Flottes de robots - Intégration

Exercice 1 :

1.1) Proposez une condition permettant de passer du mode « consensus » au mode « leader-follower » lorsque les robots sont « suffisamment » proches les uns des autres.

Une condition possible pour passer du mode « consensus » au mode « leader-follower » lorsque les robots sont « suffisamment » proches les uns des autres serait de définir une distance minimale à laquelle les robots doivent être les uns des autres pour que le mode « leader-follower » soit activé. Nous comparerons cette distance à un seuil définit , lorsque les robots sont en dessous de ce seuil, un leader est sélectionné et les autres robots deviennent des followers.

1.2) Créez un fichier MultiRobotsMission.py. En reprenant les codes développés dans les deux premiers TPs et en implémentant la condition de la question 1.1, réaliser le code permettant de réaliser la mission demandée (regroupement des robots, puis suivi de trajectoire en formation).

Etape 1 - Estimer la distance :

Dans un premier temps il nous faut définir la distance entre chaque paire de robot dans la flotte :

```
14  def calculate_distance(fleet):
15
16      distance = np.zeros((fleet.nbOfRobots, fleet.nbOfRobots))
17      for i in range(fleet.nbOfRobots):
18          for j in range(i+1, fleet.nbOfRobots):
19              distance[i][j] = np.linalg.norm(fleet.robot[i].state - fleet.robot[j].state)
20              # fill the matrix symmetrically
21              distance[j][i] = distance[i][j]
22
23      return distance
```

cette fonction **"calculate_distance(fleet)"** calcule la distance entre chaque paire de robots dans la flotte (fleet) et retourne une matrice de distances symétrique de taille (nbOfRobots, nbOfRobots).

Ensuite, la fonction utilise une double boucle "for" pour parcourir chaque paire de robots dans la flotte. La boucle extérieure parcourt chaque robot dans la flotte, tandis que la boucle intérieure parcourt tous les robots qui suivent le robot en question. Cela nous permet d'éviter de calculer deux fois la distance entre chaque paire de robots.

Afin de calculer la distance entre deux robots, nous utilisons la fonction **« np.linalg.norm() »** qui calcule la norme euclidienne entre les états des deux robots. La distance est ensuite stockée dans la matrice de distances à l'emplacement correspondant.

Résultats :

Nous observons la matrice de distance suivante :

```
distances are
 [[0.         3.         3.         3.         3.         6.        ]
 [3.         0.         4.24264069 4.24264069 6.         9.        ]
 [3.         4.24264069 0.         6.         4.24264069 6.70820393]
 [3.         4.24264069 6.         0.         4.24264069 6.70820393]
 [3.         6.         4.24264069 4.24264069 0.         3.        ]
 [6.         9.         6.70820393 6.70820393 3.         0.        ]]
```

Matrice de distances entre chaque paire de robots

Etape 2 – Affecter les conditions de changement du mode de contrôle :

Initialisation :

```
55    rRef = [np.array([[d*0], [d*0]]),
56            np.array([[d*0], [d*1]]),
57            np.array([[d*-1], [d*0]]),
58            np.array([[d*1], [d*0]]),
59            np.array([[d*0], [d*-1]]),
60            np.array([[d*0], [d*-2]])]
```

```
69    # gains for leader (L) and follower (F) robots
70    kL = 1.0
71    kF = 6.0
72    kt = 8.0
73
74    # distance threshold for switching to leader-follower mode
75    threshold_distance = 2
76    consensus_mode = True
77
```

Dans un premier temps, nous avons initialiser les conditions de la manière suivante :

```
88        distances = calculate_distance(fleet)
89        if np.all (distances >= threshold_distance and consensus_mode):
90           # switch back to consensus mode
91           consensus_mode = False
92           print("Switching back to consensus mode at t={}".format(t))
93
94
95        if np.any (distances < threshold_distance) and not consensus_mode:
96             # switch to leader-follower mode
97             consensus_mode = True
98             print("Switching to leader-follower mode at t={}".format(t))
```

- La méthode .any() renvoie True si au moins un élément du tableau vérifie la condition, sinon elle renvoie False.
- La méthode .all() renvoie True si tous les éléments du tableau vérifient la condition, sinon elle renvoie False.

Seulement un problème survenait, en effet la diagonal de la matrice des distance valait 0 par conséquent la condition **« if np.all (distances >= threshold_distance and consensus_mode) »** ne pouvait se réaliser que si le **« treshold_distance » valait 0.** Donc le mode consensus n'était jamais sélectionné avec la mise en place d'un seuil.

Nous avons donc mis en place une manière de ne pas prendre en compte la diagonale de la matrice comme suit :

```
80          # Replace all diagonal values of the 'distances' matrix with infinite values
81          distances = calculate_distance(fleet)
82          diagonal_indices = np.diag_indices_from(distances)
83          np.fill_diagonal(distances, np.inf)
```

Dans la boucle principale de la simulation, la distance entre chaque paire de robots est calculée à l'aide de la fonction **"calculate_distance"** comme vu précédemment. Ensuite, la diagonale de la matrice de distance est remplie avec des valeurs **"inf"** pour éviter de prendre en compte la distance entre un robot et lui-même.

Plus précisément :

- Premièrement, on appelle la fonction **"calculate_distance(fleet)"** qui calcule les distances entre chaque paire de robots et stocke ces distances dans une matrice. La matrice de distances est ensuite stockée dans la variable "distances".

- Ensuite, nous utilisons la fonction **"np.diag_indices_from(distances)"** pour récupérer les indices de la diagonale de la matrice "distances". La diagonale contient les distances entre un robot et lui-même, qui sont toutes égales à zéro. Ces indices sont stockés dans la variable **"diagonal_indices"**

- Enfin, la fonction **"np.fill_diagonal()"** permet de remplacer toutes les valeurs diagonales de la matrice "distances" par des valeurs infinies.

Nous avons alors désormais les conditions suivantes :

```
85      if np.all(distances[diagonal_indices] == np.inf) and np.all(distances >= threshold_distance) and consensus_mode:
86          # switch back to consensus mode
87          consensus_mode = False
88          print("Switching back to consensus mode at t={}".format(t))
89
90
91      if np.any (distances < threshold_distance) and not consensus_mode:
92              # switch to leader-follower mode
93              consensus_mode = True
94              print("Switching to leader-follower mode at t={}".format(t))
```

```
100     for i in range(0, fleet.nbOfRobots):
101
102         if consensus_mode:
103             if i == 0:  # leader
104                 # r = np.array([ [np.sin(2 * np.pi * t)],[2 * t]])
105                 fleet.robot[i].ctrl = kL * (WPListInit[i] - fleet.robot[i].state)
106             else:  # followers
107                 fleet.robot[i].ctrl = kF * (fleet.robot[0].state + (rRef[i] - rRef[0]) - fleet.robot[i].state)
108         else:
109                 fleet.robot[i].ctrl = np.zeros((2,1))
110                 if i == 0:
111                     fleet.robot[i].ctrl = kt * (WPListInit[i] - fleet.robot[i].state)
112
113                 for j in range(fleet.nbOfRobots):
114                     # check if the other robot is close enough
115                         fleet.robot[i].ctrl += kF * (fleet.robot[j].state - fleet.robot[i].state) / fleet.nbOfRobots
```

Dans la boucle principale de la simulation, le code vérifie si toutes les distances entre les robots sont supérieures ou égales au seuil de distance et que le mode de contrôle est le mode "consensus". Si tel est le cas, le code reste sur le mode "consensus". Cela est vérifié à l'aide de la fonction **"np.all(distances >= threshold_distance)".**

Si au moins une paire de robots se trouve à une distance inférieure au seuil de distance et que le mode de contrôle est le mode "leader-follower", le code reste sur le mode "leader_follower". Cela est vérifié à l'aide de la fonction **"np.any (distances < threshold_distance)".**

Si la flotte passe du mode "consensus" au mode "leader-follower", les robots suivent un leader en utilisant la commande :

"fleet.robot[i].ctrl = kF * (fleet.robot[0].state + (rRef[i] - rRef[0]) - fleet.robot[i].state)" pour les robots suiveurs et :

"fleet.robot[i].ctrl = kL * (WPListInit[i] - fleet.robot[i].state)" pour le robot leader.

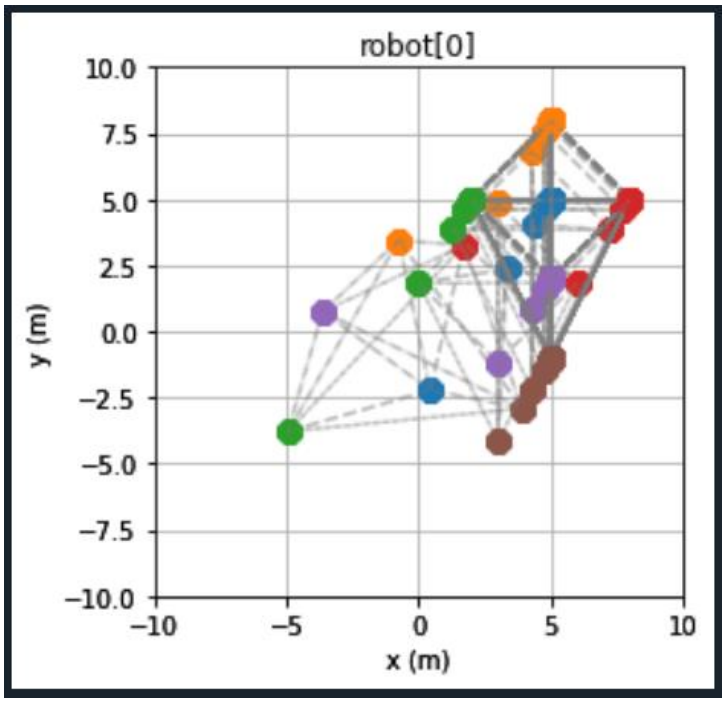
Si la flotte passe du mode "leader-follower" au mode "consensus", chaque robot suit la moyenne des positions des autres robots, en utilisant la commande

"fleet.robot[i].ctrl += kF * (fleet.robot[j].state - fleet.robot[i].state) / fleet.nbOfRobots"

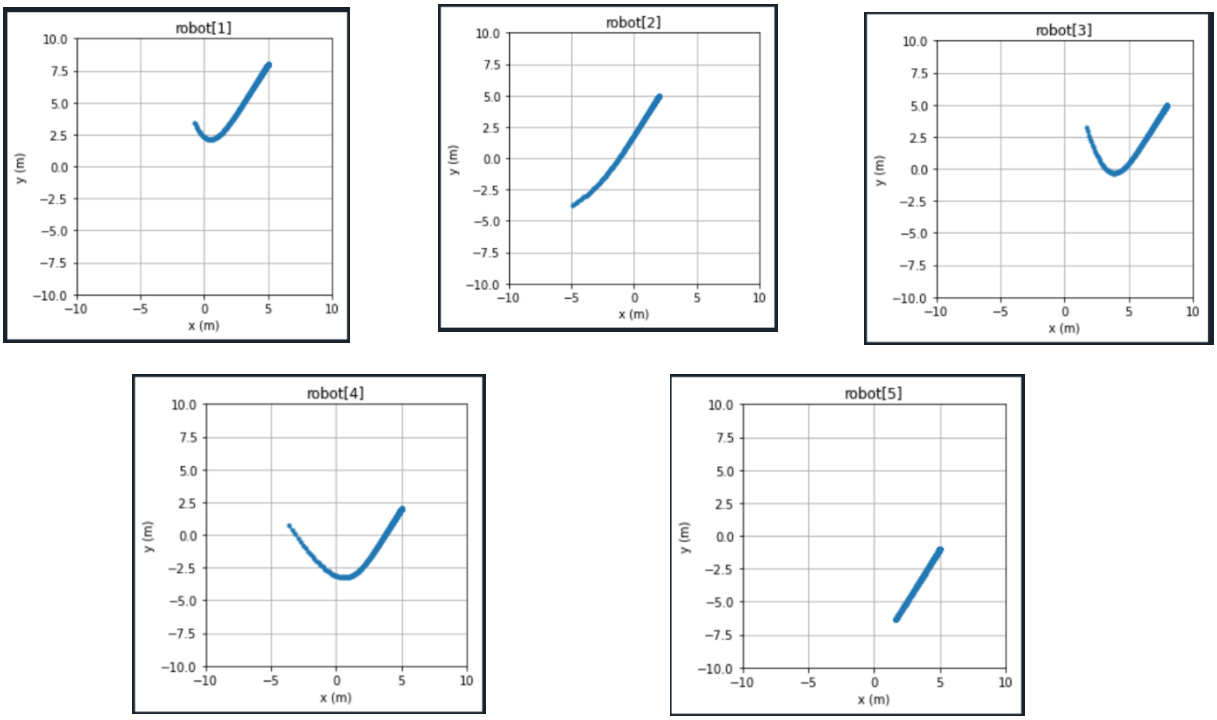
pour chaque paire de robots.

Résultats :

Cas 1 : threshold_distance = 3 / WPListInit = [np.array([[5],[5]])]



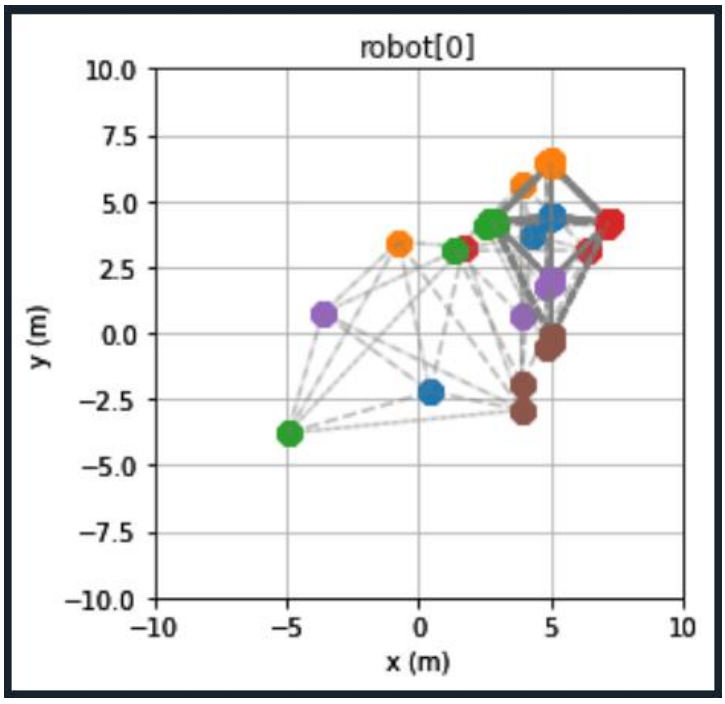
<u>Déplacement de la flotte de robots</u>



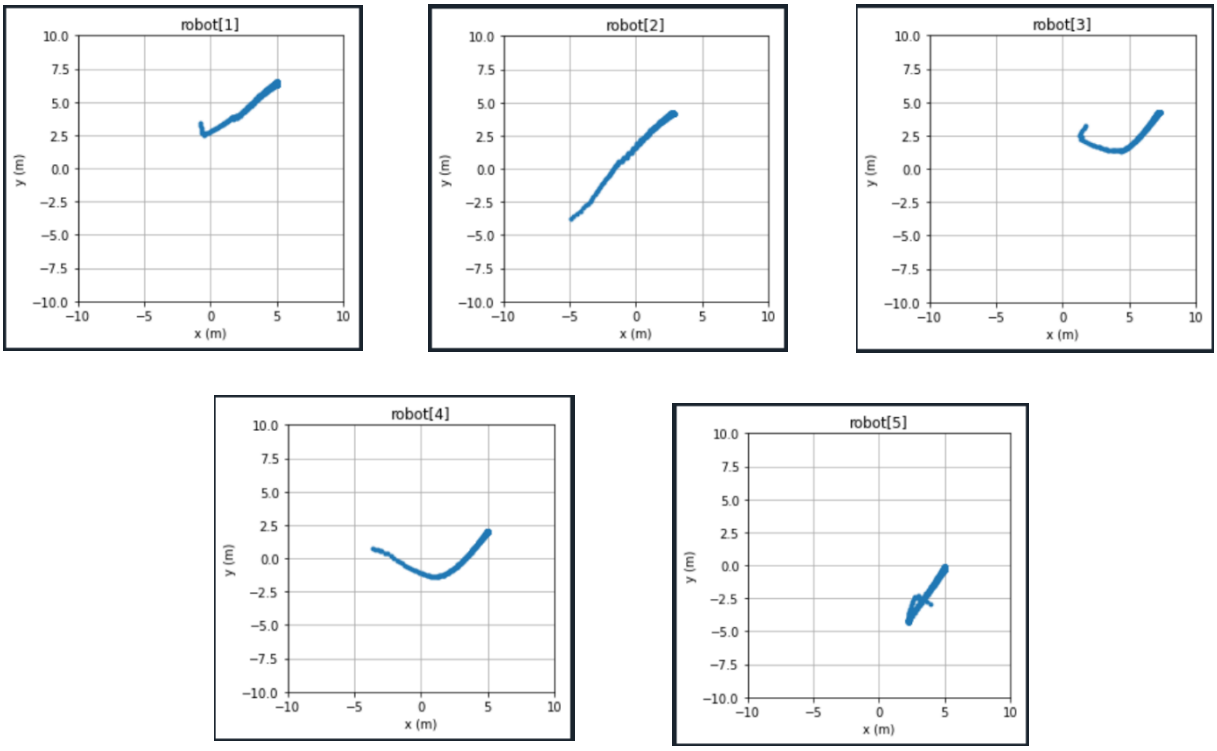
Trajectoires des 5 robots followers au cours du temps

Nous observons qu'avec un seuil = 3, il y a au moins une paire de robots qui a une distance inférieur à 3 d'unités. Par conséquent le flotte se dirige vers le point de cordonnée (5 ; 5) en mode leader follower.

Cas 2 : threshold_distance = 2 / WPListInit = [np.array([[5],[5]])]



Déplacement de la flotte de robots



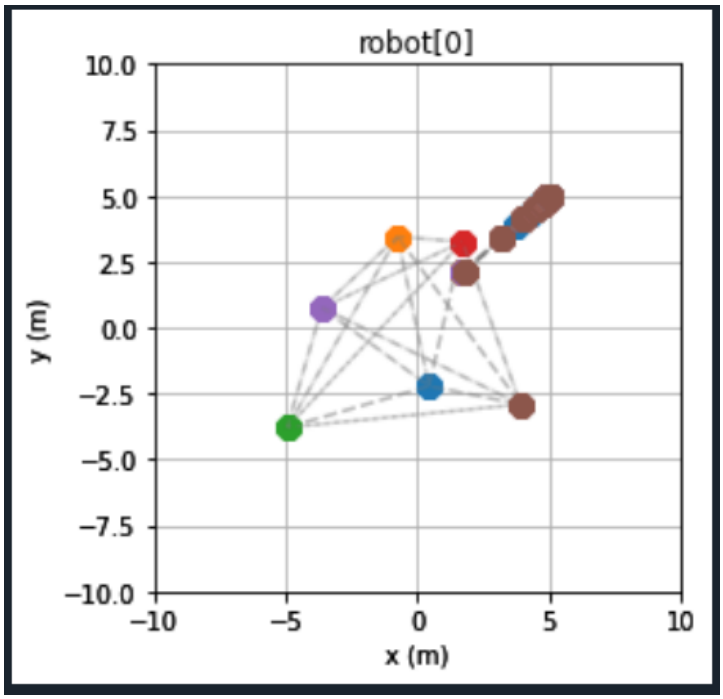
Trajectoires des 5 robots de la flotte au cours du temps

```
Console 1/A ×
Switching to leader-follower mode at t=10.66
Switching back to consensus mode at t=10.69
Switching to leader-follower mode at t=10.700000000000001
Switching back to consensus mode at t=10.73
Switching to leader-follower mode at t=10.74
Switching back to consensus mode at t=10.77
Switching to leader-follower mode at t=10.78
Switching back to consensus mode at t=10.8
Switching to leader-follower mode at t=10.81
Switching back to consensus mode at t=10.84
Switching to leader-follower mode at t=10.85
Switching back to consensus mode at t=10.88
Switching to leader-follower mode at t=10.89
Switching back to consensus mode at t=10.92
Switching to leader-follower mode at t=10.93
Switching back to consensus mode at t=10.950000000000001
```

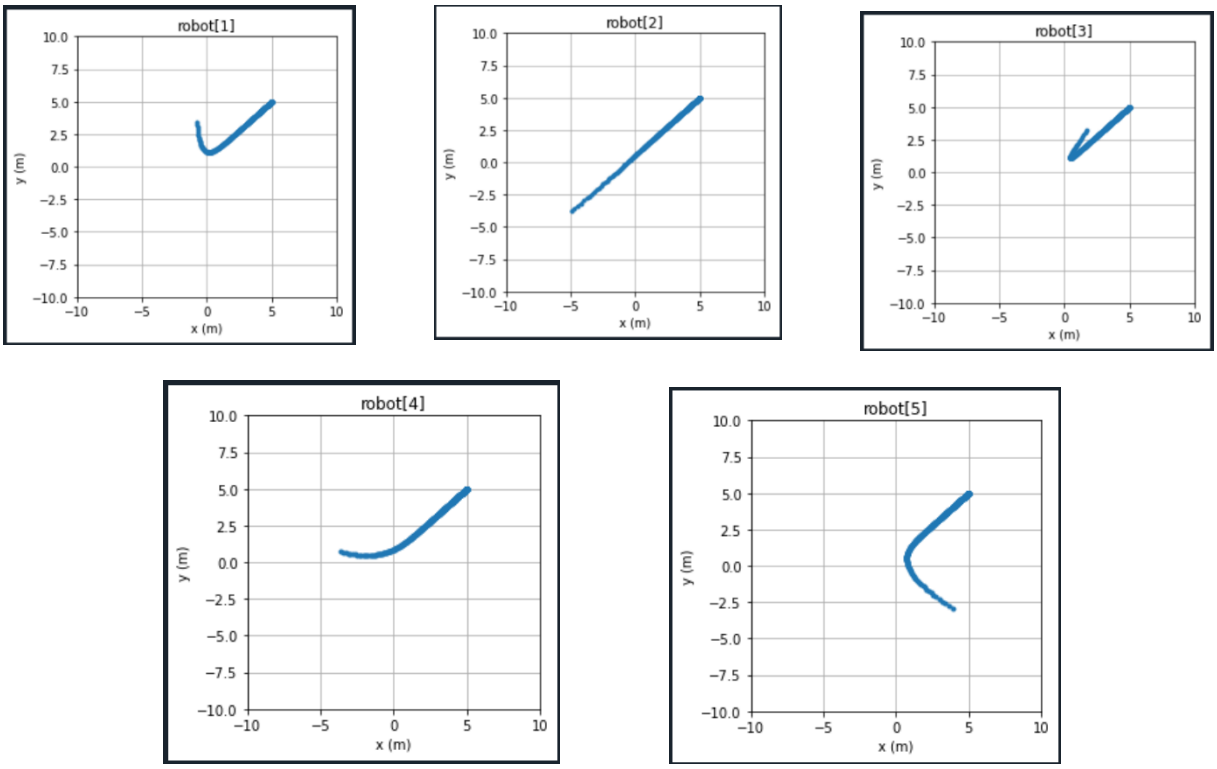
```
Switching to leader-follower mode at t=0.5
[[       inf 2.13439404 4.82723472 1.97343366 4.53062844 7.362107  ]
 [2.13439404        inf 4.4798015  4.0987675  5.80031424 8.84262712]
 [4.82723472 4.4798015         inf 6.05584361 3.85178807 6.51211271]
 [1.97343366 4.0987675  6.05584361        inf 4.32443922 6.58108477]
 [4.53062844 5.80031424 3.85178807 4.32443922        inf 3.07531567]
 [7.362107   8.84262712 6.51211271 6.58108477 3.07531567        inf]]
Switching back to consensus mode at t=0.52
[[       inf 2.13137082 4.69399046 2.01458695 4.43987367 7.29454333]
 [2.13137082        inf 4.45200695 4.1153889  5.82327359 8.86046207]
 [4.69399046 4.45200695        inf 6.04777318 3.89634711 6.53282706]
 [2.01458695 4.1153889  6.04777318        inf 4.31474174 6.59521356]
 [4.43987367 5.82327359 3.89634711 4.31474174        inf 3.06303048]
 [7.29454333 8.86046207 6.53282706 6.59521356 3.06303048        inf]]
```

Nous observons qu'avec un seuil = 2, il y a au moins une paire de robots qui a une distance inférieur à 2 d'unités (passage en leader – follower) au cours du temps, mais il y a également toutes les paires de robots qui ont une distance supérieur à 2 d'unités (passage en consensus) au cours du temps. Par conséquent le flotte se dirige vers le point de cordonnée (5; 5) en alternant mode leader follower et mode consensus.

Cas 3 : threshold_distance = 0 / WPListInit = [np.array([[5],[5]])]



Déplacement de la flotte de robots



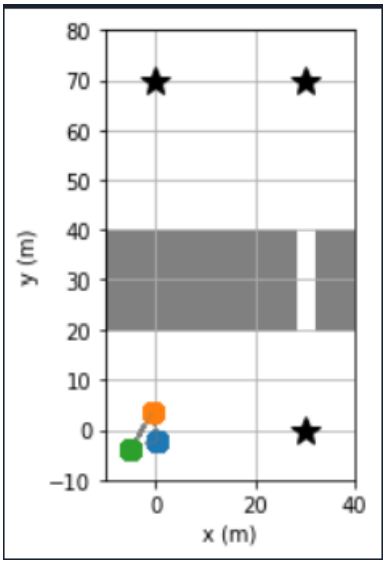
Trajectoires des 5 robots de la flotte au cours du temps

Nous observons que n'aillant pas de seuil définit, la flotte de robot reste dans le cas où toutes les paires de robots ont une distance supérieur au seuil. Par conséquent la flotte se déplace à point de coordonnées (5,5) en mode consensus.

Exercice 2:

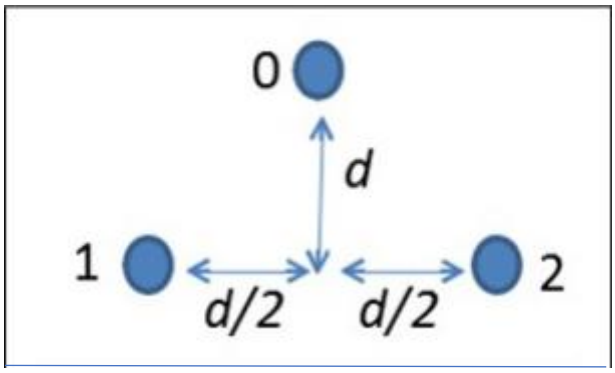
Il s'agit ici de commander une flotte de trois robots pour réaliser la mission composée des trois objectifs suivants:

- **atteindre successivement les trois points de passage (étoiles sur la figure page suivante)**
- **tenir compte de la présence d'obstacles (pas de trajectoire possible à travers les rectangles gris de la figure ci-dessous)**
- **maintenir le plus possible les robots dans la formation triangle définie sur la figure de la page suivante (avec d = 6m).**



Etape 1 - atteindre successivement les trois points de passage & maintenir le plus possible les robots dans la formation triangle :

Comme dans l'exercice 1 , nous avons paramétré la position de référence des robots dans l'environnement, mais cette fois -ci pour une flotte de 3 robots afin d'obtenir les positions de références suivantes :




```
41  d = 6
42  rRef = [np.array([[d*0], [d]]),
43          np.array([[d/2], [d*0]]),
44          np.array([[d/-2], [d*0]])]
45
```

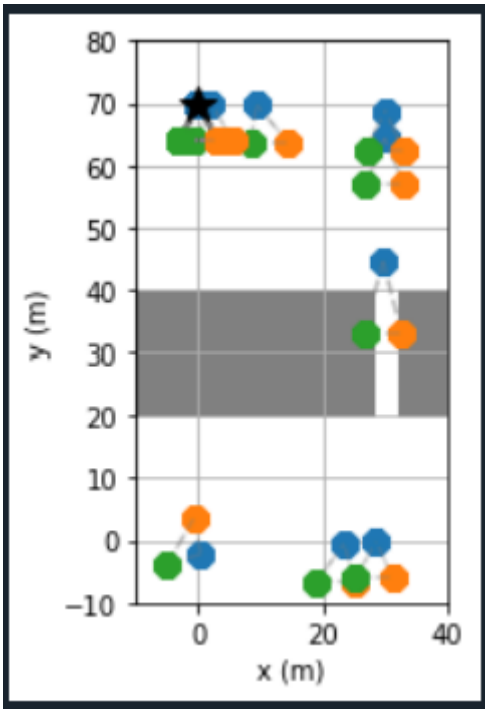
Puis suite à cela nous avons mis en place la commande de leader follower pour atteindre les points de passage comme suit :

```
94          for i in range(0, fleet.nbOfRobots):
95              r = WPListInit[0]
96
97              if i == 0:  # leader
98                  fleet.robot[i].ctrl = kL * (r - fleet.robot[i].state)
99
100             else:  # followers
101                 fleet.robot[i].ctrl = kF * (fleet.robot[0].state + (rRef[i] - rRef[0]) - fleet.robot[i].state)
102
103
104               #update waypoint if the robot is close enough
105             if np.linalg.norm(fleet.robot[i].state - r)  < 0.9:
106                 if len( WPListInit) > 1 :
107                     del WPListInit[0]
```

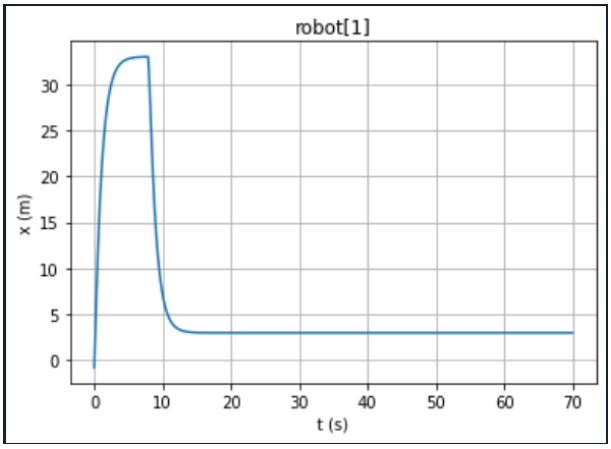
On vérifie si le robot est suffisamment proche de son waypoint actuel (r) en calculant la norme de la différence entre la position actuelle du robot et le waypoint. Si la distance est inférieure à 0,9, le code supprime le premier waypoint de la liste des waypoints (WPListInit) pour passer au waypoint suivant.

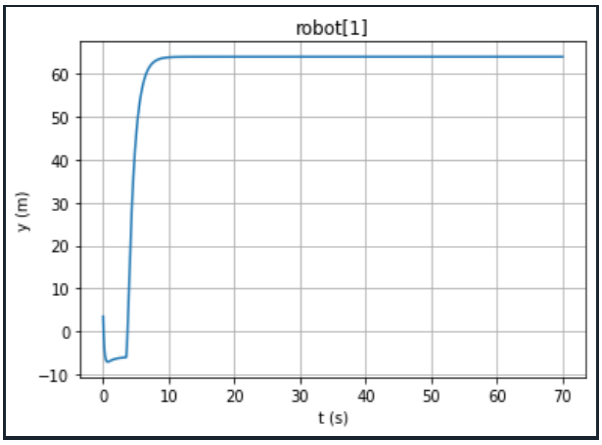
Ce code permet à chaque robot de se déplacer vers son waypoint en utilisant un contrôle leader-follower où le leader guide les followers vers leur destination en temps réel.

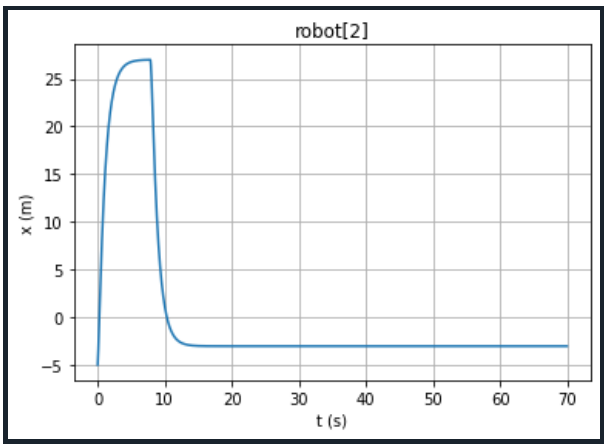
Résultats :

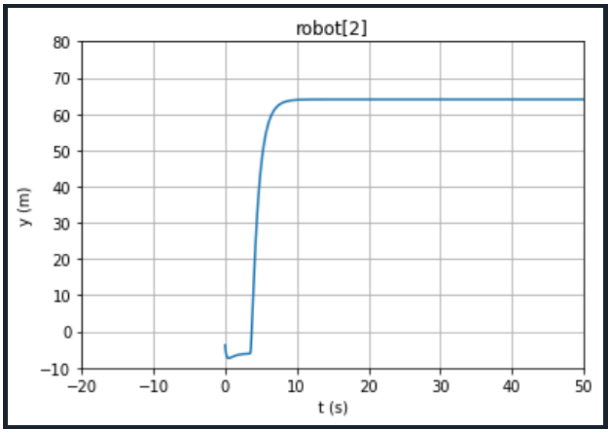


<u>Déplacement de la flotte vers les waypoints sans tenir compte des obstacles</u>









<u>Trajectoires des 2 robots followers au cours du temps sur les axes (x,y)</u>

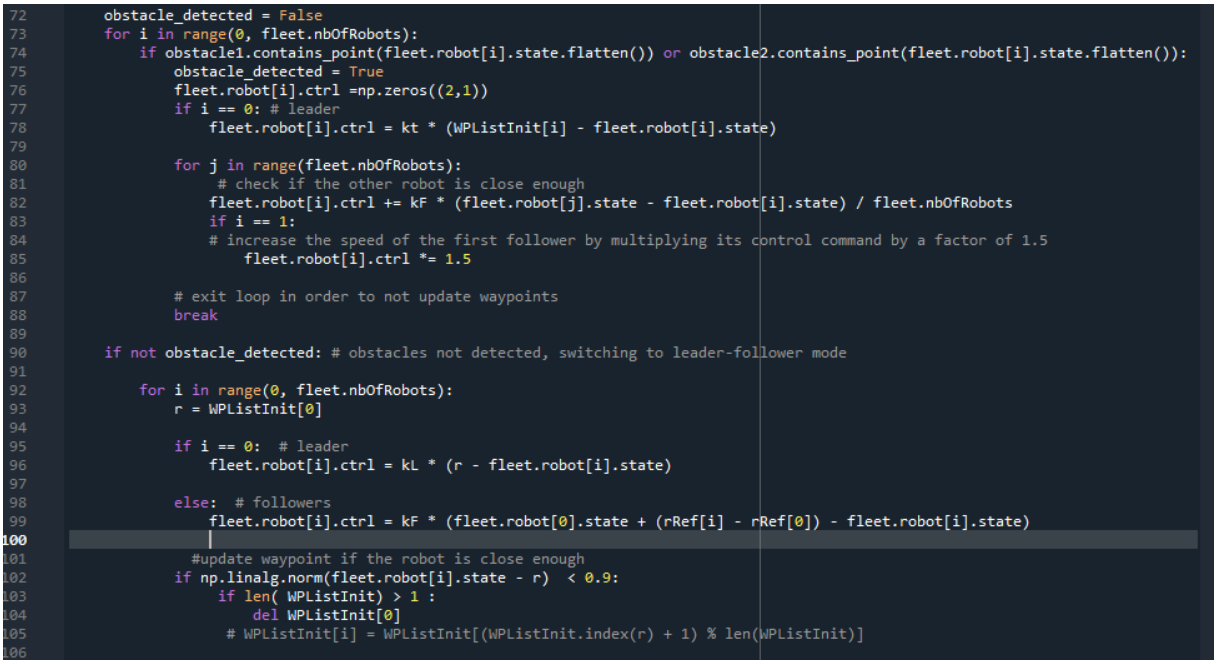
Etape 2- Evitement d'obstacles:

Durant cette étape, il nous a fallu trouver les conditions pour lesquels les robots évitent de rentrer en collision avec les obstacles. Pour se faire, nous avons utilisé la méthode suivante :

```
72      obstacle_detected = False
73      for i in range(0, fleet.nbOfRobots):
74          if obstacle1.contains_point(fleet.robot[i].state.flatten()) or obstacle2.contains_point(fleet.robot[i].state.flatten()):
75              obstacle_detected = True
```

La condition **"if obstacle1.contains_point(fleet.robot[i].state.flatten()) or obstacle2.contains_point(fleet.robot[i].state.flatten()):"** vérifie si l'état actuel du robot **(fleet.robot[i].state)** est à l'intérieur de l'un des deux polygones qui représentent les obstacles (obstacle1 et obstacle2). Si c'est le cas, cela signifie que le robot est en collision avec un obstacle et doit être évité. Dans ce cas, il est possible de définir une stratégie d'évitement d'obstacles.

Dans notre cas notre stratégie est de passer en mode consensus , si toutefois un obstacle est détecté :



```
    obstacle_detected = False
    for i in range(0, fleet.nbOfRobots):
        if obstacle1.contains_point(fleet.robot[i].state.flatten()) or obstacle2.contains_point(fleet.robot[i].state.flatten()):
            obstacle_detected = True
            fleet.robot[i].ctrl =np.zeros((2,1))
            if i == 0: # leader
                fleet.robot[i].ctrl = kt * (WPListInit[i] - fleet.robot[i].state)

            for j in range(fleet.nbOfRobots):
                 # check if the other robot is close enough
                fleet.robot[i].ctrl += kF * (fleet.robot[j].state - fleet.robot[i].state) / fleet.nbOfRobots
                if i == 1:
                # increase the speed of the first follower by multiplying its control command by a factor of 1.5
                    fleet.robot[i].ctrl *= 1.5

            # exit loop in order to not update waypoints
            break

    if not obstacle_detected: # obstacles not detected, switching to leader-follower mode

        for i in range(0, fleet.nbOfRobots):
            r = WPListInit[0]

            if i == 0:  # leader
                fleet.robot[i].ctrl = kL * (r - fleet.robot[i].state)

            else:  # followers
                fleet.robot[i].ctrl = kF * (fleet.robot[0].state + (rRef[i] - rRef[0]) - fleet.robot[i].state)

              #update waypoint if the robot is close enough
            if np.linalg.norm(fleet.robot[i].state - r)  < 0.9:
                 if len( WPListInit) > 1 :
                     del WPListInit[0]
                  # WPListInit[i] = WPListInit[(WPListInit.index(r) + 1) % len(WPListInit)]
```

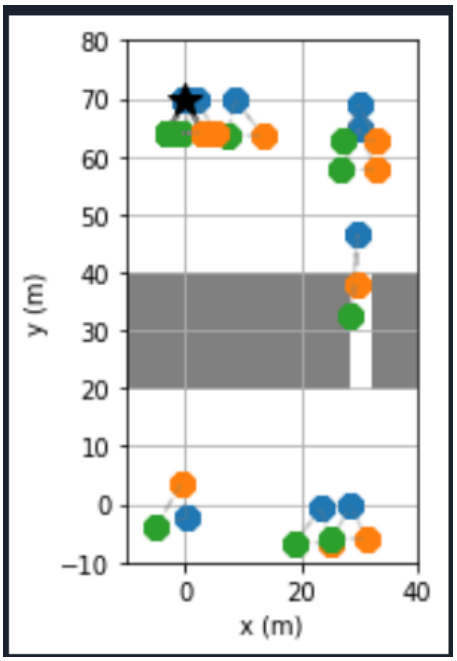
Si un robot est détecté en collision avec l'un des obstacles, son signal de commande est mis à zéro pour le stopper. Si le robot est le leader, son signal de commande est mis à jour pour le faire aller vers le premier waypoint.

Si aucun robot n'est détecté en collision avec les obstacles, le code passe en mode leader-follower. Le signal de commande du leader est mis à jour pour le faire aller vers le premier waypoint de la liste de waypoints WPListInit, car en effet à chaque waypoints atteints il est supprimé de la liste. Les signaux de commande des suiveurs sont mis à jour pour les faire suivre le leader. Nous avons rajouté un coefficient sur la commande de l'un des robots followers, afin qu'il aille plus que l'autre follower pour pouvoir passer de manière aligné à travers l'obstacle.

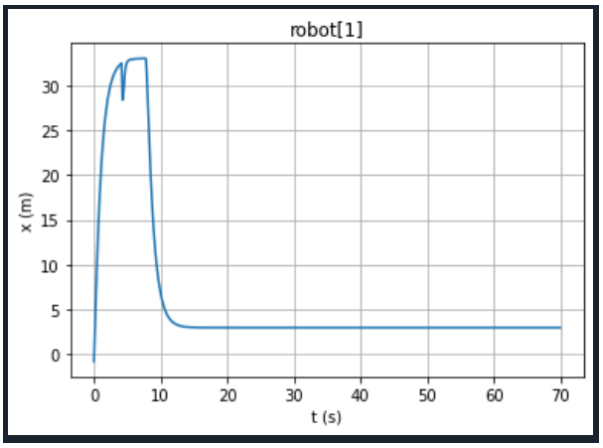
De plus nous avons rajouté un « Break » entre les deux conditions de ce fait, si un obstacle est détecté par l'un des robots, l'ensemble de la flotte s'arrête et ne met pas à jour ses waypoints. Le "Break" est utilisé pour sortir de la boucle plus rapidement car il n'est pas nécessaire de vérifier si les autres robots ont également détecté l'obstacle, car l'ensemble de la flotte doit s'arrêter dès qu'un obstacle est détecté.

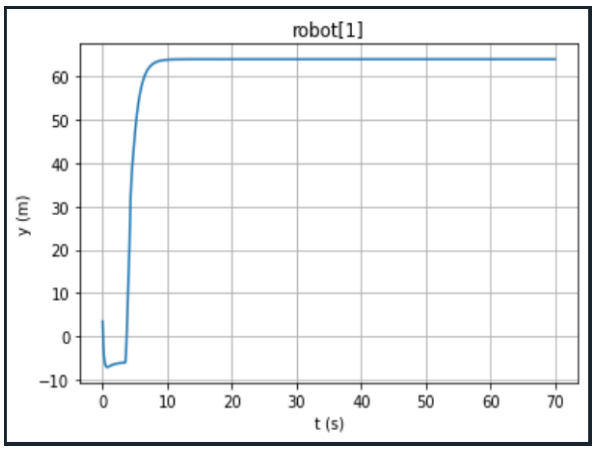
Enfin, pour chaque robot, le code vérifie s'il est suffisamment proche du waypoint actuel. Si c'est le cas, le waypoint actuel est supprimé de la liste et le prochain est pris.

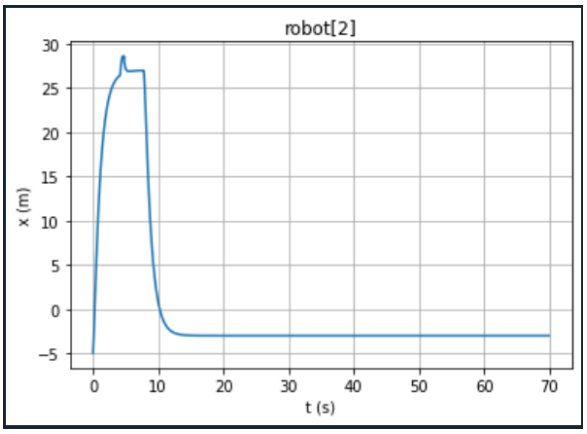
Résultats :

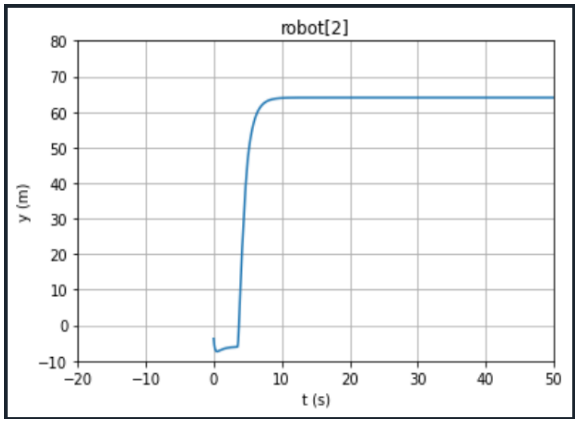


Déplacement de la flotte vers les waypoints en tenant compte des obstacles









Trajectoires des 2 robots followers au cours du temps sur les axes (x,y)

Nous observons bien que la flotte démarre bien en mode leader follower et en formation (triangle) , suite à cela une fois un obstacle détecté, la flotte pas en mode consensus avec le leader en tête de flotte. Une fois l'obstacle passé la flotte revient en mode leader-follower et continue d'atteindre les waypoints restants .