L'OPTIMISATION DU TRAJECTOIRE D'UN ROBOT MOBILE ENTRE 2 POINTS (en présence d'obstacle) AVEC PSO ET DIJKSTRA

Obstacles rectangulaires

Réalisé par :

- **EL HADEG Rida**

Sous la direction de :

- **Mr. AYAD Hassan**

UNIVERSITE CADI AYYAD

FACULTE DES SCIENCES ET TECHNIQUES

Année universitaire : 2018/2019

1- Le But de ce projet :

Le but de ce projet consiste à optimiser la trajectoire d'un robot mobile entre un point X de départ et un autre point Y d'arrivé en présences des obstacles rectangulaires et pour cela on va utiliser deux algorithmes différents sont :

- Algorithme PSO :
- Algorithme de Dijkstra

Et on termine par une comparaison entre les deux méthodes

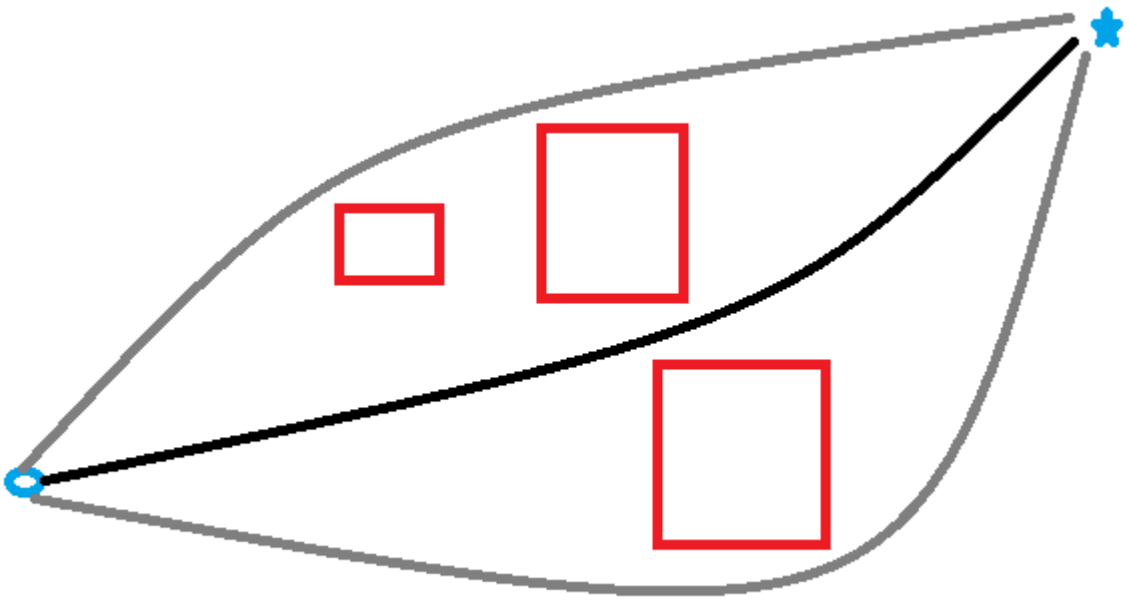
2- Optimisation par essaim de particules-PSO :

A- Introduction :

L'homme s'inspire de plus en plus de la nature qui l'entoure pour mettre en place des algorithmes simulant le comportement des animaux. En 1995, Russel Eberhart, ingénieur en électricité et James Kennedy, socio-psychologue, s'inspirent du monde du vivant pour mettre en place l'optimisation par essaim particulaire dont l'idée directrice est la simulation du comportement collectif des oiseaux à l'intérieur d'une nuée, et étudier la possibilité de son application pour contrôler les robots mobiles.

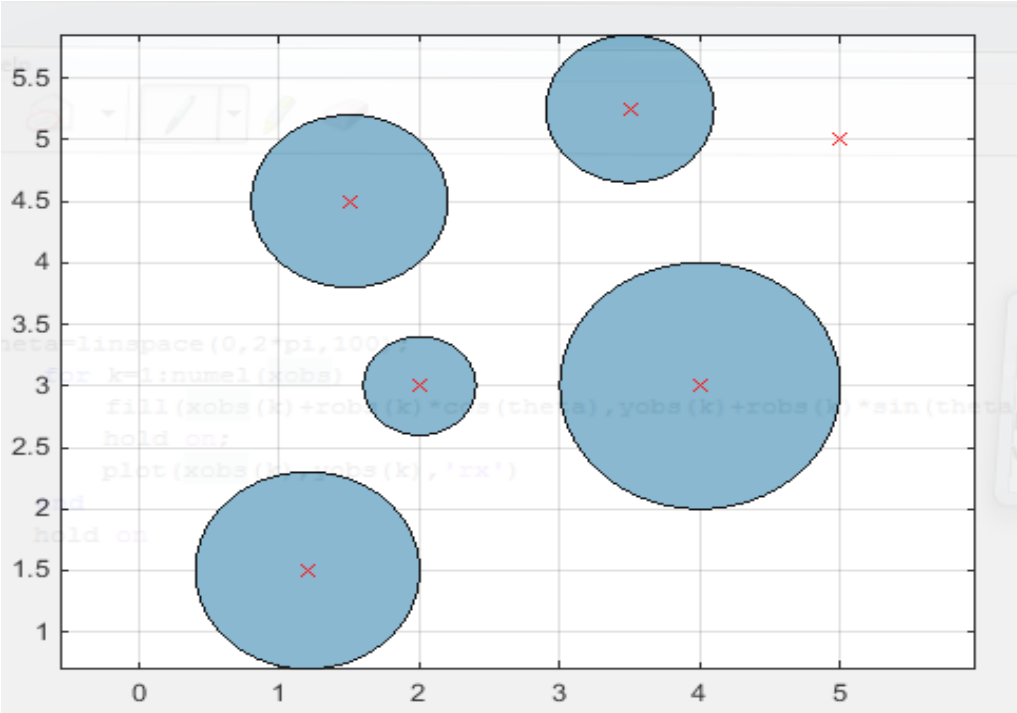
Le problème à résoudre peut souvent s'exprimer comme un problème d'optimisation : on définit une fonction objective, et on cherche à minimiser ou à maximiser par rapport à tous les paramètres concernés.

Pour notre cas le problème c'est que le robot doit arriver au point final sans qu'il touche les obstacles qu'ont une forme rectangulaire, plus que ça, il doit prendre le chemin le plus optimisé.



B- Obstacles rectangulaire :

Avant d'entamer l'objectif de ce projet qui consiste principalement à travailler sur un programme qui optimise le chemin du robot en présence des obstacles qu'ont une forme circulaire et le modifier de tel façon que le robot doit esquiver des obstacles rectangulaires, on doit traiter tout d'abord la partie du programme de base 'obstacles.m' qui permet de crée ces cercles.



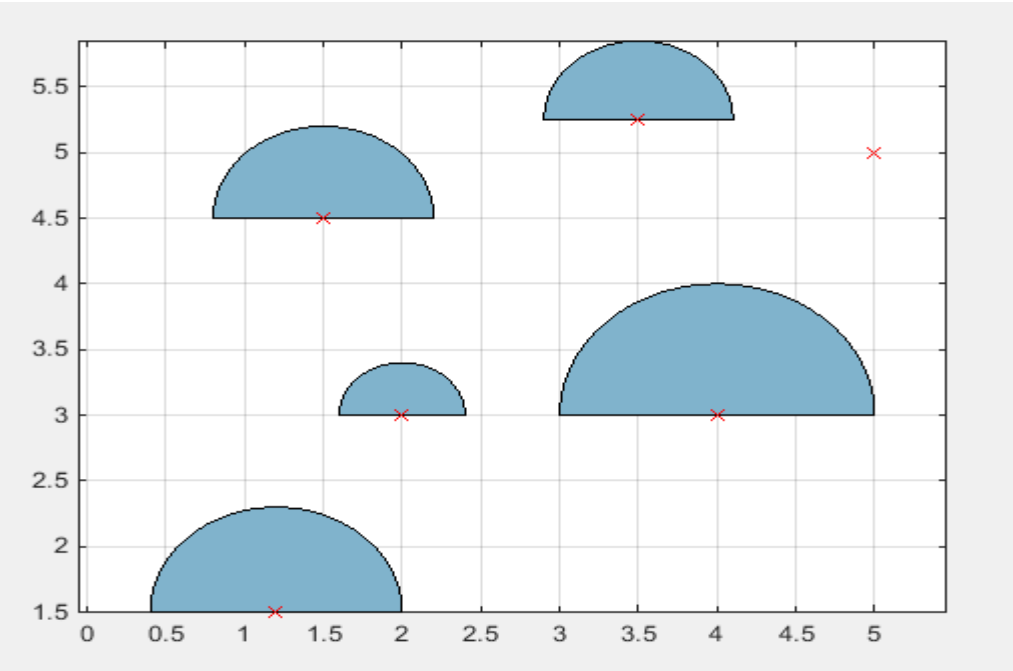
LE PROGRAMME :

```
theta=linspace(0,2*pi,100);
    for k=1:numel(xobs)
        fill(xobs(k)+robs(k)*cos(theta),yobs(k)+robs(k)*sin(theta),[0.5 0.7 0.8]);
        hold on;
        plot(xobs(k),yobs(k),'rx')
    end
    hold on
```

On voit que dans cette partie, le programme trace l'obstacle pour une angle thêta entre un intervalle A= [0 2*π] et pour N =100 ,ce qu'est logique parce que on veut tracé des cercles.

Pour un intervalle A= [0 π] et N =100 :

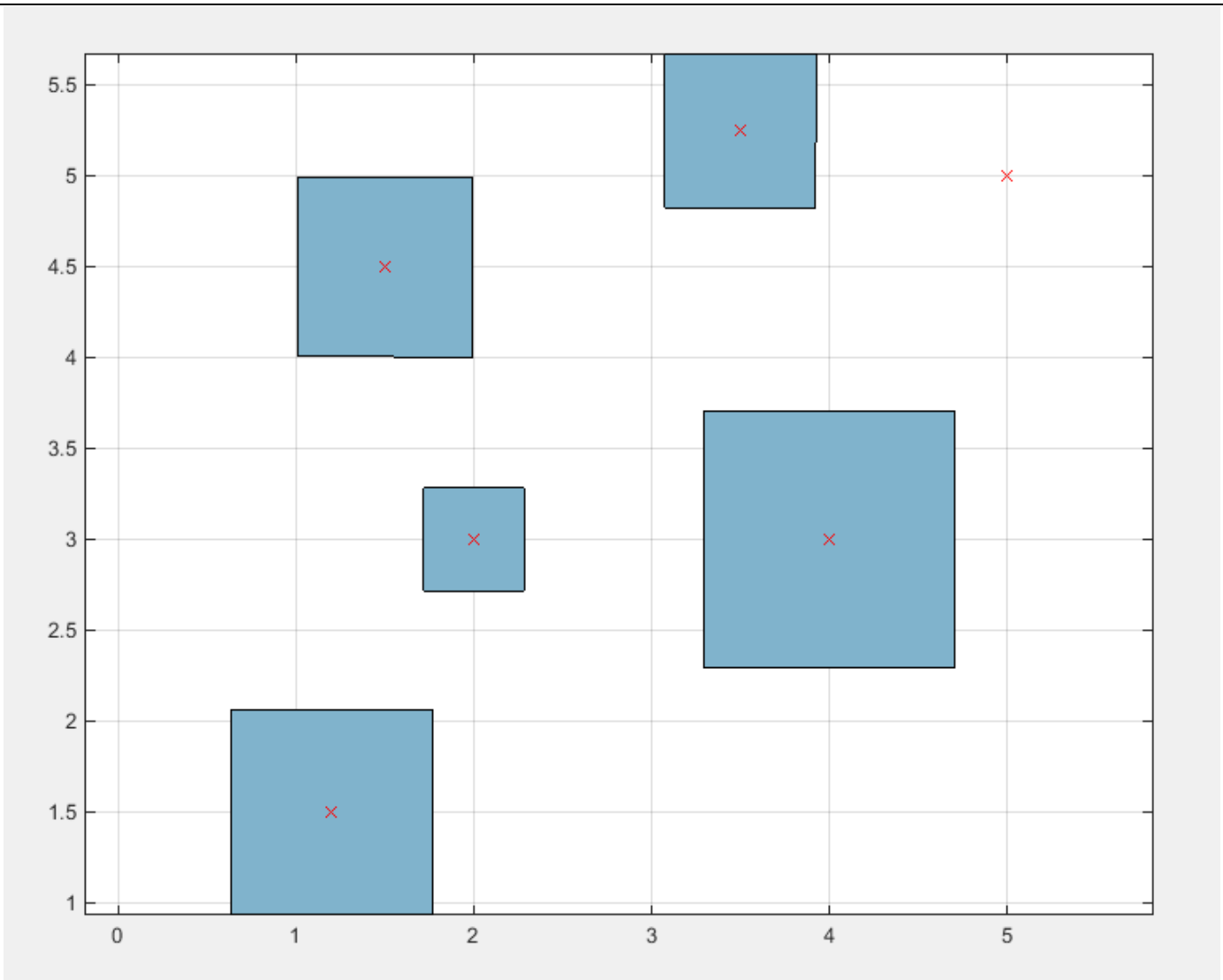
```
theta=linspace(0,pi,100);
    for k=1:numel(xobs)
        fill(xobs(k)+robs(k)*cos(theta),yobs(k)+robs(k)*sin(theta),[0.5 0.7 0.8]);
        hold on;
        plot(xobs(k),yobs(k),'rx')
    end
    hold on
```



On voit qu'on a des obstacles avec la forme d'une demi-cercle, donc l'idée consiste a joué sur ces deux paramètres A et N pour tracé des rectangles.

Après plusieurs essaies on trouve que pour A=[π/4 (3*pi/1.7146)] et N = 4 (ce qu'est équivalent au quatre sommets du rectangle) on arrive à crée des obstacles carrés.

```
theta=linspace(pi/4,(3*pi/1.7146),4);
    for k=1:numel(xobs)
        fill(xobs(k)+robs(k)*cos(theta),yobs(k)+robs(k)*sin(theta),[0.5 0.7 0.8]);
        hold on;
        plot(xobs(k),yobs(k),'rx')
    end
    hold on
```



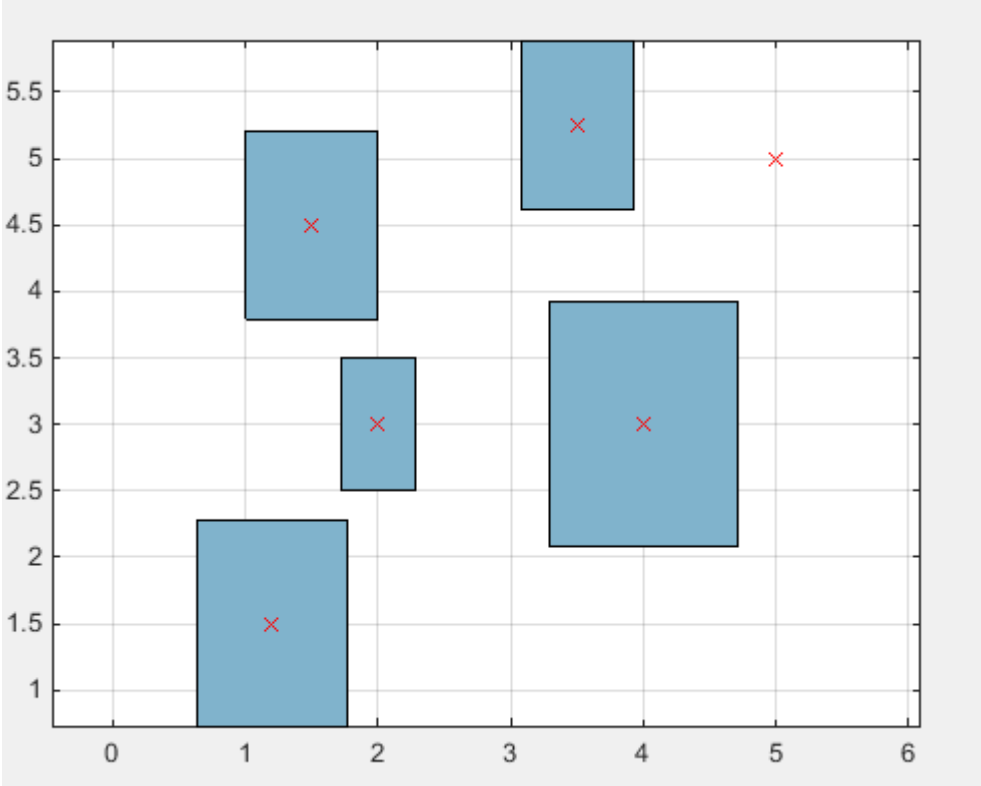
Le fait d'avoir des carrés c'est parce qu'on a travaillé avec robs (le rayon du cercle de base) comme longueur et largeur du rectangle. Pour régler ça, on rajoute la longueur comme donnée
et on modifier l'instruction `fill` qui permet de tracée l'obstacle, on remplace `yobs(k)+robs(k)*sin(thêta)` par `yobs(k)+long(k)*sin(thêta)` comme indiqué dans la figure ci-dessous :

```
xobs=[1.5 4.0 1.2 2 3.5];
yobs=[4.5 3.0 1.5 3 5.25];
robs=[0.7 1.0 0.8 0.4 0.6];
long=[1.5 0.8 0.4 0.5 1];

 % Target (Destination)
    xt=5;
    yt=5;



theta=linspace(pi/4,(3*pi/1.7146),4);
    for k=1:numel(xobs)
        fill(xobs(k)+robs(k)*cos(theta),yobs(k)+long(k)*sin(theta),[0.5 0.7 0.8]);
        hold on;
        plot(xobs(k),yobs(k),'rx')
    end
    hold on
plot(xt,yt,'rx')
```

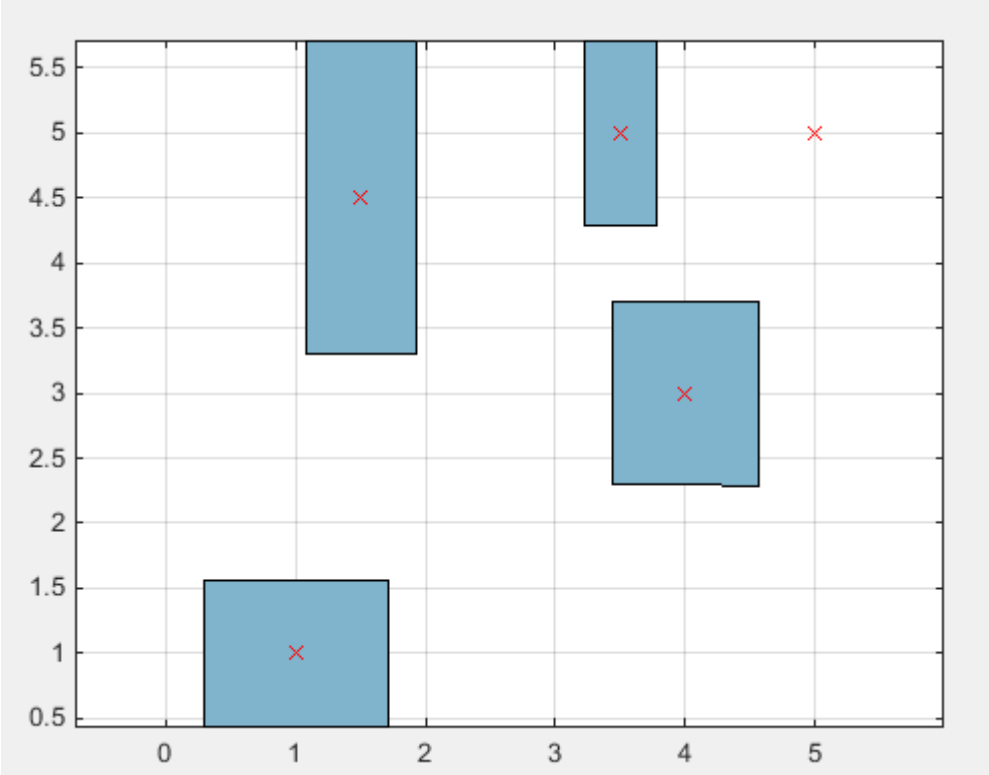
Remarque :

On doit faire les mêmes modifications dans les fichiers PLOTSOLUTION.M et CREATEMODEL

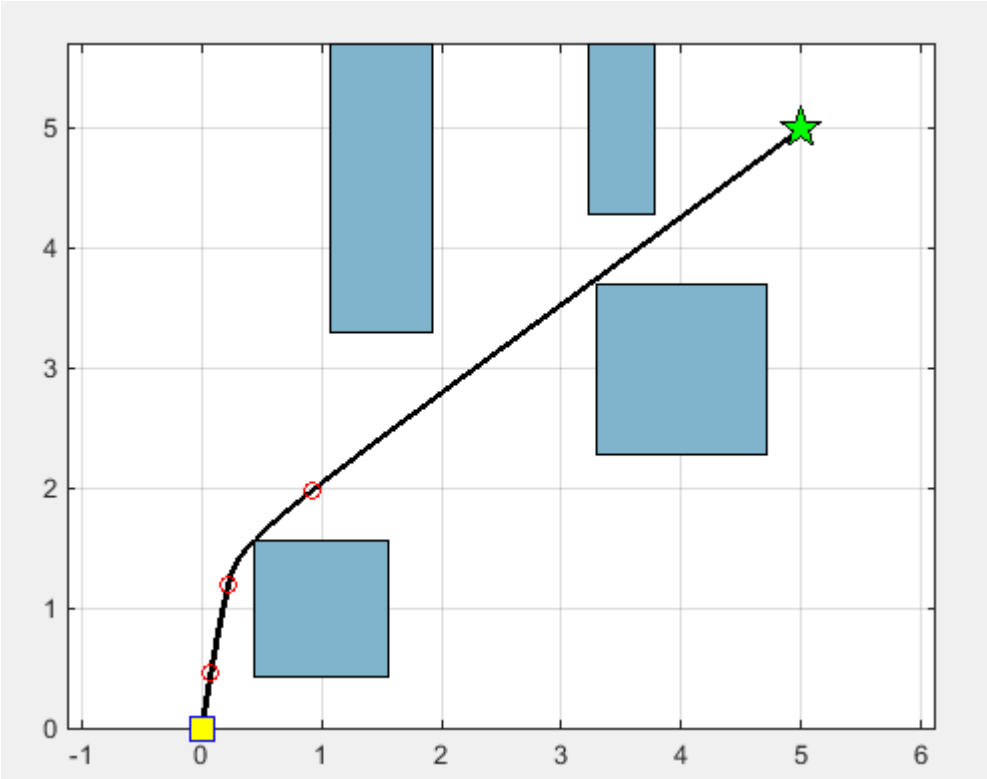
Le résultat final :

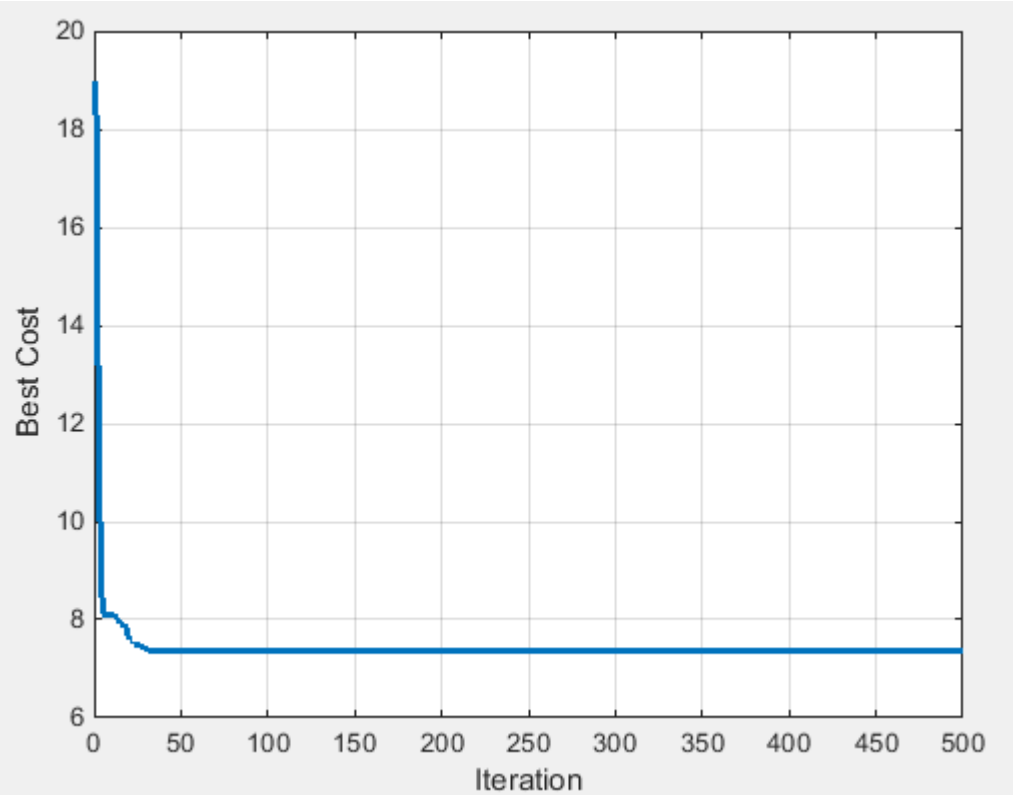


On change l'emplacement et le nombre des obstacles :



Et finalement on lance le programme principal et voici le résultat :





On voit que notre programme a réussi a trouvé le chemin le plus optimisé au bout de 35éme itération (à-peu-près).

3- ALGORITHME DE DIJKSTRA

A- Introduction :

L'algorithme porte le nom de son inventeur, l'informaticien néerlandais Edsger Dijkstra, et a été publié en 1959, et il sert à résoudre le problème du plus court chemin. Il permet, par exemple, de déterminer un plus court chemin pour se rendre d'une ville à une autre connaissant le réseau routier d'une région. Plus précisément, il calcule des plus courts chemins à partir d'une source dans un graphe.

Pour notre cas nous allons utiliser L'algorithme Dijkstra pour optimiser le trajectoire d'un robot mobile

B- La création des obstacles :

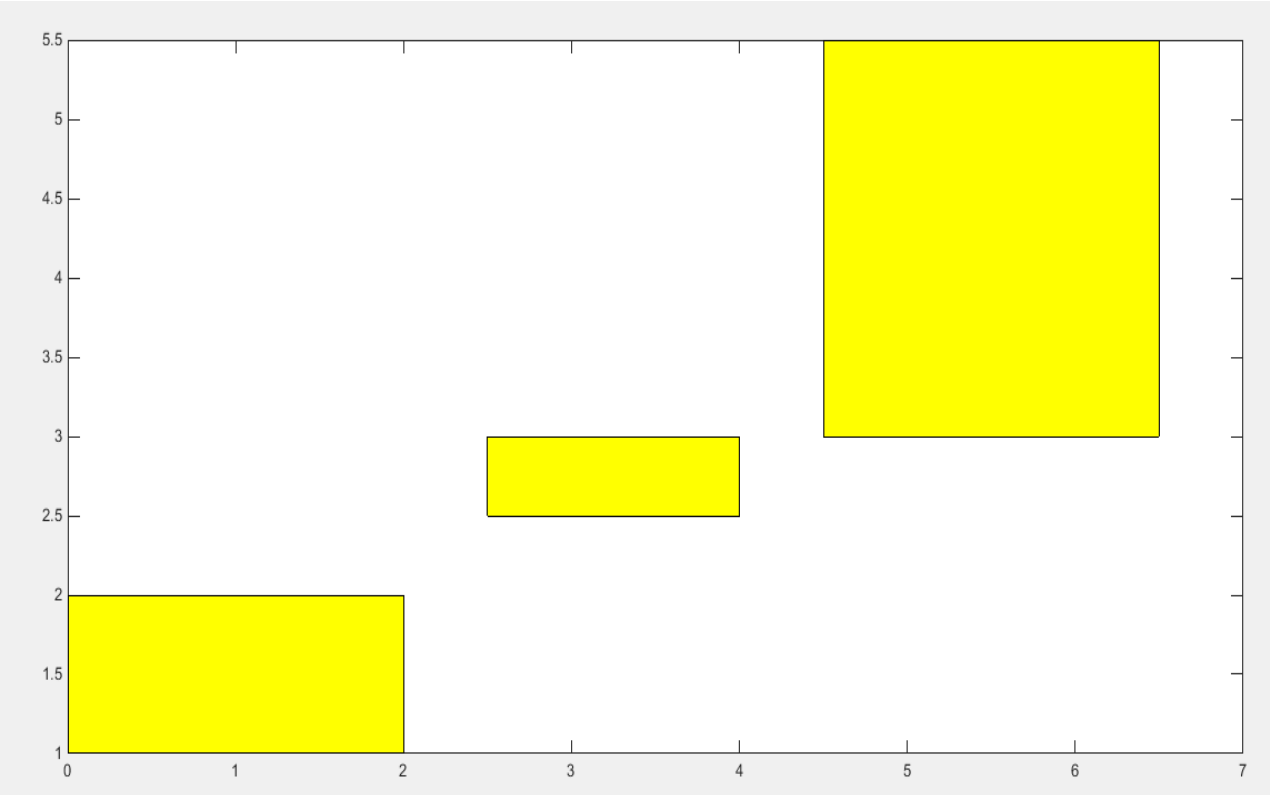
*Dans cette partie on va créer 3 obstacles rectangulaires donc on a besoin de créer 12 points (4*3 4 sommets/rectangle) plus le point de départ et le point d'arrivé (Target) :*

```
n=14;
crd = [0.5 0.5;
       0 1;
       0 2;
       2 2;
       2 1;
       2.5 2.5;
       2.5 3;
       4 3;
       4 2.5;
       4.5 3;
       4.5 5.5;
       6.5 5.5;
       6.5 3;
       7 7];
```

Après la création des points on va les lies pour crée des rectangles, pour cela on va utiliser les instructions suivants :

```
x1=[0;0;2;2]
y1=[1;2;2;1]

x2=[2.5;2.5;4;4]
y2=[2.5;3;3;2.5]

x3=[4.5;4.5;6.5;6.5]
y3=[3;5.5;5.5;3]

fill(x1,y1,'y',x2,y2,'y',x3,y3,'y')
```

Les obstacles :



C- La construction de la matrice d'optimisation

La position de chaque point i par rapport à tous les autres points va nous permettre de crée une matrice d'optimisation qui va aider notre programme a trouver le chemin le plus court pour arriver à son objectif

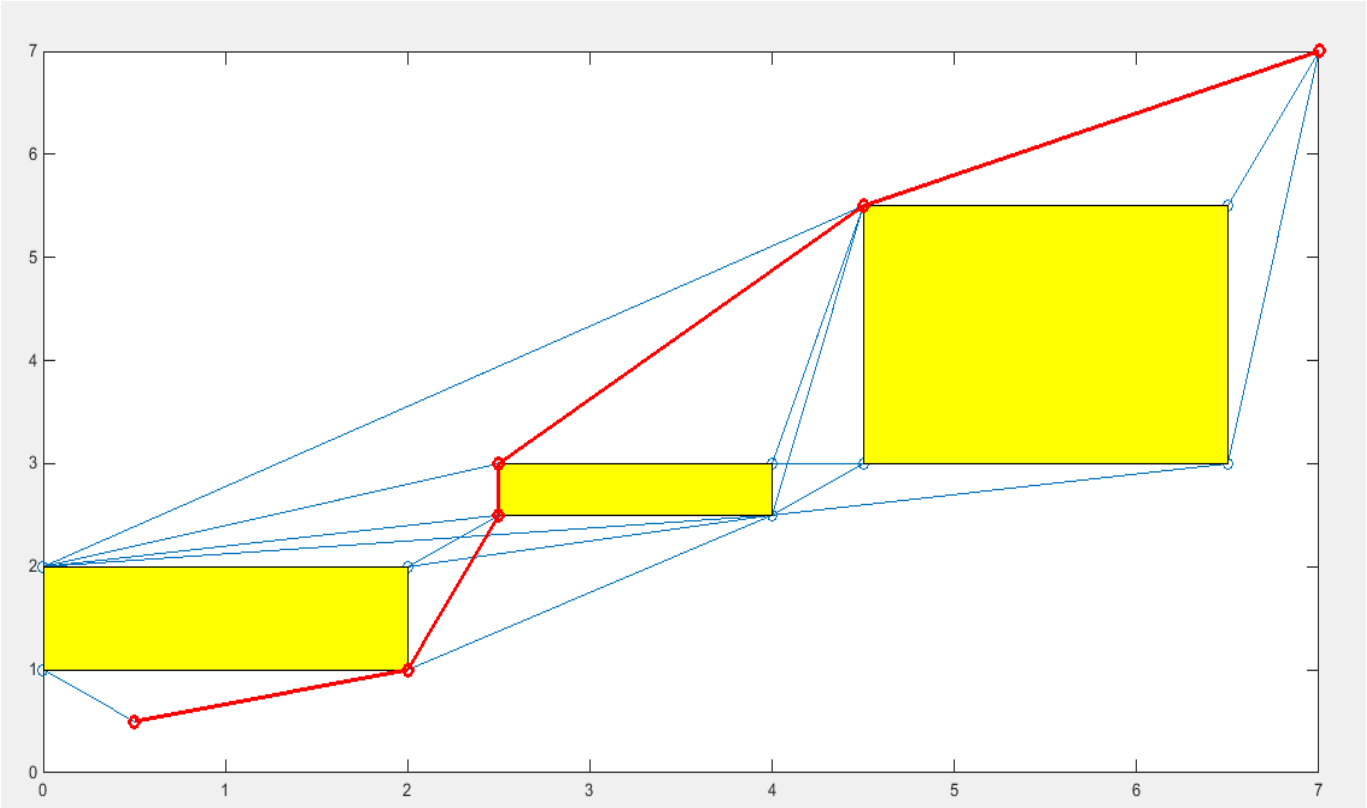
Par exemple si un point X a une vision sur un autre point Y alors l'élément qui se trouve dans ligne X,colonne Y = l'élément qui se trouve dans ligne Y,colonne X = 1

On traite tous les points et on trouve la Matrice A :

```
A = [1 1 0 0 1 0 0 0 0 0 0 0 0 0;
     1 1 1 0 1 0 0 0 0 0 0 0 0 0;
     0 1 1 1 0 1 1 0 1 0 1 0 0 0;
     0 0 1 1 1 1 0 0 1 0 0 0 0 0;
     1 1 0 1 1 1 0 0 0 0 0 0 0 0;
     0 0 1 1 1 1 1 0 0 0 0 0 0 0;
     0 0 1 0 0 1 1 1 0 1 1 0 1 0;
     0 0 0 0 0 0 1 1 1 1 1 0 1 0;
     0 0 1 1 1 1 0 1 1 1 1 0 1 0;
     0 0 0 0 0 0 1 1 1 1 1 0 1 0;
     0 0 1 0 0 0 1 1 1 1 1 1 0 1;
     0 0 0 0 0 0 0 0 0 0 1 1 1 1;
     0 0 0 0 0 0 1 1 1 1 0 1 1 1;
     0 0 0 0 0 0 0 0 0 0 1 1 1 1];
```

C'est une matrice carré d'ordre n = 14 (14 points).

Et finalement on lance le programme :



On voit en rouge la trajectoire entre un point de départ de cordonné (0.5 0.5) et un point d'arrivé de cordonné (7 7).

4- CONCLUSION :

On conclure par une comparaison entre les deux algorithmes, on voit tout d'abord que pour les deux méthodes la trajectoire peut toucher les sommets des obstacles, c'est pour ça qu'on est obliger de crée des obstacles plus grandes que les réels et pour cela on doit prend on considération la taille (rayon) de notre robot pour qu'on évite qu'il touche les obstacles lorsqu'il vas tourner à gauche ou à droite.au niveau du rapidité on trouve que l'algorithme de dijikstra et beaucoup plus rapide que le PSO mais on doit lui donner la matrice, d'autre part Dijkstra sert à résoudre le problème du plus court chemin alors qu'on peut utiliser le pso pour déférents taches, mais pour le control des robots mobiles les deux méthodes sont très efficaces