Partiel - groupe 1

Exercice 1 – Architecture de la simulation

Dans cet exercice, certaines questions sont indépendantes, d'autres pas : vous devez lire l'ensemble des questions.
Soit l'interface de robot suivante :

```
class Robot2I013:
    WHEEL_BASE_WIDTH          = 117  # distance (mm) de la roue gauche a la roue droite.
    WHEEL_DIAMETER            = 66.5 #  diametre de la roue (mm)
    MOTOR_LEFT                = 1 # constante moteur gauche, a donner comme parametre port dans les methodes
    MOTOR_RIGHT               = 2 # constante moteur droit, a donner comme parametre port dans les methodes
                                  # pour passer les deux moteurs, additionner les deux constantes
    def __init__(self):
    def set_motor_dps(self, port, dps): # mettre en route le moteur "port" (ou les deux en additionnant les codes)
                                        # dps = degre par seconde (vitesse positive = en avant, negative = en arriere)
    def read_encoders(self): # lire les angles des deux moteurs
    def offset_motor_encoder(self, port, offset): # reinitialiser les angles des moteurs
    def distance(self): # recuperer la distance du capteur frontal (en cm)
    def servo_rotate(self,position): # indiquer un angle a la tete du robot
    def get_image(self): # recuperer l'image de la webcam
```

Nous proposons de distinguer les classes `Simulation`, `Robot2I013` et `Strategie`, de manière à pouvoir utiliser une méthode `run()` dans la classe `Simulation` de la manière suivante :

```
def run(self):
    """
        Boucle principale de la simulation. Elle appelle strategie.update() en boucle
        et s'arrete quand strategie.stop() rend vrai.
    """
    cpt = 0
    while not self.strategie.stop():
        self.strategie.update()
        cpt+=1
    print("Stoping ... ")
```

Q 1.1 La classe `Simulation` doit elle avoir un `robot` en attribut ? Dessiner rapidement les classes en jeu dans l'architecture et les attributs significatifs. En se plaçant dans le `main` (ou dans le script principal), dans quel ordre doit-on créer la `simulation`, la `strategie` et le `robot` ?

Q 1.2 Quelles méthodes doivent obligatoirement être présentes dans la classe `Strategie` ?

Q 1.3 Donner le code de la classe `StrategieToutDroit70` qui :

- initialise un compteur de distance à 0 une variable de contrôle `stop` à `False` & met en route les moteurs (à 120 dps),
- à chaque appel de la méthode `update` :
  - ▶ calcule la distance parcourue :
    (angle actuel - angle précédent) $\times\pi\times$ le rayon de la roue
  - ▶ vérifie si la distance totale est inférieure à 70cm et, dans la négative, passe la variable `stop` à `True`

Q 1.4 Quel problème pose la stratégie ci-dessus ? A quelle vitesse roule le robot ?

Q 1.5 Donner le code de la classe `StrategieToutDroit70Fine` (ou juste les modifications à effectuer par rapport au code ci-dessus) pour (1) avancer rapidement jusqu'à 70cm moins un tour de roue (2) avancer doucement jusqu'à la distance manquante.

Q 1.6 Donner le code du script permettant de tester cette stratégie.

Q 1.7 En partant du principe que pour faire un quart de tour sur place, il faut pivoter les deux roues d'un demi-tour en sens opposés –e.g. un demi tour positif pour la roue droite, un demi tour négatif pour la roue droite–, donner la stratégie permettant d'effectuer la tâche évoquée en cours : dessiner un carré de 70cm de coté.

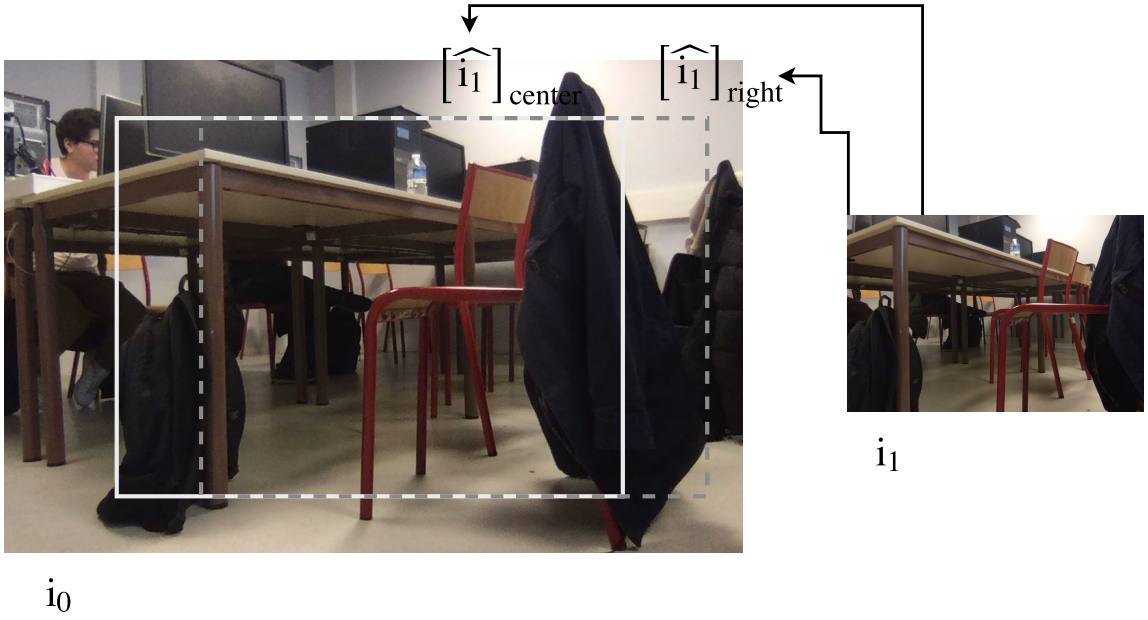
Q 1.8 Dans la pratique, comment évaluer l'efficacité de la stratégie (que le robot dessine bien le carré voulu) ?

Q 1.9 Quels méthodes de test unitaire devrait-on fournir pour tester le code ? Donner succinctement le code équivalent.

Exercice 2 – Calibration automatique

Nous voulons créer une stratégie de calibration, détectant les dérives (à gauche ou à droite) du robot lorsqu'il est censé avancé en ligne droite. L'idée de base est la suivante :

1. prendre une image i_0 à t_0
2. prendre une image i_1 à t_1, qui devrait être une sous-partie de i_0
3. calculer si i_1 est plus proche d'une sous-partie centrée ou d'une sous-partie décalée, comme le montre la figure ci-dessous.



Pour mettre en place le système de calibration :

- vous avez déjà accès à la méthode `get_image` du robot, qui renvoie une image PIL (la librairie de gestion la plus classique de python),

```
1 img = robot.get_image()
2 largeur,hauteur = img.size # acces aux coordonnees
```

- vous devez récupérer une partie de l'image courante (*crop*) :

```
3 img2 = img.crop((x, y, larg, haut)) # argument = quadruplet (coord_x, coord_y, largeur, hauteur)
```

- vous devez savoir mettre une image à l'échelle (pour avoir autant de pixels dans $\widehat{i_1}$ et dans i_1)

```
4 img2 = img.resize((new_larg, new_haut))
```

- enfin, vous devez comparer $\widehat{i_1}$ et i_1. Pour cela, vous devez accéder aux pixels de la manière suivante :

```
5 (rouge,vert,bleu) = img.getpixel((x,y))
```

  Nous considèrerons la distance entre images au sens des moindres carrés, c'est à dire la somme sur tous les pixels des écarts sur chaque composante : pour un couple de pixels (p^1, p^2) de de la première et de la deuxième image de couleur $p^1 = (r^1, b^1, g^1)$ et $p^2 = (r^2, b^2, g^2)$, la distance est $d(p^1, p^2) = (r^2 - r^1)^2 + (g^2 - g^2)^2 + (b^2 - b^1)^2)$; ainsi la distance entre les deux images de dimension $n \times m$ est : $\sum_{i=0}^{n} \sum_{i=0}^{m} d(p^1_{i,j}, p^2_{i,j})$ où $p^1_{i,j}$ est le pixel à la position (i,j) de la première image (et $p^2_{i,j}$ celui de la deuxième image).

Q 2.1 Donner le code de la méthode `dist_images` qui retourne la distance entre deux images au sens des moindres carrés.

Q 2.2 Etant donné des paramètres (existant en variables globales) `OFFSET_X, OFFSET_Y` qui indiquent la position du point haut gauche du rectangle délimitant la nouvelle image $\hat{i_1}$ dans l'image i_0 et `NEW_LARG, NEW_HAUT` sa largeur et sa hauteur, et un paramètre `OFFSET_DX` qui représente le décalage entre $\hat{i_1}$ centré et $\hat{i_1}$ droite (en pointillé), donner le code de la méthode `get_best_image` prenant en argument i_0 et i_1 et renvoyant -1 en cas de décalage à gauche, 0 si tout est OK et 1 en cas de décalage à droite.

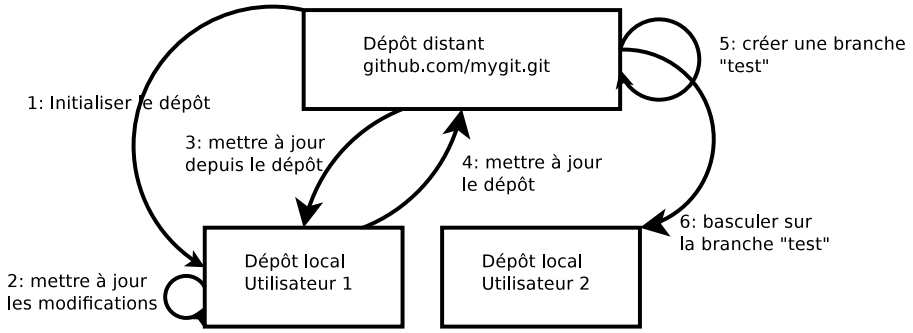
Q 2.3 Donner le code d'une stratégie pour avancer tout droit (toujours sur 70cm) avec un paramètre d'équilibrage pour régler les `dps` des deux moteurs. A l'issue de la ligne droite, la stratégie stockera le fait qu'elle ait dérivé à gauche, à droite ou pas du tout.

Q 2.4 En imaginant que nous disposons d'une grande salle, donner le code de la super stratégie permettant de régler automatiquement le paramètre d'équilibrage.

Exercice 3 (6 points) – Questions indépendantes

Q 3.1 Définir en quelques lignes ce qu'est `git`, ses avantages et inconvénients.

Q 3.2 Donner les commandes qui permettent d'exécuter les différentes opérations du diagramme suivant.



Q 3.3 Python

Q 3.3.1 Comment définit-on un constructeur en Python ?

Q 3.3.2 À quoi sert un fichier `__init__.py` dans un répertoire Python ?

Q 3.3.3 Quelle est la différence entre import relatif et import absolu ?

Q 3.3.4 Que veut dire sérialiser un objet ? Donner une suite d'instructions Python permettant de sérialiser un objet "simple" `obj` **sans** utiliser de module (`pickle` ou autre). Quelle(s) difficulté(s) peuvent apparaître pour la sérialisation d'un objet complexe ?

Q 3.4 Qu'est ce qu'un bus en informatique ? Donner deux exemples et leur fonctionnement.

Q 3.5 Résumer en quelques lignes la philosophie Agile/Scrum et son vocabulaire (objectifs, fonctionnement, éléments nécessaires à l'application).