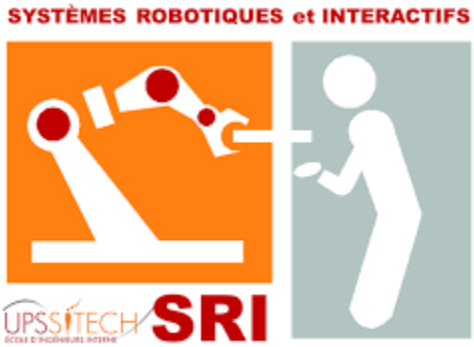
Crampette Olivier
Danton Laloy Solal

TP3 - Optimisation de placement de robots



1-Travail

Quelles sont les variables du problème ?

Les variables sont les trois positions des robots (xi,yi).

Écrire la fonction à minimiser/maximiser:

La fonction à minimiser est l'aire du triangle formé par les trois robots:

Théorème : Aire d'un triangle par les déterminants

L'aire d'un triangle de sommets $(x_1 ; y_1), (x_2 ; y_2)$ et $(x_3 ; y_3)$ est donnée par

$$\text{aire} = \frac{1}{2}\left|\det\begin{pmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{pmatrix}\right|.$$

Ici on a donc : **aire = ½ | x1(y2-y3) + x2(y3-y1) + x3(y1-y2) |**

Quelles sont les contraintes à prendre en compte ?

Nous avons 3 contraintes cités dans l'énoncé:

-Champ d'action des robots à savoir:
  STAUBLI : rayon d'action= 1000 mm
  MITSUBISHI : rayon d'action= 2000 mm
  EPSON : rayon d'action= 750 mm
-Longueur du bus reliant les robots : 6520 mm.
-Pas de robot au-dessus de 750 mm au-dessus de l'origine.

Formalisme :

Les robot doivent atteindre le centre de tâche (0,0) :
  sqrt(xi² + yi²) < Champ d'action
Longueur du bus = périmètre:
  sqrt((x2-x1)²-(y2-y1)²) + sqrt((x3-x2)²-(y3-y2)²) + sqrt((x3-x1)²-(y3-y1)²) < 6520
Hauteur max:
  yi < 750

Quel choix pour les fonctions de scipy.optimize?

Nous avons choisi d'utiliser minimize() pour plusieurs raison:

L’utilisation de least-square comme au tp2 risque d’être compliqué car nous n’avons aucune mesure comme au tp2. Least_square se base sur l’erreur entre valeur mesurée et valeur calculée il est donc assez compliqué de le mettre en œuvre!

Concernant minimize() la fonction a plusieurs paramètres et peut gérer les problème linéaire et non linéaire plus globaux comme nous avons ici. Parmis les paramètres vu dans la doc on a donc la fonction a optimiser, les contraintes si il y en a , une valeur initiale pour plus de précision.

Il y a aussi le paramètre “method” qui nous permet de choisir l’algo si on en veut un spécifique. Sans le mettre il prend par défaut le meilleurs pour notre utilisation entre: `BFGS, L-BFGS-B, SLSQP` slsqp est une sorte de least square comme au tp2.

Pour résumer minimize() est approprié aux vu de nos données, des contraintes et du caractère non-linéaire de notre problème. Least_square seras imprécis car pour l’algo se base sur des données cibles, ce que nous n’avons pas.

2 - Optimisation de trajectoires

Dans cette partie on ajoute les temps de traitement des différents robots. Notre fonction a optimiser devient la fonction du temps total (un simple **t = d/v** pour chaque section) . Les résultats de la partie 1 nous permettent d’ajouter une contrainte supplémentaire pour la fonction de temps.

On aurait pu voir le problème différemment et créer une nouvelle fonction a minimiser avec un facteur pour les deux composantes aire et temps. Cependant ici il est bien renseigné d'être précis à l' unité près.
Nous trouvons donc un temps optimisé à 62 sec

Finalement, nous trouvons que la distance entre le STAUBLI et le MITSUBISHI est un peu augmentée, en effet on a v1 = 2000mm/s ce qui est assez rapide, donc rajouter de la distance ne fait pas perdre trop de temps.

Ensuite nous trouvons que la distance entre le MITSUBISHI et le EPSON est un peu réduite, de même que pour le EPSON et le STAUBLI, car les vitesses v2 = 50mm/s et v3 = 100mm/s sont largement inférieures à v1 donc rajouter de la distance fait perdre plus de temps.

Les résultats sont donc cohérents avec les données. Cependant peut être faire attention à ne pas trop accumuler de contraintes basées sur de précédentes approximations sous peine de voir le résultat un peu biaisé.