Programmation de robots

Exercices 2

1 Introduction

Nous allons permettre à notre robot NXT de *sentir* son environnement en exploitant un peu mieux son capteur de distance. Ce capteur sera monté sur une tourelle, à l'aide d'un troisième moteur, de façon à permettre de prendre des relevés panoramiques de la distance des obstacles à proximité du robot. Ces relevés seront exploités dans différentes situations. Nous commencerons par nous en servir pour déterminer comment sortir d'une impasse, en faisant rouler le robot vers les grands espaces vides. Une autre possibilité serait d'utiliser le relevé comme un moyen de se repérer à partir d'une cartographie préalablement établie des obstacles.

Ce travail durera plusieurs séances et il fera l'objet d'une évaluation. Les grandes étapes seront les suivantes :

1. Écriture d'un simulateur [aujourd'hui]
2. Assemblage du robot
3. Mise au point d'un algorithme d'échappement
4. Implémentation sur robot
5. Améliorations et autres problèmes

Figure 1 – NXT piégé ?

2 Simulateur

Le module `turtle` crée un monde simulé dans lequel nous pouvons déplacer un ou plusieurs curseurs graphiques (appelés tortues). Nous allons simuler notre problème de télémétrie dans ce monde à deux dimensions qui correspond à une vue du dessus d'un terrain plat. Cette simulation sera utile pour entrer dans le problème et anticiper sur certaines questions de programmation du robot, et plus tard nous pourrons nous en servir pour percevoir le monde réel du robot à sa façon.

Question A. Votre programme. Comme souvent en programmation, et en particulier avec python, nous allons utiliser un certain nombre de bibliothèques pour éviter d'avoir à écrire certaines parties de notre programme. Créer un programme `TelemetreTortue.py` et y insérer les lignes :

```
from sympy import N
from sympy.geometry import Polygon, Line, Point, intersection
import turtle
import random
from math import *
```

Il y aura bien entendu la bibliothèque `turtle`, dont nous nous servirons pour l'affichage. Nous l'utiliserons pour représenter les obstacles du terrain et le robot. Pour simplifier ces obstacles seront des carrés de différentes dimensions mais vous pourrez tout à fait étendre la simulation à n'importe quelle sorte de polygones.

Question B. Prise en main de turtle. Sauriez-vous dessiner un triangle isocèle ou un carré avec une tortue ? Dessiner une maison, un conifère à l'aide du carré et d'un triangle ? Dessiner un hameau de dix maisons dans les conifères ? Dessiner un octogone, un cercle ? Si vous pensez savoir le faire passez à la suite sans perdre de temps.

Instruction `turtle`	Effet
`T = turtle.Turtle()`	créer une nouvelle tortue `T`
`T.fd(50)`	avancer `T` de 50 pixels
`T.back(50)`	reculer `T` de 50 pixels
`T.penup()`	relever le crayon de `T`
`T.pendown()`	abaisser le crayon de `T`
`T.left(60)`	tourner `T` de 60° vers la gauche
`T.setheading(60)`	donner un angle de 60° à `T`
`T.setpos(x, y)`	placer T en x, y
`T.hideturtle()`	ne plus afficher la tortue
`T.xcor()`	coordonnée x de la tortue
`T.heading()`	orientation de la tortue en degrés
`T.color("pink")`	choisir une couleur de crayon rose

Instruction `sympy`	Effet
`A = Point(x, y)`	Créer un point (peut être vu comme vecteur)
`Line(A, B)`	Créer la droite (AB)
`Ray(A, B)`	Créer la demi-droite [AB)
`Polygon(A, ...)`	Créer un polygone fermé à partir des points en argument
`intersection(f, g)`	Trouver tous les points d'intersection entre deux formes f et g
`A.dot(B)`	Produit scalaire entre les vecteurs A et B
`A.distance(B)`	Distance du point A au point B (peut être sous forme symbolique)
`N(A.x)`	Valeur numérique approchée de l'abscisse de A

La bibliothèque `sympy` de calcul mathématique symbolique offre des fonctions de géométrie élémentaire, que nous utiliserons essentiellement pour représenter les obstacles et simuler les relevés télémétriques. Pour cela nous supposons qu'une mesure de télémétrie est portée par une demi-droite partant du robot dans une direction précise et qu'il s'agit de trouver la distance à la première intersection de cette demi-droite avec les segments qui constituent nos obstacles, comme sur la figure 2.

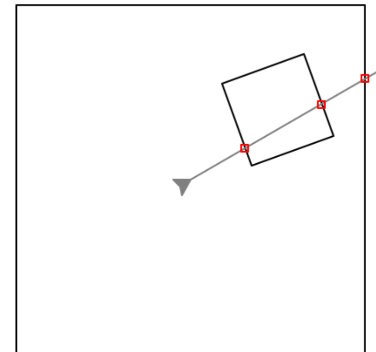

Figure 2 – Intersections

Pour obtenir de l'aide sur une bibliothèque vous pouvez utiliser la fonction `help()`. Par exemple `help("turtle.forward")` vous donnera de l'aide sur la fonction `forward` de la bibliothèque turtle.

Ces deux biblothèques, `sympy` et `turtle` offrent des fonctionnalités complémentaires : `turtle` va nous permettre d'afficher notre simulation, de dessiner les obstacles, de déplacer le robot mais pas de nous rendre compte si le robot touche un obstacle ou de simuler une prise de mesure de distance (une télémétrie) ; `sympy` va nous permettre de représenter mathématiquement les obstacles, et de calculer de nouvelles données à partir de leurs représentation, par exemple la distance au premier obstacle dans une direction.

Question C. Obstacle carré. écrire une fonction `faire_boite(x, y, cote, angle = 0)` qui dessine un carré de centre `x, y` de coté `cote` penché selon un angle `angle` valant par défaut zéro.

Question D. Tests. En écrivant ce qui suit dans votre fichier `TelemetreTortue.py`,

```
if __name__ == '__main__':
    test1()
    raw_input()
```

où `test1()` est une fonction de votre choix, vous pourrez utiliser votre programme de deux façons : soit comme bilbiothèque en l'inclant dans un autre programme, dans ce cas le main que vous venez d'écrire ne sera pas exécuté ; soit comme programme à part entière, dont l'exécution provoquera celle de `test1()`. De plus l'appel à `raw_input` étant bloquant votre programme ne quittera pas avant que vous ne tapiez entrée au clavier.

Écrire une fonction qui teste le bon fonctionnement de `faire_boite`.

Question E. Obstacle dans sympy. faire en sorte que `faire_boite` retourne un polygone représentant exactement le carré (mêmes coordonnées que dans le dessin). Tester (utiliser `print`).

Question F. Liste d'obstacles. Dans une fonction de test, construire une liste d'obstacles avec `faire_boite`. Cette liste contiendra des polygones carrés parmis lesquels au moins un carré centré en 0, 0 dont les bords délimiteront le terrain du robot. La position initiale du robot est 0, 0.

Question G. Demi-droite et intersections. Construire une demi-droite issue du point 0, 0 et d'angle 30 degrés.

Question H. Intersections. Inspirez vous du code suivant pour trouver la distance du premier obstacle du robot dans la direction pointée par la demi-droite.

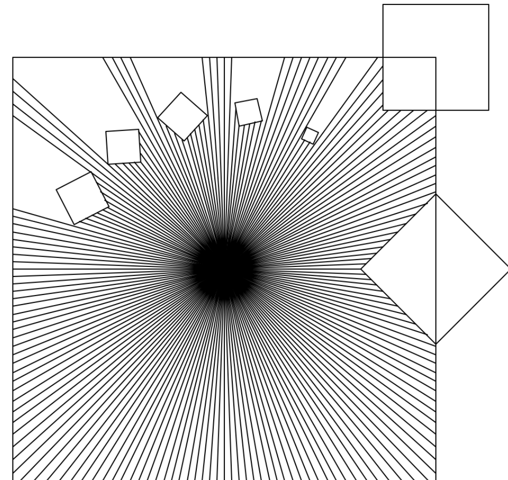

Figure 3 – Simulation de télémétrie

```
# Trouver les intersections
intersections = [p for boite in obstacles for p in intersection(boite, demi_droite)]
# dessiner les intersections
[faire_boite(N(p.x), N(p.y), 4, couleur="red") for p in intersections]
# trouver la distance
d = min([A.distance(p) for p in intersections])
```

Question I. Télémétrie. Faire effectuer un tour d'horizon au robot (une tortue) avec une télémétrie tous les cinq degrés. Afficher le résultat comme dans la figure 3, de façon à avoir une représentation plus nette ce que perçoit le robot avec son capteur.

3 Construction du robot

S'il vous reste du temps, commencez à modifier le montage du robot. Il faut en particulier que le capteur de distance (tête du robot) soit monté comme une tourelle qui puisse tourner sur elle même à 360 degrés, actionnée par un troisième moteur et si possible à la verticale du centre des roues. L'image figure 4 vous montre ce type de montage. Attention aux câbles.

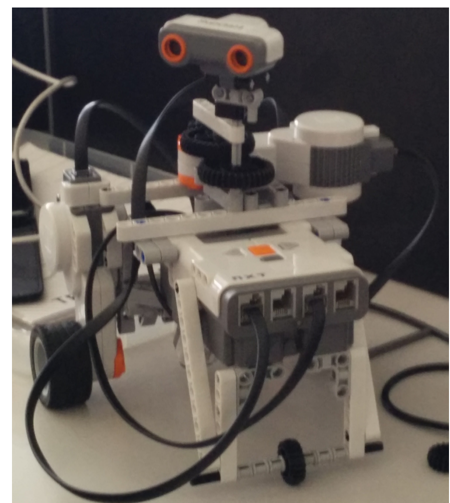

Figure 4 – NXT et sa tourelle

Question J. Finesse angulaire. Combien de positions différentes pouvez-vous commander à votre tourelle de prendre ? Autrement dit en combien de déplacements arrivez vous à un tour complet de la tourelle ?