kanade :

I_x représente le gradient de niveau de gris de l'image $I(t)$ suivant l'axe x .

I_y représente le gradient de niveau de gris de l'image $I(t)$ suivant l'axe y .

En utilisant l'approximation du point milieu, on peut écrire, si on n'est pas au bord de l'image :

$$I_x = \frac{I(t)[ux+\delta x,uy]-I(t)[ux-\delta x,uy]}{2 \times \delta x} \text{ et } I_y = \frac{I(t)[ux,uy+\delta y]-I(t)[ux,uy-\delta y]}{2 \times \delta y}$$

Avec $\delta x = \delta y = 1$ pixel, soit :

```
Ix=(imgI[ux+1,uy]-imgI[ux-1,uy])/2
Iy=(imgI[ux,uy+1]-imgI[ux,uy-1])/2
```

I_t représente la variation de niveau de gris pour un point donné entre les instants t et $t + dt$, soit, directement :

```
It=(imgJ[ux,uy]-imgI[ux,uy])/dt
```

Q9. Fonction creation_patch():

```
def creation_patch(P,patch_size):
    [Px,Py]=P
    patch=[] # On initialise le patch
    for i in range(Px-patch_size//2,Px+patch_size//2+1):
        for j in range(Py-patch_size//2,Py+patch_size//2+1):
            patch.append(i) #On ajoute la coordonnée en x
            patch.append(j) #et en y
    return patch
```

Q10. Fonction Calc_Ab()

```
def calc_Ab(imgI,imgJ,patch):
    A=zeros((patch_size**2,2)) #Initialisation de A, on considère que path_size est
    une variable globale
    b=zeros((patch_size**2,1)) #Initialisation de b
    for i in range(len(patch)**2-2,2):
        [Ix,Iy,Itdt]=calc_LK_terms([patch[i],patch[i+1]],imgI,imgJ) #On calcule
        les termes de l'équation de stationnarité
        A[i//2,0]=Ix
        A[i//2,1]=Iy
        b[i//2]=-Itdt
    return A,b
```

Q11. Résolution

On note ici que l'utilisation du type matrix de numpy aurait été bien plus pratique ! On utilise ici la méthode des déterminants pour obtenir les valeurs de dx et dy.

Soit le système : $\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} e \\ f \end{bmatrix}$. On a alors nécessairement :

$$x = -\frac{\begin{vmatrix} b & e \\ d & f \end{vmatrix}}{\begin{vmatrix} a & b \\ c & d \end{vmatrix}} \quad \text{et} \quad y = \frac{\begin{vmatrix} a & e \\ c & f \end{vmatrix}}{\begin{vmatrix} a & b \\ c & d \end{vmatrix}}$$

```
def resoud_LK(A,b):
    #On forme d'abord le problème
    (nlines,ncolones)=A.shape
    A_T=zeros((ncolones,nlines)) #On initialise la transposée
    for i in range(nlines):
        for j in range(ncolones):
            A_T[j,i]=A[i,j] #Transposition
    A_T_A=A_T*A #Matrice 2x2
    A_T_b=A_T*b #Vecteur 2x1
    #On résoud, avec un résultat en valeur entière (normalement égal à +-1)
    dx=int(-(A_T_A[0,1]*A_T_b[1,0]-A_T_A[1,1]*A_T_b[0,0])/(A_T_A[0,0]*A_T_A[1,1]-A_T_A[0,1]*A_T_A[1,0]))
    dy=int((A_T_A[0,0]*A_T_b[1,0]-A_T_A[1,0]*A_T_b[0,0])/(A_T_A[0,0]*A_T_A[1,1]-A_T_A[0,1]*A_T_A[1,0]))
    return dx,dy
```

Q12. Fonction recherche_points() :

```
def recherche_points(imgI, imgJ, pts):
    fpts=[] #On initialise la liste des points déplacés
    for i in range(len(pts)-2, 2): #Pour chaque point de pts
        patch=creation_patch([pts[i],pts[i+1]],path_size)
        (A,b)=calc_Ab(imgI, imgJ, patch)
        (dx, dy)=resoud_LK(A, b) #D'après ce qu'on a écrit, ce sont des entiers
        fpts.append(pts[i]+dx) #Coordonnée en x, entière
        fpts.append(pts[i+1]+dy) #Coordonnée en y, entière
    return fpts
```

Q13. Limites de l'algorithme :

On a supposé que le déplacement d'un point était au maximum d'un pixel dans chaque direction. Si le taux de rafraîchissement des images est trop faible ou la vitesse de déplacement trop élevée, on peut supposer qu'on aura une estimation erronée du déplacement des points.

Les cas pour lesquels on n'obtient pas de solution sont les cas pour lesquels $A^T A$ est non inversible. Cela peut par exemple arriver quand :

- Les I_x et I_y sont tous nuls : image uniforme (un seul niveau de gris).
- Tous les I_x sont nuls : variation de niveau de gris suivant y , voire rayures.
- Tous les I_y sont nuls : variation de niveau de gris suivant x , voir rayures.
- Les I_x et I_y sont tous égaux : dégradé de gris.

Q14. Script du statut :

```
statut=[] #Initialisation de la liste des statuts
for i in range(len(pts)-2, 2): #On suppose que pts a déjà été défini
    P1=[pts[i],pts[i+1]]
    P1n=recherche_points(imI, imJ, P1)
    P1nn=recherche_points(imI, imJ, P1n)
    if P1==P1nn: #Python compare les listes élément par élément
        statut.append(True)
    else:
        statut.append(False)
```

Q15. Fonction calcul1

La syntaxe donnée dans le sujet ne fonctionne pas avec le type liste, mais avec le type array de numpy. On construit un tableau des distances séparant les points correspondants des listes points1 et points2.

La dimension du résultat renvoyé par la fonction est celle du motif fourni en argument.

Q16. Fonction médiane :

On propose ici d'utiliser l'algorithme de tri fusion, puis de prendre la valeur milieu de la liste triée obtenue, qui est la médiane de la liste a, de longueur impaire.

```
def fusionr(L1,L2):#Fusion de deux listes triées, méthode récursive
    if len(L1)==0:
        return L2
    if len(L2)==0:
        return L1
    if L1[0]>L2[0]:
        return [L2[0]]+fusionr(L1,L2[1:])
    else:
        return [L1[0]]+fusionr(L1[1:],L2)

def trifusionr(L):#Tri fusion
    if len(L)==1:
        return L
    else:
        return fusionr(trifusionr(L[:len(L)//2]),trifusionr(L[len(L)//2:]))

def mediane(a):
    return trifusionr(a)[len(a)//2+1]#La liste est de longueur impaire
```

Le tri par insertion et le quicksort peuvent aussi être utilisés.

Q17. Complexité

Si on parle de complexité en temps, son avantage est d'être de complexité constante en $\mathcal{O}(n \times \log(n))$.

Q18. Vérification

```
def verification(pts,fpts,statut):
    nouveaux_pts=[]
    distances=calcul1(pts,fpts)#Liste des distances de déplacement
    med=mediane(distances)#Calcul de la médiane des distances de déplacement
    for i in range(len(statut)):
        if statut[i] and distances[i]<=med:#test
            nouveaux_pts+=fpts[2*i],fpts[2*i+1]]#Concaténation
    return nouveaux_pts
```

Q19. Corrélation croisée

La documentation de la fonction indique que celle-ci renvoie une image de la taille de l'image fournie en argument et dont chaque point contient la valeur du coefficient de corrélation entre l'image et la modèle (template). L'image en niveau de gris qu'on utilise correspond bien au codage sur 1 octet imposé par la fonction.

On doit donc, pour chaque point de la fenêtre d'étude (imgI) :

- Construire le patch correspondant.
- Calculer l'image de corrélation dans l'image imgJ, avec pour modèle le patch créé précédemment.

```
(m,n)=imgI.shape
for i in range(m):
    for j in range(n):
        patch=creation_patch([i,j],patch_size)
        imcorr=cv2.matchTemplate(imgJ,patch,CV_TM_CCORR_NORMED)
```

Le sujet n'indique pas ce qu'on fait de chacune des images de corrélation. La documentation semble indiquer qu'on va chercher la position de la valeur maximale, avec la méthode utilisée, pour situer le centre du patch dans la nouvelle image.

Q20. Variables et modification

Les arguments pt0 et pt1 utilisés dans la fonction sont de type tableau ou liste de liste. Les coordonnées y sont stockées en ligne de deux coordonnées (en x et y). Pour que cela fonctionne dans le script principal, il faut :

- Soit redéfinir les boucles de com1 et com3 pour lire une liste de coordonnées stockées les unes derrière les autres.
- Soit convertir pt0 et pt1 de liste vers tableau : `pts1=pts1.reshape((len(pts1)//2,2))` ou `pts1=array(pts1).reshape((len(pts1)//2,2))`.

Q21. Commentaires

- Com1 : Construction des listes respectives de déplacement des points en x et en y entre image précédente et image courante.
- Com2 : Calcul des médianes de ces déplacements, considéré comme le déplacement du motif avant homothétie.
- Com3 : Détermination des distances de chaque point de pt0 aux autres et respectivement de chaque point de pt1 aux autres puis du rapport de ces distances qu'on ajoute à une liste pour en extraire la médiane, considérée comme le facteur d'homothétie.
- Com4 : Détermination du demi-allongement de la nouvelle fenêtre, dans les directions x et y.
- Com5 : Détermination des coordonnées des coins opposés de la nouvelle fenêtre, puis de sa largeur et de sa hauteur.
- Com6 : Construction de la liste bb1 qui représente la nouvelle image et envoi des résultats (image et facteur d'homothétie).