Devoir surveillé N°4 (1h15)

- Cette partie est un extrait de sujet de concours CCINP. Il n'y a que les 10 premières questions (sur un total d'environ 40). Le sujet original était prévu pour 3h. Par rapport à d'habitude, vous devez en plus faire l'effort de bien comprendre et vous approprier le contexte et les notions expliquées pour répondre à chaque question.
- **La présentation, la rédaction, la lisibilité des scripts entrent pour une part importante dans l'appréciation des copies.**
- **Important:** vous pouvez librement utiliser les fonctions issues des questions précédentes, même si vous ne les avez pas traitées.

Dans ce sujet, les fonctions sont définies avec leur signature :

```
ma_fonction(arg1:type1, arg2:type2) -> type3
```

Cette notation permet de définir une fonction qui se nomme `ma_fonction` qui prend deux arguments en entrée `arg1` de type `type1` et `arg2` de type `type2`. Cette fonction renvoie une valeur de type `type3`.
Il n'est pas nécessaire de recopier les signatures des fonctions dans le Document Réponse, il suffit d'écrire directement :

```python
def ma_fonction(arg1,arg2) :
    #liste d'instructions
```

Détection d'objets dans le cadre de la conduite autonome

L'entreprise Easymile a mis en place à Toulouse des bus 100 % autonomes sur le campus universitaire de Paul Sabatier. Il s'agit d'un projet de recherche mené par l'IRIT et l'Université, mais dès aujourd'hui, les minibus (figure 1) qui peuvent accueillir vingt personnes sont en fonction et se déplacent dans un rayon de 5 km . Ces véhicules autonomes nécessitent un niveau élevé d'informations pour fonctionner en toute sécurité et sont donc équipés d'une gamme complète de capteurs.

Ces capteurs collectent et analysent les données enregistrées pour créer une image à 360 degrés de l'environnement, y compris les infrastructures, les autres véhicules, les piétons et tout ce qui se trouve sur le chemin.

Le traitement en temps réel des données permet au système du véhicule autonome de décider comment se comporter pour progresser en toute sécurité sur la route (s'arrêter, partir, ralentir, etc.).

Ce sujet s'intéresse à une partie des algorithmes mis en place afin de rendre le projet possible, notamment la partie détection d'obstacles dans l'espace.

En effet, pour détecter les objets, les piétons et les véhicules avec précision et rapidité, il est nécessaire de mettre en place des programmes robustes avec une complexité limitée.

Dans un premier temps, nous nous intéresserons à un algorithme simple de détection d'obstacles, basé sur l'apprentissage supervisé. Il faudra tout d'abord réfléchir à la méthode de localisation dans l'espace du piéton. Dans un deuxième temps, nous nous pencherons sur le traitement de l'image et la détection des obstacles. Enfin, dans le but d'améliorer et de comprendre le fonctionnement des programmes du bus, la collecte d'un grand nombre de données est réalisée à chaque déplacement, c'est l'objet de la dernière partie du sujet.

Figure 1 - Bus Easymile

Partie I - Localisation dans l'espace

Un premier calcul important pour détecter les piétons et les obstacles est la prise en compte de la distance focale de la caméra de détection. Pour cela, un calcul simplifié est réalisé à partir d'une taille réelle standard. Les caméras permettent en effet de relever une matrice de pixels qui donne ainsi une dimension aux formes et aux distances.
Sur la figure 2, lorsque A' est confondu avec F', la relation entre la focale f et la distance réelle est la suivante :

$$D_{\text{réelle}} = \frac{\left\|\overrightarrow{OF'}\right\| \cdot \|\overrightarrow{AB}\|}{\left\|\overrightarrow{A'B'}\right\|} = f \cdot \frac{H_{\text{objet}}}{H_{\text{image}}} \quad (1)$$

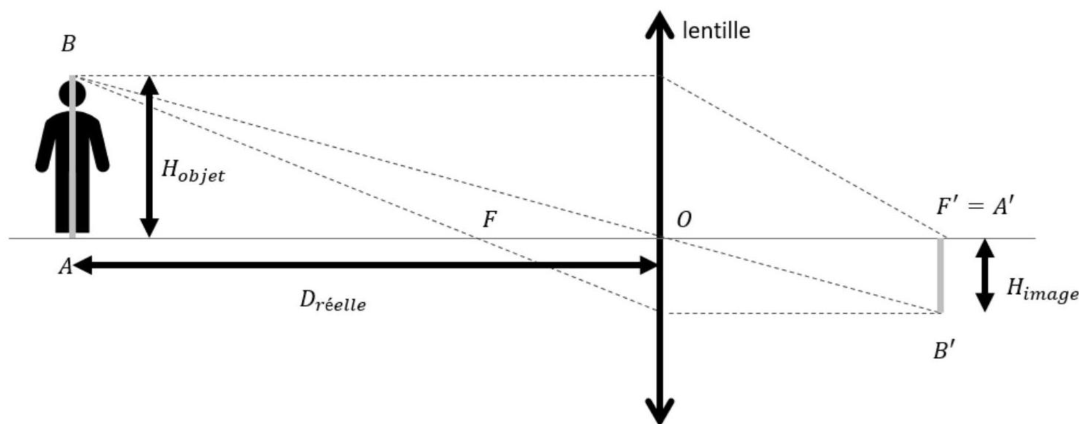


Figure 2 - Calcul simplifié de la focale **f**

Nous allons tout d'abord créer un dictionnaire avec 5 obstacles "classiques" qui doivent être détectés.

Q 1 Créer un dictionnaire nommé `dico`, possédant 5 clés de type `str` : `'adulte'`, `'enfant'`, `'animaux'`, `'véhicule'`, `'indéterminé'`. Chaque clé sera initialisée avec comme valeur une liste vide.

Pour chacune des clés du dictionnaire créé à la question Q 1, nous allons construire un intervalle de hauteur pouvant correspondre à l'obstacle et réparti à 20% de part et d'autre de la hauteur moyenne de celui-ci, sous la forme d'une liste de deux éléments :

- pour l'adulte, la hauteur moyenne est fixée à 175 cm , la liste associée à la clé sera alors `[175*0.8, 175*1.2]` ;
- pour l'enfant, la hauteur moyenne est fixée à 110 cm ;
- pour les animaux, la hauteur moyenne est fixée à 50 cm ;
- pour les véhicules, la hauteur moyenne est fixée à 200 cm ;
- pour les autres valeurs indéterminées, le couple associé à la clé sera composé des valeurs les plus basses `[0, 50*0.8]`.

Q 2 Écrire une suite d'instructions qui modifie votre dictionnaire initialisé à la question Q 1 en associant le couple d'intervalles à chacune des clés du dictionnaire.

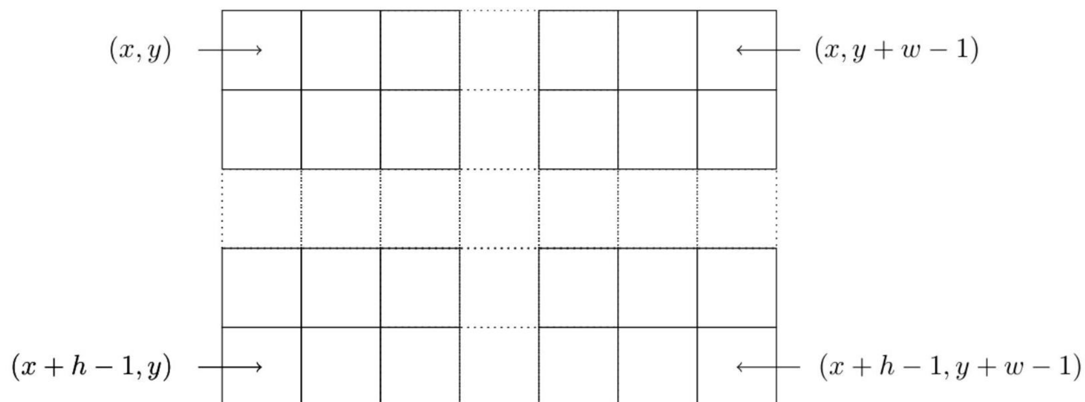
Q 3 Écrire une fonction `obstacles_rencontres(dico : dict, hauteur : float)->list[str]` qui permet de vérifier que pour une hauteur donnée, le nom de l'obstacle est bien renvoyé. Elle prend donc en entrée une hauteur réelle en cm et le dictionnaire créé aux questions précédentes, contenant les couples clé, intervalle de hauteur. Dans le cas d'au moins une possibilité, tous les noms seront renvoyés dans une liste de chaîne de caractères. Si par contre la hauteur n'est dans aucun intervalle, la liste vide sera renvoyée.

⟶ Par exemple, pour une hauteur de 170 cm , la fonction renvoie `['adulte', 'véhicule']`.

Les algorithmes de détection d'obstacles doivent traiter les images provenant des caméras. Les images issues de la caméra sont ici considérées comme un tableau à deux dimensions qui contient des pixels.

Nous allons ici travailler dans l'une de ces images de pixels pour comprendre la détection des obstacles. On dispose d'une liste `faces_detec` constituée de quadruplets `( x, y, h, w )` correspondant chacun à un contour détecté dans une image captée *(Un contour est donc ici un rectangle...)*. Ici `x, y` désignent les coordonnées du coin supérieur gauche dans l'image, `h` la hauteur du contour, `w` sa largeur.

Nota: il est important de rappeler que les images sont parcourues de haut en bas avec `x` et `h` travaillant sur les lignes, `y` et `w` sur les colonnes, `x, y, h` et `w` sont en nombre de pixels. Le dernier pixel de l'image se situe donc à la coordonnée `(x+h-1, y+w-1)`.



Q 4 Écrire une fonction `focale(faces_detec: list, Dr: float, Hr: float)->list[float]` prenant pour argument une liste `faces_detect` de contours, une distance `Dr` réelle et une hauteur `Hr` réelle données, et qui renvoie la liste contenant la valeur de la focale définie dans l'équation (1) pour chaque contour détecté. *(Pour info, la hauteur de l'image est aussi la hauteur du contour...)*

Q 5 Écrire une fonction `moy(L: list[float])->float` qui renvoie la moyenne des éléments de la liste `L`.

Partie II - Traitement de l'image

II. 1 - Calcul de luminance

Nous allons nous intéresser maintenant au format des images et plus particulièrement au passage en niveau de gris, c'est-à-dire le calcul de la luminance moyenne utile à la détection de contour. La teinte d'un pixel peut être représentée de plusieurs façons. Une méthode courante, basée sur la synthèse additive, consiste à la décomposer en trois composantes qui correspondent aux couleurs rouge, vert et bleu.

On parle de représentation RVB (Rouge, Vert et Bleu). Chacune des trois composantes donne l'intensité de la couleur correspondante dans la teinte finale sous forme d'un nombre entier compris entre 0 et 255. 0 indique l'absence de cette couleur et 255 l'intensité maximale. Ainsi, le triplet $(0, 0, 0)$ désigne un pixel noir et $(255, 255, 255)$ un pixel blanc.

Rappelons que les images sont représentées sous la forme d'une liste de lignes où chaque ligne est une liste de triplets RVB. Ainsi, on accède au pixel de coordonnées `(x, y)` de l'image `I` par l'expression `I[x][y]`.

Q 6 Sachant qu'une composante de couleur est représentée sur 8 bits, préciser l'espace mémoire nécessaire pour stocker un pixel, puis pour stocker une image de taille 8000×6000 pixels. La réponse pour l'image sera donnée en Mo ($1\text{Mo} =$ un million d'octets).

Pour représenter une image en niveau de gris, il suffit d'une valeur par pixel représentant l'intensité de gris entre le noir et le blanc, c'est la luminance. Pour convertir une image en couleurs en niveaux de gris, plusieurs méthodes sont possibles. Faire la simple moyenne des composantes R, V et B d'un pixel donne visuellement des résultats décevants.

On procède donc selon le protocole suivant :

- notons C une des composantes R, V ou B d'un pixel. On pose $C_1 = C/255$; elle sera transformée en une variable C_{lin} selon la définition suivante :
  - $C_{\text{lin}} = \dfrac{C_1}{12,92}$ si $C_1 \leqslant 0,04045$;
  - $C_{\text{lin}} = \left(\dfrac{C_1 + 0,055}{1.055}\right)^{2.4}$ sinon ;
- puis on calcule la valeur "linéaire" du niveau de gris Y_{lin} avec la formule :

$$Y_{lin} = 0,2126 \cdot R_{lin} + 0,7152 \cdot V_{lin} + 0,0722 \cdot B_{lin}$$

- enfin l'intensité Y du pixel de l'image en niveau de gris sera calculée comme suit :
  - $Y = \lfloor 255 * 12,92 * Y_{\text{lin}} \rfloor$ si $Y_{\text{lin}} \leqslant 0,0031308$;
  - $Y = \left\lfloor 255 * \left(1,055 * Y_{\text{lin}}^{1/2.4} - 0,055\right)\right\rfloor$ sinon.
    ($\lfloor x \rfloor$ désigne la partie entière de x)

Q 7 Écrire une fonction `Clinear(C: int)->float`, qui prend en argument une composante de couleur `C` d'un pixel et qui renvoie la valeur C_{lin} décrite plus haut.

Q 8 Écrire une fonction `Intensite(pix: tuple)->int` qui prend en argument un triplet de trois composantes correspondant à un pixel au format RVB et qui renvoie la valeur Y du niveau de gris correspondant.

Q 9 Écrire une fonction `init(h:int, w:int)->list` prenant en argument les 2 entiers `h` et `w` caractérisant la taille d'une image et qui renvoie une liste de `h` listes remplie de `w` zéros.
Par exemple, `init(2,3)` renvoie `[[0,0,0],[0,0,0]]`.

Q 10 Écrire une fonction `NiveauxGris(I:list)->list` prenant en argument une image `I` au format RVB et qui renvoie une image de même taille en niveau de gris.