

Détection de rangée dans un champ (d'après CCP SI 2016)

Un robot maraicher doit se déplacer au milieu d'une rangée de poireaux. Il dispose d'une caméra qui filme le champ, et nous allons voir comment il traite l'image obtenue pour contrôler et corriger sa position.

L'image initiale est une image bitmap RVB (rouge vert bleu). Elle est enregistrée sous forme d'un tableau, où chaque case correspond à un pixel, et contient trois entiers de 8 bits représentant les trois composantes de couleur du pixel.

La commande `imshow` de la bibliothèque `matplotlib.pyplot` permet d'afficher une telle image. Vous trouverez sur le site quelques images de champs de poireaux et un fichier `bib_poireaux.py` montrant comment les afficher.

Nous utiliserons dans ce sujet des tableaux numpy (la commande `imread` de `matplotlib`, utilisée dans `bib_poireaux.py`, permet de charger en mémoire une image depuis un fichier extérieur à Python, et elle renvoie un tableau numpy).

1 Passage en noir et blanc

1. Calculer la mémoire nécessaire pour enregistrer une image de format 700×393 . On exprimera le résultat en kilooctets (ko).
2. La première étape est de passer à une image en niveaux de gris. Une telle image sera représentée toujours par une matrice dont chaque case représente un pixel, mais cette fois une case contiendra un seul entier de 8 bits et non plus trois. On utilise la formule suivante : si R , V , et B sont les entiers qui représentent les composantes rouge, vert, et bleu d'un pixel, alors le niveau de gris correspondant est $\lfloor 0.299R + 0.587V + 0.114B \rfloor$. C'est donc un entier de $\llbracket 0, 255 \rrbracket$. La valeur 0 correspond au noir, et 255 au blanc.

Écrire une fonction prenant en entrée une image couleur et renvoyant l'image en niveaux de gris sur 8 bits correspondant.

Remarque : La fonction `imshow` pourra afficher également ce type d'images. Si on souhaite que la couleur affichée soit réellement du gris, il faut fixer la « colormap » de la manière suivante : `plt.imshow(image_grise, cmap = plt.get_cmap("gray"))`.

3. On passe ensuite à une image en noir et blanc. On choisit un seuil s , puis tout pixel dont le niveau de gris est $\geq s$ sera transformé en blanc (valeur 0), et tout pixel de niveau de gris $< s$ passera noir (valeur 1).
 - (a) Quelle valeur faut-il prendre pour s pour que la moitié des pixels deviennent noirs, et l'autre moitié blancs ?
 - (b) Écrire une fonction `seuil` prenant en entrée une image en niveaux de gris et renvoyant le seuil à prendre. Vous pouvez utiliser les programmes vus en cours. Par exemple utiliser `from cours import *` en début de fichier si votre fichier `cours.py` est présent dans le même répertoire que celui de ce TP.
 - (c) Écrire enfin une fonction `noir_et_blanc` pour convertir une image en niveaux de gris en une image en noir et blanc.
4. Donner la complexité de la conversion d'une image couleur vers une image noir et blanc en fonction des dimensions de l'image.

2 Positionnement dans le rang

Le principe de l'algorithme de traitement global pour chercher la position du robot dans le rang consiste à déterminer, à partir de l'image monochrome, deux droites qui correspondent à la rangée droite et la rangée gauche de culture. Ces deux droites sont obtenues en faisant l'hypothèse qu'elles possèdent le maximum de pixels noirs sur l'image monochrome. Une fois ces droites déterminées, une fonction décalage permet de déterminer le positionnement du robot dans l'allée.

Dans la suite, on se concentre sur la droite correspondant à la rangée de droite. Nous faisons les hypothèses suivantes, en notant (n, p) le format de l'image :

- Son point de départ est sur la première ligne de l'image. Son indice de colonne, noté j_0 , est supérieur à $\frac{p}{2} - \frac{p}{7}$;
 - Son point d'arrivée est sur la dernière ligne de l'image (indice $n - 1$). Son indice de colonne, noté j_1 est strictement supérieur à j_0 .
1. Exprimer en fonction de j_0 et j_1 l'équation de la droite passant par les points $(0, j_0)$ et $(n - 1, j_1)$.
 2. Écrire une fonction `nb_pixels_noirs` prenant en entrée une image noir et blanc et deux indices j_0 et j_1 et renvoyant le nombre de pixels noirs sur la droite passant par les points $(0, j_0)$ et $(n - 1, j_1)$.

On utilisera la méthode naïve suivante : pour tout $i \in \llbracket 0, n \rrbracket$, on calcule l'ordonnée j du point de la droite correspondante. Puis on arrondit j à l'entier le plus proche (utiliser `round`) et on teste si le pixel correspondant est noir.

3. En déduire une fonction `detection_rang_droit` renvoyant les valeurs j_0 et j_1 pour lesquelles le nombre de pixels noirs est maximal.
4. Estimer la complexité de cette fonction.
5. Expliquer le rôle de la fonction suivante (on suppose connue une fonction `detection_rang_gauche` analogue à `detection_rang_droit` mais pour le rang gauche) :

```
1 def f(IM):  
2     IMN=noir_et_blanc(IM)  
3     d=detection_rang_droit(IMN)[1]  
4     g=detection_rang_gauche(IMN)[1]  
5     return (d+g-IMN.shape[1])/2
```
