# TP 3 - PROGRAMMATION ROBOTIQUE - INFO2

YANN CHEVALEYRE & PIERRE BOUDES

Aujourd'hui nous allons traiter le problème de la **localisation.**

Nous supposerons que le robot connait parfaitement le plan, mais pas sa position, il devra essayer de retrouver ses coordonnées $(x, y)$ dans le plan. Pour cela, il utilisera son télémètre.

## 1. Introduction

Dans ce TP, on commencera les programmes par:

```
from robosimnxt2 import *
init('robot_obstacles')
```

Le package `robosimnxt2` definit des commandes de navigation fonctionnant à la fois sur NxT et sur le simulateur (pas tout à fait les mêmes que la dernière fois).

```
# Commandes de navigation
av, td, tg
```

Les commandes `td` et `tg` prennend en argument un angle égal à $30, 60$ ou $90$ seulement (pour que le NxT puisse faire facilement de même). Le résultat est bruité.

Le capteur de télémétrie est accessible avec les fonctions suivantes:

```
# Capteurs, mesures et informations
telemetre, telemetre_coords_list, diametre_robot, taille_terrain
```

## 2. Monte-Carlo Uniforme sur (x,y,angle)

Dans cette partie, le robot du simulateur va devoir retrouver ses propres coordonnées et son orientation approximativement. L'idée est de tirer plein de triplets $(x, y, angle)$ candidats, de calculer ce que le télémètre observerait si le robot était localisé sur ce triplet, et de comparer avec le résultat du télémètre de robot réel. Ainsi, on fait des hypothèses, et on les triera selon leur vraissemblance.

- D'abord, créez une fonction `robo_tele_liste()` qui utilise `telemetre(rel_angle)` pour construire une liste de distances aux obstacles autour du robot. La liste devra partir de l'angle -180 et aller jusqu'à l'angle 180 (relativement à l'orientation du robot) par pas de 30 degrés. Consultez l'aide de la fonction `telemetre` pour vous aider.

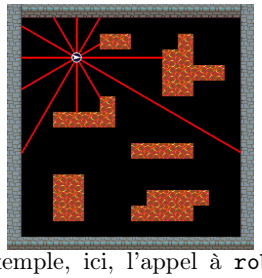

FIGURE 2.1. par exemple, ici, l'appel à `robo_tele_liste` renvoie la liste de distances `[113.0, 130.36, 93.64, 81.0, 93.64, 55.56, 176.0, 387.98, 95.88, 112.0,226.24, 130.36]`

- Créez une fonction `hypo_tele_liste()` qui tire aléatoirement un triplet $(x, y, angle)$ et qui renvoie ce triplet ainsi qu'une liste de distances aux obstacles, comme si le robot était en $x, y$ orienté selon l'angle *angle* (donc cette fonction fait une `hypo`thèse)

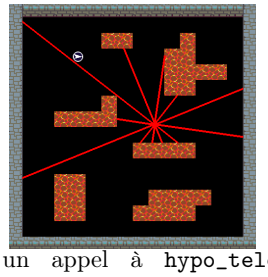

FIGURE 2.2. Ici, un appel à `hypo_tele_liste` a renvoyé `301,249,8,[77.78, 341.26, 167.31, 141.42, 74.81, 194.23, 181.72, 61.22, 40.85, 38.32, 48.41, 290.24]`. Le triplet de coordonnées est donc $(301, 249, 8)$.

- Créez une fonction `similarite_listes(l1,l2)` qui prend deux listes de réels, et qui renvoie la somme des valeurs absolues des différences entre les deux listes[1].
- Créez une fonction `genere_hypothese` qui appelle `robo_tele_liste` et `hypo_tele_liste`, et qui renvoie `(x,y,a,d)` où $(x, y, a)$ est le triplet renvoyé par `hypo_tele_liste` et $d$ est la similarité entre les deux listes obtenues.
- Créez une fonction `genere_hypotheses_triees` qui appelle un grand nombre de fois `genere_hypothese` (par exemple 200 fois) et qui renvoie une liste `[(x_i,y_i,a_i,d_i)...]` telle que les `(x_i,y_i,a_i,d_i)` sont les quadruplets renvoyés par `genere_hypothese` triés dans l'ordre décroissant des `d_i`.
- Affichez des cercles centrés sur les 10 premières hypothèses de la liste précédente.

[1] en réalité, il s'agit d'un dissimilarité

## 3. Monte-Carlo Uniforme sur (x,y)

Maintenant, on ne tirera aléatoirement que les coordonnées $x, y$ et pas l'angle. De plus, au lieu d'un pas de 30 degrés, on prendra un pas plus fin de 10 degrés. Mais pour comparer deux listes de distances, il faudra les comparer en testant toutes les rotations possibles de la liste.

Par exemple, supposons que le télémètre du robot renvoie:

```
l1=[64, 30, 46, 96, 51, 26, 41, 64, 64, 25, 60, 69, 71, 88, 76, 89,
69, 12, 94, 17, 63, 69, 62, 62, 40, 87, 68, 50, 17, 71, 70, 100, 96, 71,
36, 75]
```

Et supposons que le simulateur renvoie la liste:

```
l2=[100, 51, 29, 39, 59, 68, 28, 57, 75, 74, 84, 78, 95, 71, 6, 91,
14, 65, 67, 56, 63, 42, 84, 71, 47, 14, 66, 67, 99, 101, 73, 42, 80, 65,
35, 45]
```

On voit que $l2[i] \backsimeq l1[i+3]$ donc le meilleur décalage entre les deux listes (celui qui minimise leur dissimilarité) est de 3, ce qui fait un angle de 30 degrés.

- Créer une fonction `similarite_decalage_listes(l1,l2)` qui renvoie le décalage qui minimise la dissimilarité entre ces deux listes.
- Recodez le TP avec cette fonction.