Résumé de « An Introductory Robot Programming Tutorial », article écrit par McCrea :

McCrea est un développeur freelance qui a mis au point un petit simulateur pour son robot virtuel. Il explique dans l'article comment le robot réagit avec son environnement et y expose les difficultés qu'il a pu rencontré.

Le simulateur utilise beaucoup de classes et d'interfaces donc des connaissances en programmation orientée objet sont exigées.

Du fait qu'un robot interagit avec son environnement uniquement par le biais de ses capteurs, il faut prévoir les retours de ces capteurs et ainsi reposer toute la simulation sur un « modèle du monde réel ». Il existe donc très souvent beaucoup de divergences entre simulation et test grandeur nature car dès que le monde réel s'éloigne du modèle, les réactions du robot ne seront plus adaptées.

Le simulateur est en Python, et le robot est contrôlé par les classes suivantes:

- **supervisor.py**, contrôle les interactions entre le robot et le monde simulé

- **supervisor_state_machine.py**, qui choisit le meilleur comportement à adopter selon les réponses des capteurs

- **controllers**, les classes dans ce dossier s'occupent des différents comportements du robot

Le but est de chercher un point précis sur une carte est de s'y rendre
Les obstacles seront repérés avec les 9 capteurs infrarouges pointant dans toutes les directions du robot, et sa direction sera décidé par la vitesse de ses 2 roues. Quand les 2 roues vont à la même vitesse, le robot avancera en ligne droite, sinon le robot sera entrain de tourner. Les roues ont aussi un compteur pour savoir précisément le nombre de rotations qui ont été effectuées.

L'interface **robot_supervisor_interface.py** contient les fonctions read_proximity_sensors() qui retourne les réponses des 9 capteurs, read_wheel_encoders() qui retourne le nombre total de rotation des roues et set_wheel_drive_rates(v_l, v_r) qui initialise la vitesse de la roue gauche et droite par v_l et v_r (en radian/s)
Mais au lieu de déterminer a chaque fois v_l et v_r, il choisit juste la vitesse v du robot et l'angle de rotation omega voulues et calcule mathématiquement v_l et v_r (voir la partie **A Nifty Trick: Simplifying the Model**)

L'idée est de faire une boucle qui détermine a chaque itération la position du robot et si oui ou non il y a des obstacles autour de lui. (Voir la partie **Estimating State: Robot, Know Thyself**)

A partir de ces données, il explique les différentes approches qu'on peut adopter :

1- l'approche **je me dirige directement vers l'objectif**, inutilisable quand il y a des obstacles

2- l'approche **j'évite à tout prix les obstacles,** qui considère les capteurs comme des vecteurs. En faisant la somme vectorielle des vecteurs où aucun obstacle n'a été détecté et en faisant de cette somme la nouvelle direction du robot, il évite bien les obstacles mais oublie l'objectif (Voir la partie **Python Robot Programming Methods: Avoid-Obstacles Behavior**)

3- l’approche **mélange des deux** qui utilise l’approche 1 quand il y a aucun obstacle et l’approche 2 quand il y en a un, mais ceci présente un risque de boucle infini où le robot reprend le même chemin qu’il essayait d’éviter dès que il n’y a plus d’obstacle

4- l’approche **suivre l’obstacle** calcule le vecteur parallèle à la surface de l’obstacle et le robot suit cette nouvelle direction jusqu’à dépasser l’obstacle (le code est compliqué et se trouve dans **follow_wall_controller.py**)
Cette approche suppose que les obstacles sont rectangulaires !

L’approche adoptée est un mélange de l’approche 1,2 et 4, quand il y a aucun obstacle, on utilise l’approche 1, sinon on utilise l’approche 4. Dans le cas où le robot se retrouve trop près d’un obstacle, on utilise l’approche 2.

Aucune ligne sur l’affichage du simulateur.