projet robotique RRRRR

Arthur GONTIER, Jonathan FONTAINE

December 2019

1 Introduction

Le but de ce projet est de calibrer un robot RRRRR, puis de dessiner un cercle dans son espace de travail, enfin de relier deux points de l'espace de travail sans en évitant des obstacles.

Notre travail ce trouve dans le fichier python : 'projet.py', pour éxécuter la suite du projet il faut lancer un terminal python puis d'utiliser la commande :

```
exec(open("projet.py").read())
```

Puis lancer les commandes relatives au projet. Les parties path planning et path following ne peuvent être lancer ensemble (l'affichage sera illisible) il est donc nécessaire de relancer le script.

2 Calibration

La partie calibration est une adaptation du fichier "RR - Calibration demo" donnée en cours. Pour obtenir nos résultats il faut lancer la commande :

```
calibrated_architecture = main_calibration()
```

- statut : satisfied
- erreur : 2.842492951564968
- a_1^1 = -22.50740768
- a_2^1 = 0.09213397
- a_1^2 = 22.40329459
- a_2^2 = 0.05412184
- l_1 = 17.98993144
- l_2 = 18.11024034
- l_3 = 17.8925407
- l_4 = 17.60744955

error : 2.842492951564968 result : [-22.50740768 0.09213397 22.40329459 0.05412184 17.98993144 18.11024034 17.8925407 17.60744955]

Pour réaliser ces mesures, nous avons prit la liberté de modifier le robot, en ajoutant un return à la fonction actuate, afin de détecter si le robot a rencontrer un singularité.

les angles sont testés de 0 à 90 pour q1 et de 135 à 180 pour q2 puis de 0 à 45 pour q1 et 90 à 180 pour q2.

Le tout avec un pas de 10.

```
f1 = -l3**2 + (-a1 + x1 - l1*math.cos(q1))**2 + (x2 - l1*math.sin(q1) - a2)**2
```

```
f2 = -l4**2 + (-a3 + x1 - l2*math.cos(q2))**2 + (x2 - l2*math.sin(q2) - a4)**2
```

3 Path following

Pour obtenir des résultats dans cette section, il faut lancer la commande `main_path_folowing()`

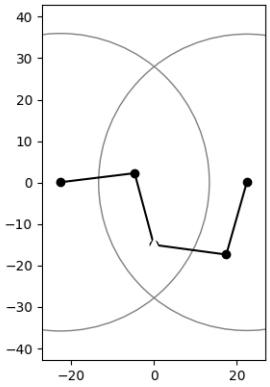
On souhaite suivre le cercle de centre x0 = (0,-20) et de rayon r = 5. on le discrétise avec la formule $r(\cos(t), \sin(t)), t \in [0, 2\pi]$

Pour le point de départ, on choisi (0,-15), on résout avec ibex pour trouver les commandes et on obtiens les quatre solutions suivantes :

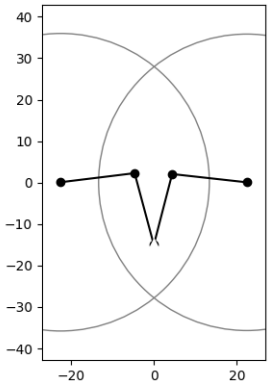
— solution n 1 = ([4.98004458289678, 4.980044582896783] ; [4.435230218253536, 4.435230218253539]) **Singularity**

— solution n 2 = ([4.98004458289678, 4.980044582896783] ; [3.0312718784439, 3.031271878443902]) **Singularity**

— solution n 3 = ([0.1217809921733804, 0.1217809921733817] ; [4.435230218253536, 4.435230218253539])

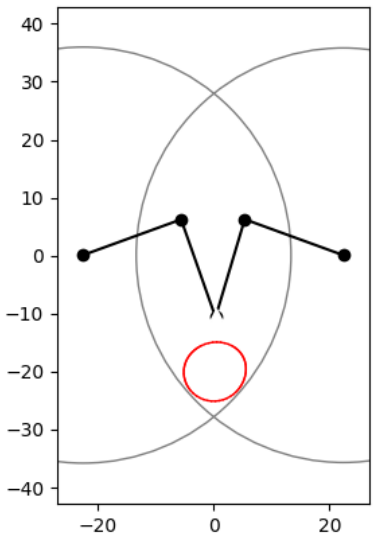


— solution n 4 = ([0.1217809921733804, 0.1217809921733817] ; [3.0312718784439, 3.031271878443902])

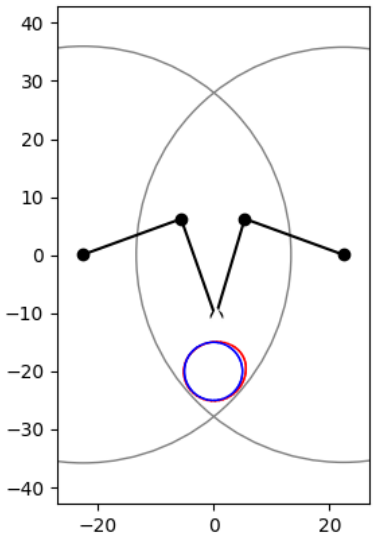


On test les commandes ci-dessus avec notre robot et on choisit la solution 4 comme point de départ.

On dessine d'abord en rouge le cercle avec le robot non calibré :



On observe que le cercle n'est pas très circulaire. On dessine alors par dessus le cercle avec le robot calibré.



Ce qui nous donne un cercle bien plus proche de ce que l'on souhaite.

4 Path planning

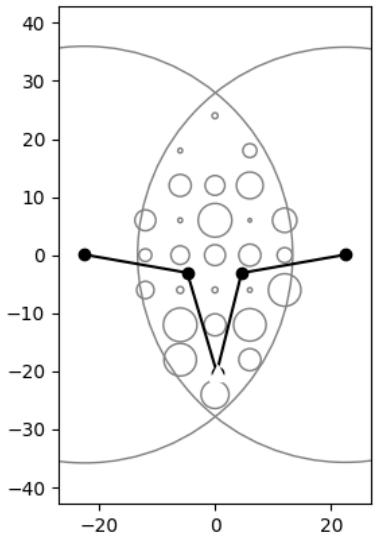
Pour obtenir les résultats sur ce problème, il faut lancer la commande `main_path_planning()`

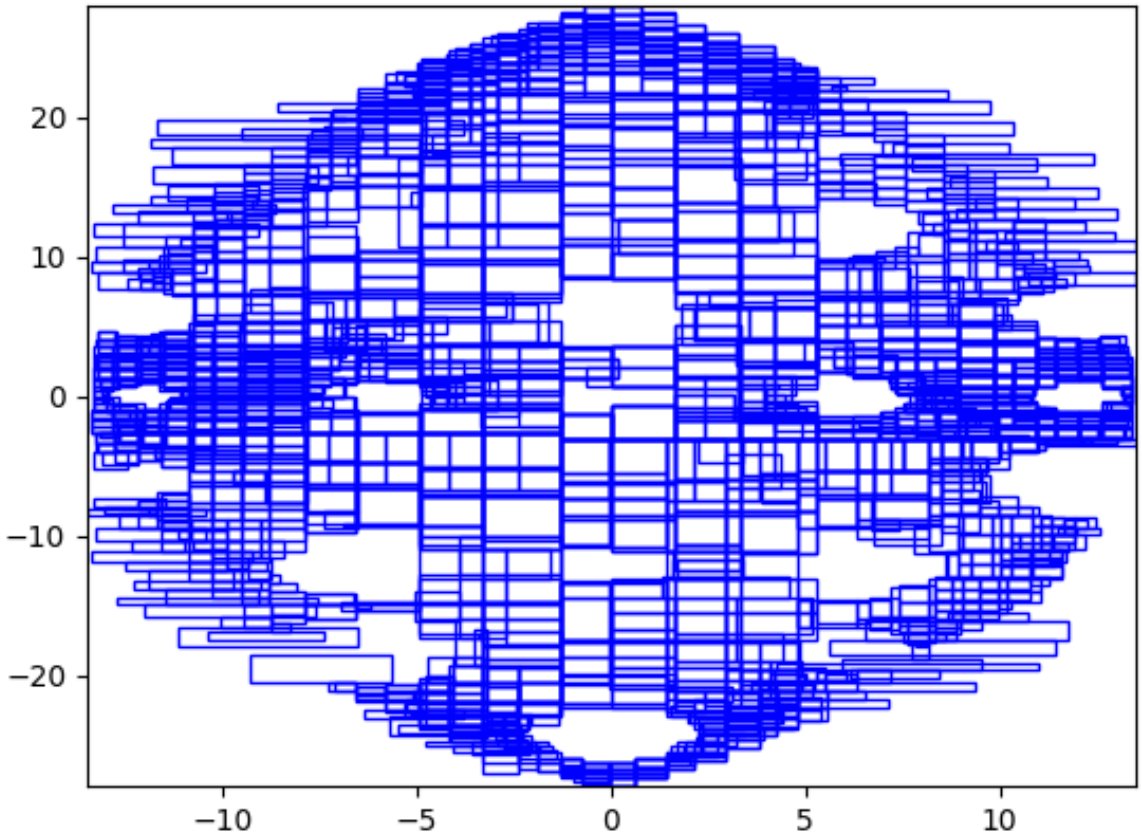
On veux se déplacer de (0,-15) à (0,15) sans entrer en collision avec les obstacles suivants : chaque obstacle est un cercle (x,y,r) de coordonnée x,y et de rayon r.

Obstacles : `[(-12, -6, 1.5),(-6, 18, 0.4),(6, -12, 2.8),(-12, 0, 1.1),(0, -24, 2.4),(6, -6, 0.4),(-12, 6, 1.8),(0, -12, 1.9),(6, 0, 1.9),(-6, -18, 2.8),(0, -6, 0.5),(6, 6, 0.3),(-6, -12, 2.9),(0, 0, 1.8),(6, 12, 2.3),(-6, -6, 0.6),(0, 6, 2.9),(6, 18, 1.2),(-6, 0, 1.6),(0, 12, 1.7),(12, -6, 2.8),(-6, 6, 0.4),(0, 24, 0.5),(12, 0, 1.3),(-6, 12, 1.9),(6, -18, 1.9),(12, 6, 2.1)]`

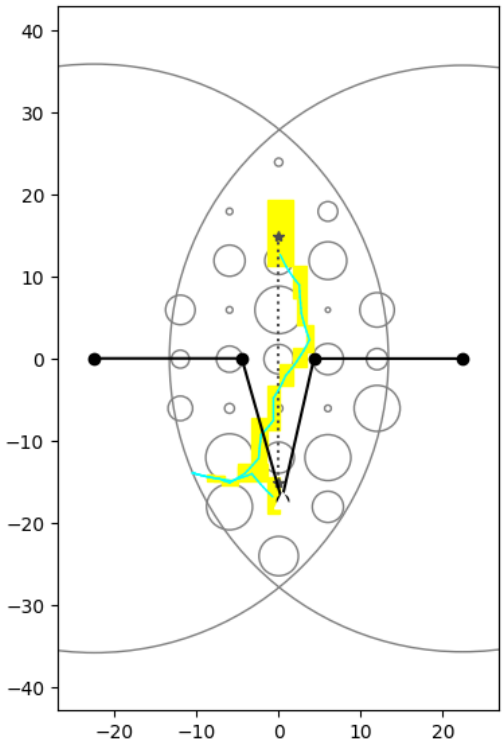
Pour chaque cercle (c_1, c_2,r) on ajoute les contraintes suivantes pour la résolution dans le modèle ibex :

$$(x_1 - c_1)^2 + (x_2 - c_2)^2 > r^2;$$

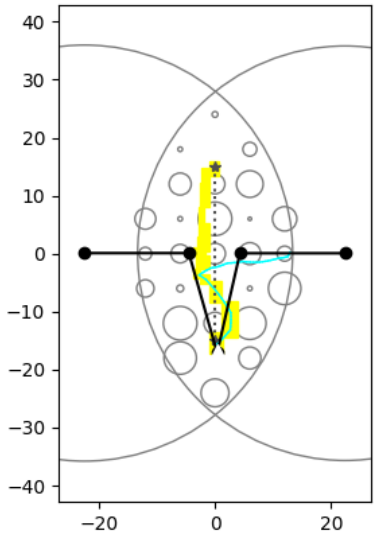




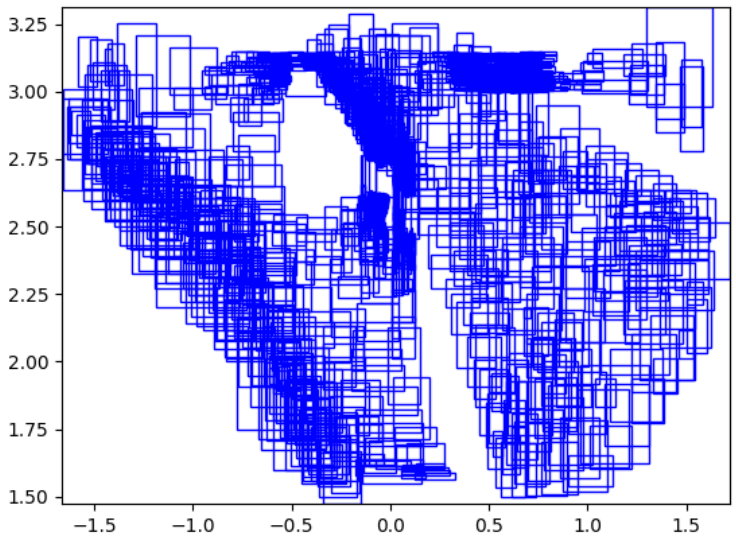
Dans notre premier jet avec une boite minimale de ibex à 0.5, on obtient le chemin suivant :



On observe que lorsque le robot part de la position (0,-15), il est dans la configuration 3 de la question 1, on souhaite alors empêcher ce chemin en interdisant la configuration 3. Nous obtenons alors :



On observe que le robot devient incohérent et si on affiche les commandes :

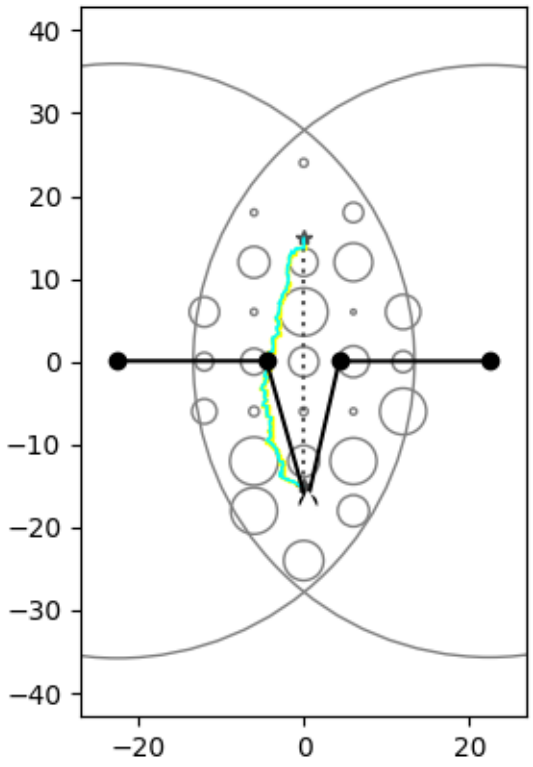


On observe que les composantes connexes sont se chevauchent, ce qui est très problématique.

La configuration initiale est cette fois avec le bras droit en haut et le bras gauche en bas, de plus on réduit la taille max des boites à 0.5

On décide donc de contraindre les deux bras à être en haut.

ce qui nous donne le chemin optimal suivant :



Ainsi que l'espace des positions et des commandes suivants (lancer la commande `main_plot_axe()`) :