DST Informatique : Étude d'un système autofocus d'appareil photo numérique

Nous nous intéressons dans ce sujet à l'étude de différents méthodes permettant de réaliser l'auto-focus sur les appareils photos numériques. L'auto-focus consiste à régler de manière automatique la netteté de l'image avant d'effectuer la prise de vue.
On suppose que tous les modules nécessaires ont été importés. Par exemple, toutes les fonctions de `numpy` ou celles de `matplotlib.pyplot`. Les fonctions importées peuvent être utilisées sans préfixe (`plot`, `arange`...). Pour réaliser l'autofocus, il existe deux méthodes principales :

— la mesure du contraste,
— la mesure de détection de phase.

Partie I.1 - Mesure du contraste

Le principe de cette méthode repose sur le fait qu'une image bien mise au point présente un maximum de contraste. Plus précisément, l'écart de valeur entre les pixels est alors maximal.
En fonctionnement, l'appareil mesure le contraste, puis par tâtonnement, va déplacer l'objectif jusqu'à détecter le maximum.

Représentation d'une image Une image est représentée sous la forme d'une matrice. Chaque élément de la matrice correspond à un pixel qui est généralement représenté par une liste de 3 entiers naturels, représentant les composantes rouge, vert et bleu. Chaque composante est représentée par un entier allant de 0 à 255 ; c'est le codage RVB.
On suppose, dans cette partie, travailler avec un appareil photo doté d'un capteur de 48 MPixels, c'est-à-dire que la photo sera composée de 48 millions de pixels.
On suppose que le stockage des photos se fait en mode RAW sans compression, c'est-à-dire que l'on stocke directement ce que récupère le capteur.

Question 1. Préciser l'espace mémoire nécessaire pour stocker la valeur d'une composante, puis celle d'un pixel et enfin celle d'une image en Mo (= 1000 ko).

L'image est représentée dans la suite par une variable `I`, de type liste de liste de liste Python classique ou de type `numpy.ndarray`. Dans les deux cas, on accédera aux données RVB d'un pixel de coordonnées (i, j), par l'instruction `I[i][j]`. Le candidat choisira librement le type qu'il préfère manipuler.

`I[i][j]` est désormais une liste contenant 3 valeurs de type `float`, obtenues en prenant les valeurs entières (de 0 à 255) du codage RVB que l'on divise par 255.

Conversion en niveaux de gris La première étape de la détection du contraste est de convertir l'image `I` en niveaux de gris pour n'obtenir qu'une valeur par pixel. Cette transformation s'opère en plusieurs étapes :

— les composantes R, V et B, qui ne sont pas linéaires en terme d'intensité lumineuse sur le rendu, sont d'abord linéarisées. Chaque composante C est transformée selon la définition suivante :
  • $C_{\text{lin}} = \frac{C}{12{,}92}$ si $C \leq 0{,}04045$;
  • $C_{\text{lin}} = \left(\frac{C+0{,}055}{1{,}055}\right)^{2{,}4}$ sinon.
— la valeur du niveau de gris Y_{lin} en échelle linéaire est calculée à partir des valeurs linéarisées par : $Y_{\text{lin}} = 0{,}2126R_{\text{lin}} + 0{,}7152V_{\text{lin}} + 0{,}0722B_{\text{lin}}$
— on repasse dans l'espace non linéaire avec le calcul suivant :
  • $Y = 12{,}92Y_{\text{lin}}$ si $Y_{\text{lin}} \leq 0{,}0031308$;
  • $Y = 1{,}055Y_{\text{lin}}^{\frac{1}{2{,}4}} - 0{,}055$ sinon.

Question 2. Écrire une fonction `Clinear(val)`, qui prend en argument une valeur de l'espace non linéaire et qui renvoie la valeur linéarisée.

Question 3. Écrire une fonction `Y(pix)` qui prend en argument une liste de trois valeurs correspondant à un pixel au format RVB et qui renvoie la valeur Y du niveau de gris dans l'espace non linéaire.

Question 4. Écrire une fonction `NiveauxGris(I)` prenant en argument une image `I` au format RVB et qui renvoie une image de même taille en niveau de gris.

Pour détecter le contraste d'une image en niveaux de gris, on va comparer pour chaque pixel les valeurs autour de celui-ci. Plus l'écart est grand (ce qui est le cas quand l'image est nette), plus les pixels autour du pixel de référence ont une valeur différente ; on calcule d'une certaine manière la dérivée en chaque pixel.
Cette opération est réalisée par un filtre de Sobel. Partant d'une image I, on extrait les pixels autour du pixel (i, j) sous la forme d'une matrice 3×3, notée $I_e = \begin{pmatrix} I_{i-1,j-1} & I_{i-1,j} & I_{i-1,j+1} \\ I_{i,j-1} & I_{i,j} & I_{i,j+1} \\ I_{i+1,j-1} & I_{i+1,j} & I_{i+1,j+1} \end{pmatrix}$.
On réalise ensuite une convolution entre cette matrice et la matrice de filtration. On définit la convolution entre deux matrices A et B de taille 3x3 par $A \otimes B = \sum_{i=0}^{2} \sum_{j=0}^{2} A_{ij} B_{ij}$
On réalise une filtration selon les deux directions de l'image. On donne les deux matrices de filtration :
$G_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$ et $G_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$
Enfin on calcule la norme des deux convolutions dont on prend la partie entière pour obtenir la valeur du contraste c du pixel (i, j) : $c = \text{Ent}\Big(\sqrt{(I_e \otimes G_x)^2 + (I_e \otimes G_y)^2}\Big)$.
Pour terminer, on calcule le contraste de cette manière pour chaque pixel intérieur à l'image (on ne calcule rien sur les bords pour simplifier), puis on réalise la moyenne des valeurs pour obtenir l'indice de contraste de référence c_{ref}.

Question 5. Écrire une fonction `convolution(A,B)` prenant en argument deux matrices de taille 3×3 et qui renvoie la valeur du produit de convolution.

Question 6. Écrire une fonction `contraste_pixel(I,i,j)` prenant en argument une image `I` au format niveaux de gris et les coordonnées du pixel (i, j) qui renvoie la valeur du contraste défini précédemment par la quantité c.

Question 7. Écrire une fonction `contraste(I)` prenant en argument une image `I` au format niveaux de gris et qui renvoie la valeur du contraste de référence c_{ref}.

Pour régler la netteté de l'image, l'objectif est déplacé à l'aide d'un moteur pas à pas. L'objectif est déplacé d'un pas, une photo est prise, la valeur du contraste de référence est calculée puis comparée à la valeur obtenue au pas précédent ; l'algorithme s'arrête dès que la valeur du contraste de référence diminue. Le moteur pas à pas recule d'un pas pour retourner à la position précédente où le contraste était maximal.
Pour cela, on utilise une fonction `position_objectif(val)` qui prend en argument un entier `val` allant de 0 à 1000 correspondant à la position en pas demandée et qui déplace l'objectif à cette position. Une fonction `prise()` permet de prendre un cliché et de retourner une image au format RVB.

Question 8. Écrire une fonction `reglage`, dont les arguments et les valeurs de retour sont à définir, répondant au comportement décrit en partant de la position 0 en pas. On supposera que le maximum de contraste existe.

Partie I.2 - Détection de phase

La méthode par détection de phase consiste à mesurer une petite partie de l'image avec deux capteurs différents. Quand les deux mesures sont identiques, l'image est nette. En calculant la différence de phase entre les deux mesures, il est possible de calculer directement la valeur dont doit se déplacer l'objectif.
On suppose que l'on dispose de deux capteurs de longueur 100 pixels (dont les valeurs seront stockées dans des listes) et que l'on a exactement la même séquence de valeurs mais décalées de quelques pixels. La différence de phase est le nombre de pixels (donc le nombre d'indices de décalage entre les deux listes) permettant de retrouver les mêmes séquences de pixels.
L'objectif est donc de comparer deux listes de valeurs, de même dimension, extraites des listes de données des deux capteurs. Trois cas peuvent se poser en fonction de la valeur du décalage noté `dec` et sont illustrés sur la figure 1. Les sous-listes extraites sont grisées.

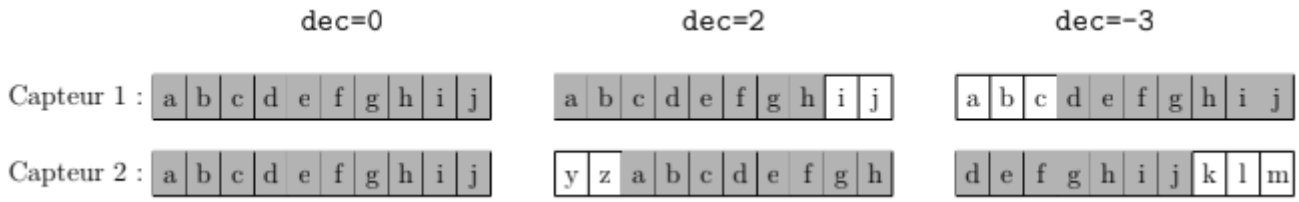

FIGURE 1 – Illustration du décalage (sur l'alphabet)

Question 9. Écrire une fonction `extraction(L1,L2,dec)` prenant en argument deux listes `L1` et `L2` ainsi qu'une valeur entière de décalage et qui renvoie deux sous-listes à comparer de longueur `len(L1)-dec`, conformément aux régles présentées ci-dessus. On supposera que les deux listes `L1` et `L2` sont de même taille.

Question 10. Écrire une fonction `comparaison(L1,L2)` (n'utilisant pas la comparaison interne de Python entre les listes) prenant en argument deux listes `L1` et `L2` qui renvoie `True` si les listes sont identiques et `False` sinon.

On suppose que le décalage est compris entre -80 et 80.
Question 11. Écrire une fonction `recherche_decalage(L1,L2)` prenant en argument deux listes `L1` et `L2` et qui renvoie la valeur du décalage ou `None` s'il n'existe pas.

Question 12. Évaluer la complexité de la fonction `recherche_decalage(L1,L2)` en prenant en compte le nombre de comparaisons en fonction de n la taille des listes et de m le nombre de décalages maximal à prendre en compte (161 dans notre exemple) dans le meilleur et dans le pire des cas.

En pratique, il n'y a pas de raisons que les deux capteurs CCD mesurent exactement les mêmes valeurs décalées. Les valeurs mesurées décalées seront approximativement les mêmes. Pour résoudre ce problème, on va calculer une erreur quadratique moyenne entre les deux sous-listes et puis chercher le minimum en fonction du décalage.
Pour deux sous-listes $L1$ et $L2$ de taille n, l'erreur sera définie par :

$$\text{erreur} = \frac{1}{n}\sum_{i=0}^{n-1}(L1_i - L2_i)^2$$

Question 13. Écrire une fonction `erreur(L1,L2)` qui retourne l'erreur quadratique définie ci-dessus.

Question 14. Écrire une fonction `recherche_decalage_2(L1,L2)` qui cherche le minimum de l'erreur pour des décalages de -80 à 80 et qui retourne la valeur de ce décalage. Évaluer la complexité de cette fonction.

Partie I.3 - Comparaison des deux méthodes

Deux méthodes de réglage de l'objectif ont été introduites dans les parties précédentes.

Question 15. Décrire en 5 lignes maximum les avantages et les inconvénients de ces deux méthodes et laquelle vous semble être la plus pertinente.

Partie I.4 - Commande du moteur pas à pas

L'objectif de mise au point est motorisé par un moteur pas à pas bipolaire commandé en « demi-pas ». Les paragraphes suivants permettent de décrire le principe de fonctionnement et la commande du moteur mais leur compréhension détaillée n'est pas utile pour répondre à la suite des questions.
Le principe du moteur pas à pas consiste à positionner le rotor sur lequel se trouvent des aimants permanents polarisés nord ou sud régulièrement espacés. Sur le stator se trouvent deux demi-bobines. La commande des demi-bobines se fait de telle sorte que le rotor va s'aligner sur le stator par pôle opposé (un nord en face d'un sud et un sud en face d'un nord).
Prenons un exemple simplifié de la figure 2 d'un moteur avec deux bobines (AB pour la première et CD pour la seconde). Les bobines sont en deux parties situées de part et d'autres du rotor. Sur le rotor se trouve un aimant permanent avec deux pôles.

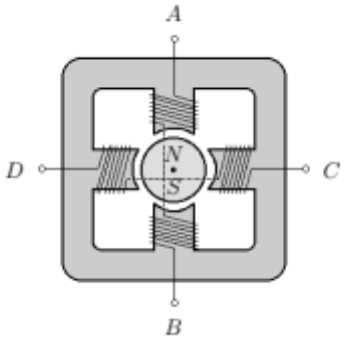

FIGURE 2 – Structure élémentaire d'un moteur pas à pas

Les deux demi-bobines étant positionnées à 90°, la commande en demi pas consiste à faire des :

- pas entier : les bobines sont alimentées l'une après l'autre
- demi-pas : les deux bobines sont alimentées et le rotor s'aligne entre deux bobines

La figure 3 montre les différentes positions associées à la séquence d'alimentation présentée plus loin.

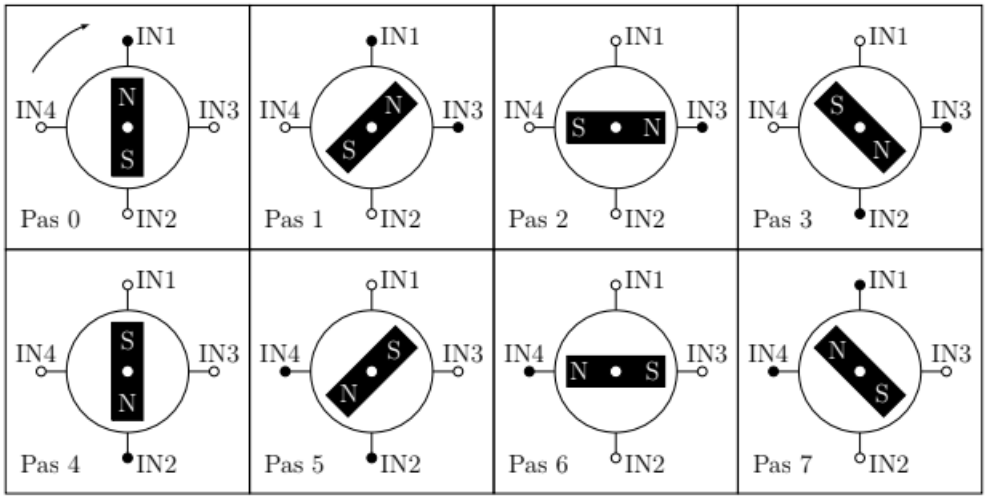

FIGURE 3 – Séquence d'alimentation des bobines : —● IN à 1 ; —○ IN à 0

La séquence d'alimentation du préactionneur des bobines (non détaillé ici) est donné par le chronogramme de la figure 4 et donne les 3 cas d'alimentation possibles pour obtenir la rotation dans le sens positif. Pour obtenir la rotation dans le sens négatif, il suffit de parcourir la séquence d'alimentation dans l'autre sens.

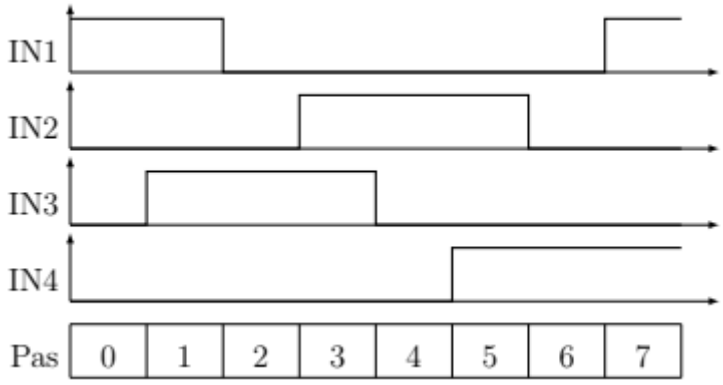

FIGURE 4 – Chronogramme d'alimentation des 4 entrées

On se propose pour la suite de programmer le déplacement pas à pas de l'objectif. Pour cela, on supposera que les variables `IN1`, `IN2`, `IN3` et `IN4` ont été déclarées de manière globale et représentent les sorties à piloter sur un microcontrôleur. Pour piloter ces sorties, on utilisera la fonction `modif_sortie(IN, valeur)` où `IN` sera la sortie à modifier et `valeur` la valeur de type booléen souhaitée pour la sortie.
Exemple : `modif_sortie(IN4, True)` passera la sortie `IN4` à 1 et `modif_sortie(IN4, False)` la passera à 0.
On remarquera que la séquence d'alimentation présente les 8 premiers pas, les autres pas étant obtenus à un modulo près.

Question 16. Écrire une fonction `faire_un_pas_positif(pas_actuel)` qui prend en argument la valeur du pas courant qui modifie l'état des sorties `IN1` à `IN4` et qui renvoie la nouvelle valeur du pas. On veillera à ne changer l'état que des sorties nécessaires.

On supposera l'existence d'une variable globale `pas_courant` comprise entre 0 et 1000 correspondant à la position courante du moteur pas à pas, ainsi qu'une fonction `faire_un_pas_negatif(pas_actuel)` qui permet de réaliser un pas négatif.

Question 17. Écrire une fonction `position_objectif(pas)` qui prend en argument la position en pas à atteindre à partir de la position courante réalisant le déplacement demandé pas à pas. Il peut y avoir plus d'un pas à effectuer. On vérifiera que le pas demandé est atteignable (compris entre 0 et 1000).

Partie II - Gestion des photographies

Afin d'afficher correctement les clichés, il faut réorienter l'image en fonction de l'orientation initiale du cliché. Les réorientations automatiques classiques à gérer sont le pivotement de 90 degrés dans le sens horaire ou le sens trigonométrique ainsi que la rotation de 180 degrés.

Question 18. Illustrer sur un schéma les nouvelles coordonnées d'un pixel de coordonnées (i, j) après une rotation de 180 degrés. Écrire une fonction `rotation_180(image)` qui prend en argument une image au format RVB et qui renvoie une nouvelle image pivotée de 180 degrés.

Question 19. Illustrer sur un schéma les nouvelles coordonnées d'un pixel de coordonnées (i, j) après une rotation de 90 degrés. Écrire une fonction `rotation_90(image)` qui prend en argument une image au format RVB et qui renvoie une nouvelle image pivotée de 90 degrés dans le sens trigonométrique.