

MASTER PROFESSIONNEL RAIA

TRAVAUX PRATIQUES

Atelier Robots Mobiles

TPN°3 : Robot Suiveur de Mur dans un Labyrinthe

Date : ----- **Classe :** ----- **Durée : 3h**

	<i>Nom & Prénom :</i>	<i>AB/PR</i>	<i>Mot/Part</i>	<i>TP N°</i>	<i>Total</i>
1	-----		/10	/10	/20
2	-----		/10	/10	/20
3	-----		/10	/10	/20

Objectifs du TP :

- ✓ -----
- ✓ -----
- ✓ -----

Conditions de réalisation et moyens :

- ✓ -----
- ✓ -----
- ✓ -----

Objectifs :

- Concevoir, implémenter et tester un algorithme de navigation autonome pour un robot mobile dans un labyrinthe inconnu, en utilisant des techniques de suivi de mur.
- Comprendre les principes de base de la navigation autonome en robotique mobile, en particulier les algorithmes de suivi de mur.

I- Introduction :

Un labyrinthe [9][10] est un ensemble de chemins ou d'itinéraires entrelacés, qui sont conçus de manière à créer une structure complexe et déroutante qui peut être difficile à naviguer. Les labyrinthes peuvent être construits en utilisant des murs, des haies, des miroirs ou d'autres obstacles pour créer des passages et des culs-de-sac. Il existe de nombreuses applications de l'utilisation de labyrinthes dans le domaine de la robotique. En voici quelques exemples :

- ✓ Les robots de nettoyage utilisent des labyrinthes pour naviguer de manière autonome dans des espaces complexes comme les maisons, les bureaux, les hôpitaux, les aéroports et les usines.
- ✓ Les robots de tonte de pelouse peuvent utiliser des labyrinthes pour naviguer autour des obstacles tels que les arbres et les bordures de pelouse tout en évitant les zones interdites.
- ✓ Les drones peuvent utiliser des labyrinthes pour effectuer des missions de surveillance et d'exploration dans des environnements difficiles à atteindre, tels que les zones de catastrophe, les sites industriels et les zones de guerre.
- ✓ Les robots de déminage peuvent utiliser des labyrinthes pour détecter et désamorcer des explosifs dans des zones dangereuses sans mettre en danger les soldats.
- ✓ Les robots de recherche et de sauvetage peuvent utiliser des labyrinthes pour trouver des survivants dans des environnements dangereux comme les décombres des tremblements de terre et des catastrophes naturelles.
- ✓ Les robots de livraison peuvent utiliser des labyrinthes pour planifier leur itinéraire de livraison le plus efficace tout en évitant les obstacles et en optimisant leur consommation d'énergie.
- ✓ Les robots de fabrication peuvent utiliser des labyrinthes pour optimiser leur trajectoire de mouvement et réduire les collisions entre les robots et les équipements de production.
- ✓ Les robots explorateurs peuvent utiliser des labyrinthes pour explorer des environnements extraterrestres tels que Mars et la Lune.

Pour les robots autonomes et la recherche en intelligence artificielle, il existe plusieurs méthodes pour résoudre un labyrinthe de manière autonome et efficace. Voici quelques-unes des plus courantes :

- ✓ Algorithme de recherche de chemin A* [1]: C'est une méthode de recherche de chemin qui utilise une fonction heuristique pour estimer le coût le plus probable pour atteindre la sortie. Cet algorithme peut être utilisé pour résoudre des labyrinthes de manière efficace et trouver le chemin le plus court vers la sortie.

- ✓ Algorithme de Q-Learning [2] : C'est une méthode d'apprentissage par renforcement qui peut être utilisée pour résoudre des labyrinthes de manière autonome. Dans cette méthode, un agent apprend à naviguer dans un labyrinthe en explorant l'environnement et en choisissant des actions qui maximisent sa récompense. Au fil du temps, l'agent apprend à résoudre le labyrinthe en trouvant le chemin le plus court vers la sortie.
- ✓ Réseaux de neurones [3] : Les réseaux de neurones peuvent être utilisés pour apprendre à résoudre des labyrinthes. Les réseaux de neurones convolutifs (CNN) sont souvent utilisés pour traiter les images de labyrinthes et apprendre à naviguer dans l'environnement. Les réseaux de neurones récurrents (RNN) sont également utilisés pour modéliser le comportement des robots autonomes dans des labyrinthes.
- ✓ Algorithme de Monte Carlo [4] : C'est une méthode d'optimisation qui peut être utilisée pour trouver le chemin le plus court vers la sortie d'un labyrinthe. Dans cette méthode, l'algorithme simule plusieurs trajectoires aléatoires pour trouver le chemin optimal. Cet algorithme peut être utilisé pour résoudre des labyrinthes de manière efficace et est souvent utilisé pour résoudre des problèmes d'optimisation complexes.

Certains robots utilisent aussi des algorithmes basés sur l'exploration 3D, comme l'algorithme Rapidly exploring Random Trees (RRT), pour cartographier leur environnement et trouver des chemins optimaux. Ces algorithmes sont plus avancés mais nécessitent des capteurs plus sophistiqués et une puissance de calcul plus élevée pour fonctionner efficacement [7].

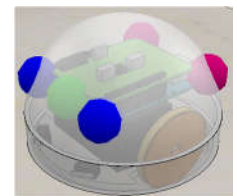
Il existe également des algorithmes plus avancés, tels que l'algorithme de Pledge [8], qui utilise des changements de direction pour naviguer dans un environnement, et l'algorithme de Trémaux, qui utilise une méthode de marquage pour éviter de revenir sur ses pas. Ces algorithmes sont plus complexes mais peuvent être utiles pour naviguer dans des environnements plus complexes [5].

L'un des algorithmes le plus utilisé est l'algorithme de mur (en anglais "wall follower algorithm") est une méthode de résolution de labyrinthes qui consiste pour un robot mobile à suivre les murs du labyrinthe pour trouver la sortie.

L'algorithme de mur droit et gauche [5][6] (en anglais "right-hand rule" et "left-hand rule") est une variante de l'algorithme de mur, qui est une méthode de résolution de labyrinthes pour les robots mobiles.

II- Le robot mobile lumibot suiveur de mur (Wall Follower):

1- Evaluation des différentes distances entre le robot et le mur :

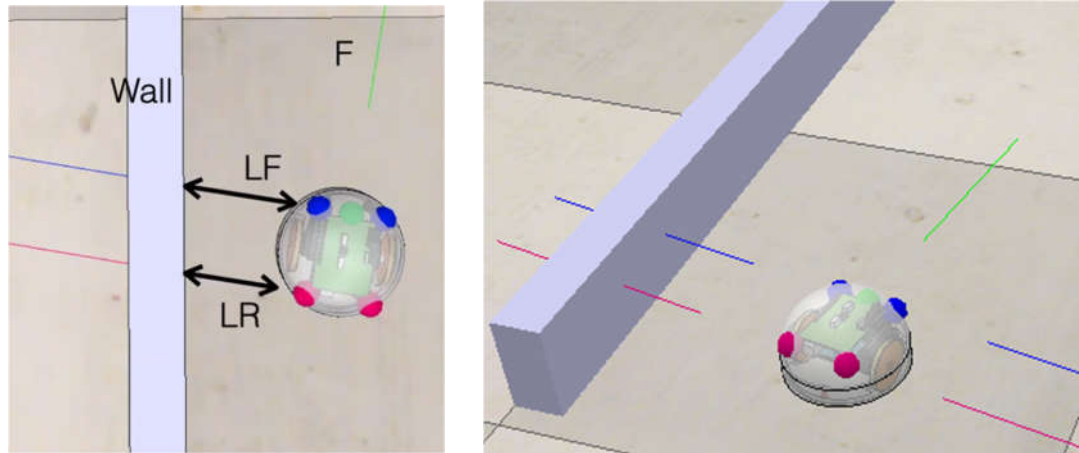


Le robot suiveur de mur utilise cinq capteurs de proximité pour évaluer les distances qui le séparent du mur. Ces capteurs comprennent deux capteurs situés à l'avant droit (RF) et à l'avant gauche (LF), deux autres capteurs placés à l'arrière droit (RR) et à l'arrière gauche (LR), ainsi qu'un capteur situé au milieu à l'avant (F).

Le modèle de contrôle utilisé pour diriger le robot suiveur de mur implique le calcul de la moyenne (average) de la distance, qui est représentée par la variable "avg" égale à 0.5 fois la somme des valeurs des capteurs avant gauche (LF) et arrière gauche (LR). En outre, ce modèle

de contrôle calcule également la différence entre ces deux capteurs, représentée par la variable "diff" égale à la soustraction de LF et LR.

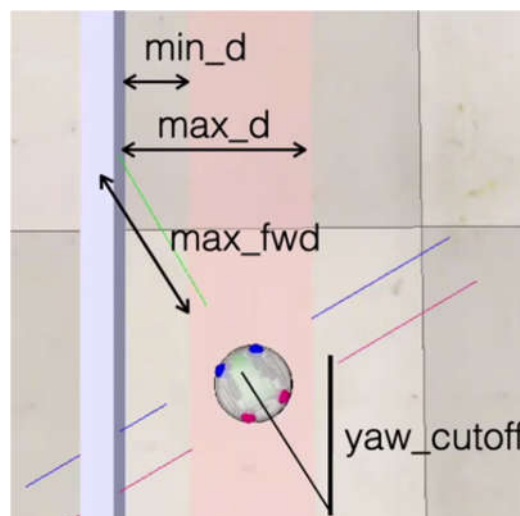
$$\text{avg} = 0.5 * (\text{LF} + \text{LR}) \quad \text{et} \quad \text{diff} = \text{LF} - \text{LR}$$



2- Principe de l'algorithme suiveur de mur à gauche :

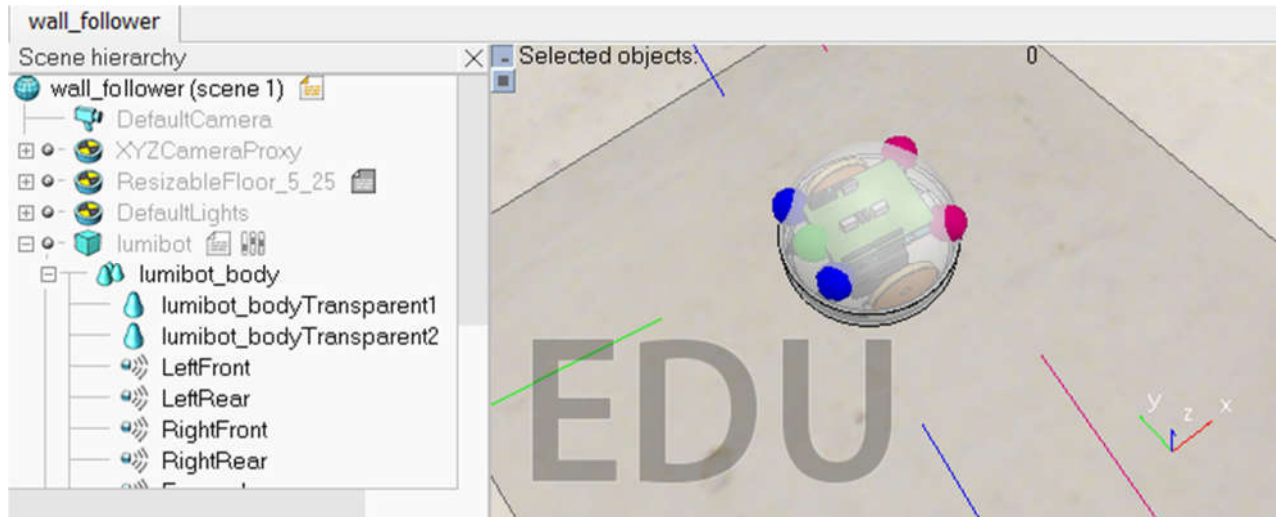
Le principe de l'algorithme suiveur de mur à gauche consiste à programmer le robot pour qu'il longe un mur situé à sa gauche en utilisant ses capteurs de proximité. Lorsque le robot démarre, il avance tout droit jusqu'à ce qu'un de ses capteurs détecte le mur. Ensuite, le robot suit le mur en gardant son capteur avant gauche (LF) et son capteur arrière gauche (LR) à une distance constante du mur. Si le robot détecte un obstacle avec son capteur avant au milieu (F), cela signifie qu'il s'est trop rapproché du mur, il doit alors tourner légèrement vers la droite pour retrouver sa trajectoire. Si le robot détecte un espace vide avec son capteur avant gauche (LF), cela signifie qu'il s'est éloigné du mur, il doit alors tourner légèrement vers la gauche pour se rapprocher du mur à nouveau. De cette façon, le robot suit le mur tout en évitant les obstacles et en restant à une distance constante du mur à sa gauche.

- ✓ la distance moyenne devrait être comprise entre min_d et max_d
- ✓ le yaw (lacet : rotation autour de l'axe Z) doit être compris entre yaw min et max (yaw_cutoff)
- ✓ le robot s'éloigne du mur si le capteur avant (F) détecte max_fwd (fwd_cutoff)

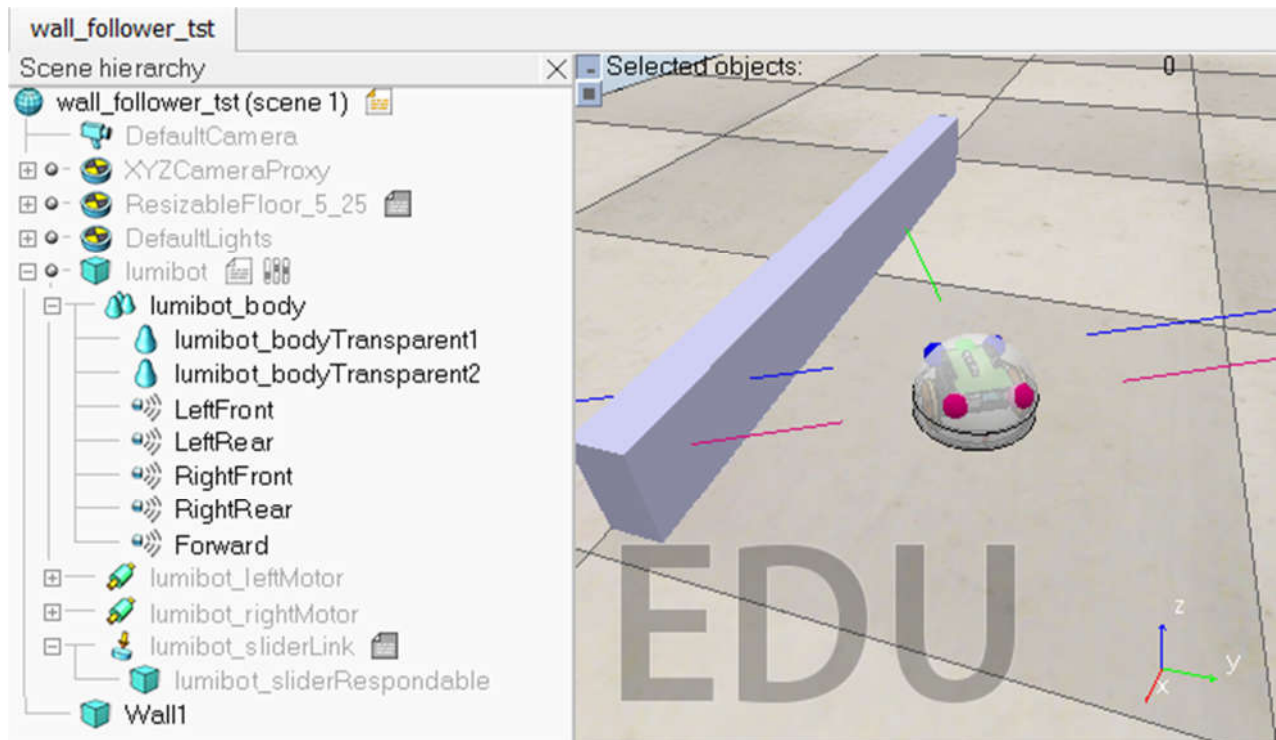


III- Mise en œuvre du robot mobile lumibot suiveur de mur (Wall Follower):

⇒ Lancer CoppeliaSim et ouvrir le fichier



- ⇒ Configurez les propriétés des capteurs ultrasoniques, tels que l'orientation et la distance de détection.
- ⇒ Ajouter un mur dans la scène en utilisant l'option "Add" puis « Cuboid » dans le menu "Primitive Shape".
- ⇒ Renommer « Cuboid » en « Wall1 » ou « Mur1 »
- ⇒ Configurez les propriétés du mur, tels que la taille, la forme et qu'il soit détectable.



⇒ Cliquer sur le script et écrire le programme suivant :

```

1 function sysCall_init()
2
3     lmotor=sim.getObjectHandle("./lumibot_leftMotor")
4     rmotor=sim.getObjectHandle("./lumibot_rightMotor")
5     SensorLF=sim.getObjectHandle('./LeftFront')
6     SensorLR=sim.getObjectHandle('./LeftRear')
7     SensorRF=sim.getObjectHandle('./RightFront')
8     SensorRR=sim.getObjectHandle('./RightRear')
9     SensorF=sim.getObjectHandle('./Forward')
10
11 end
12
13 function sysCall_actuation()
14
15     sim.setJointTargetVelocity(lmotor,1)
16     sim.setJointTargetVelocity(rmotor,1)
17
18 end
19
20 function sysCall_sensing()
21
22     flag1, LF=sim.readProximitySensor(SensorLF)
23     flag2, LR=sim.readProximitySensor(SensorLR)
24     flag3, F=sim.readProximitySensor(SensorF)
25     print ('flag1 = '..flag1..' '..LF = '..LF)
26     print ('flag2 = '..flag2..' '..LR = '..LR)
27     print ('flag3 = '..flag3..' '..F = '..F)
28 end
29
30 function sysCall_cleanup()
31
32 end

```

⇒ Lancer la simulation et interpréter les résultats.

⇒ Cliquer une autre fois sur le script et écrire le programme complet suivant :


```

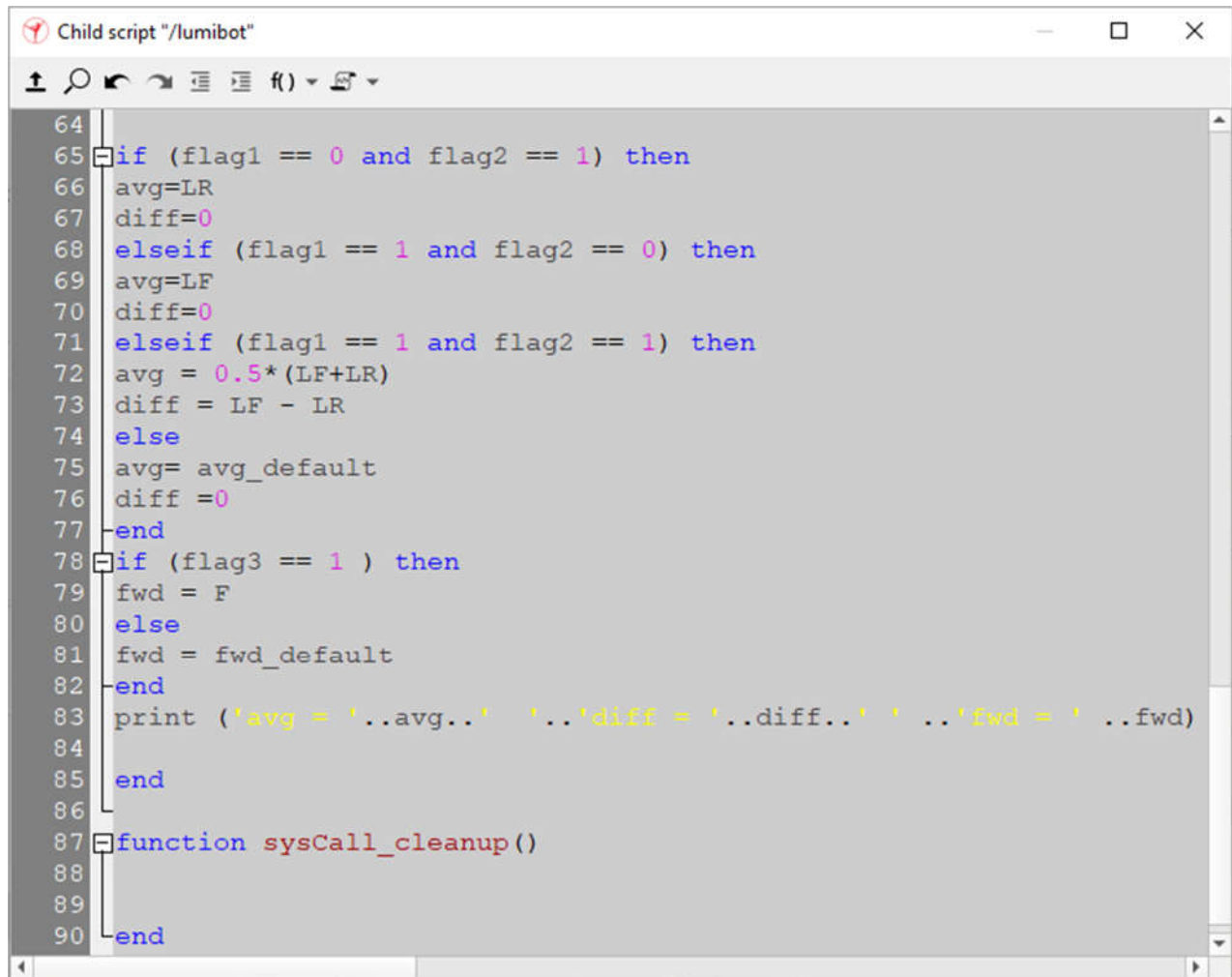
1 function sysCall_init()
2
3     lmotor=sim.getObjectHandle("./lumibot_leftMotor")
4     rmotor=sim.getObjectHandle("./lumibot_rightMotor")
5     SensorLF=sim.getObjectHandle('./LeftFront')
6     SensorLR=sim.getObjectHandle('./LeftRear')
7     SensorRF=sim.getObjectHandle('./RightFront')
8     SensorRR=sim.getObjectHandle('./RightRear')
9     SensorF=sim.getObjectHandle('./Forward')
10
11     min_d = 0.10
12     max_d = 0.15
13     yaw_cutoff = 0
14     fwd_cutoff = 0.15
15     avg_default=0.15
16     fwd_default=50
17     v = 1
18     dv = 1
19     v_sharp = 1
20     avg = avg_default
21     fwd=fwd_default
22     diff=0
23 end
24
25 function sysCall_actuation()
26
27     sim.setJointTargetVelocity(lmotor,v)
28     sim.setJointTargetVelocity(rmotor,v)
29     if (fwd < fwd_cutoff) then
30         print('going away from the wall, turn right')
31         sim.setJointTargetVelocity(lmotor,v_sharp)
32         sim.setJointTargetVelocity(rmotor,0)
33     elseif(fwd > fwd_cutoff) then

```

```

34
35     if (avg > max_d) then
36     print('going away from the wall, turn left')
37     sim.setJointTargetVelocity(lmotor,v-dv)
38     sim.setJointTargetVelocity(rmotor,v)
39
40     elseif (avg < min_d) then
41     print('going away from the wall, turn right')
42     sim.setJointTargetVelocity(lmotor,v)
43     sim.setJointTargetVelocity(rmotor,v-dv)
44
45     elseif (avg > min_d and avg < max_d) then
46     if (diff > yaw_cutoff ) then --c?d LF>LR
47     print('yaw correction, turn left')
48     sim.setJointTargetVelocity(lmotor,v-dv)
49     sim.setJointTargetVelocity(rmotor,v)
50
51     elseif (diff < -yaw_cutoff ) then --c?d LF<LR
52     sim.setJointTargetVelocity(lmotor,v)
53     sim.setJointTargetVelocity(rmotor,v-dv)
54     end
55     end
56 end
57 end
58
59 function sysCall_sensing()
60
61 flag1, LF=sim.readProximitySensor(SensorLF)
62 flag2, LR=sim.readProximitySensor(SensorLR)
63 flag3, F=sim.readProximitySensor(SensorF)

```

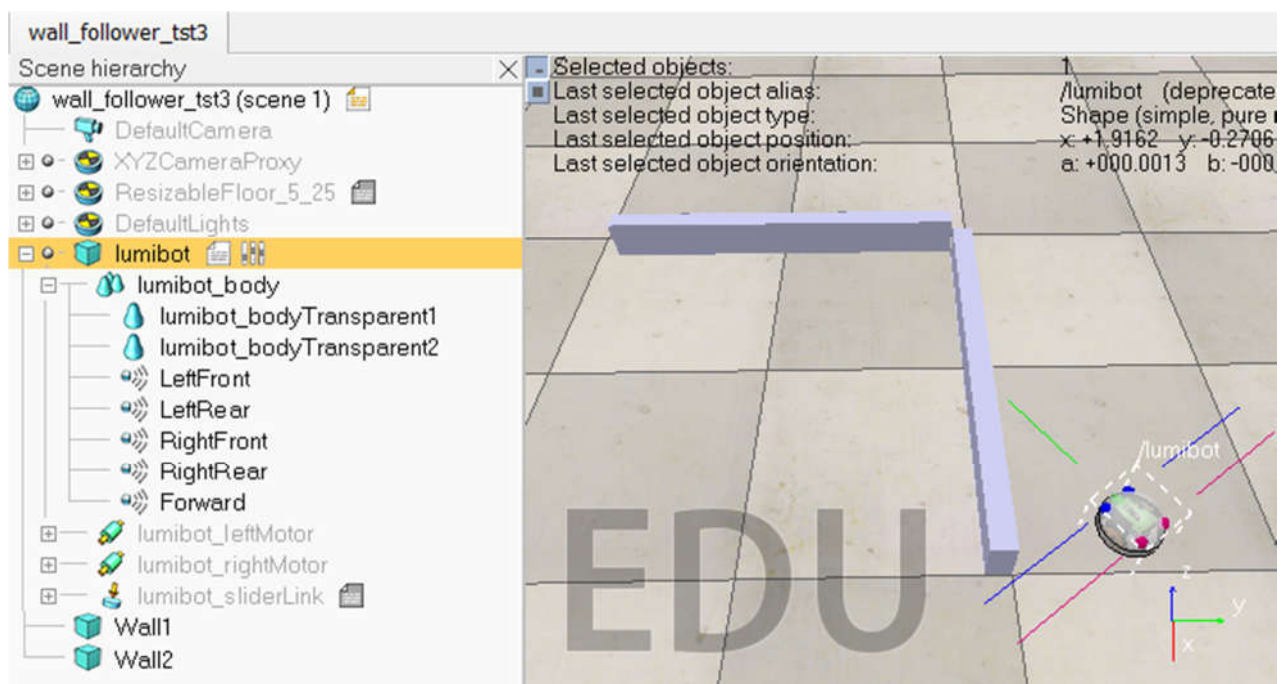
```

Child script "/lumibot"

64
65 if (flag1 == 0 and flag2 == 1) then
66   avg=LR
67   diff=0
68 elseif (flag1 == 1 and flag2 == 0) then
69   avg=LF
70   diff=0
71 elseif (flag1 == 1 and flag2 == 1) then
72   avg = 0.5*(LF+LR)
73   diff = LF - LR
74 else
75   avg= avg_default
76   diff =0
77 end
78 if (flag3 == 1 ) then
79   fwd = F
80 else
81   fwd = fwd_default
82 end
83 print ('avg = '..avg..' ' '..diff = '..diff..' ' '..fwd = ' ..fwd)
84
85 end
86
87 function sysCall_cleanup()
88
89
90 end

```

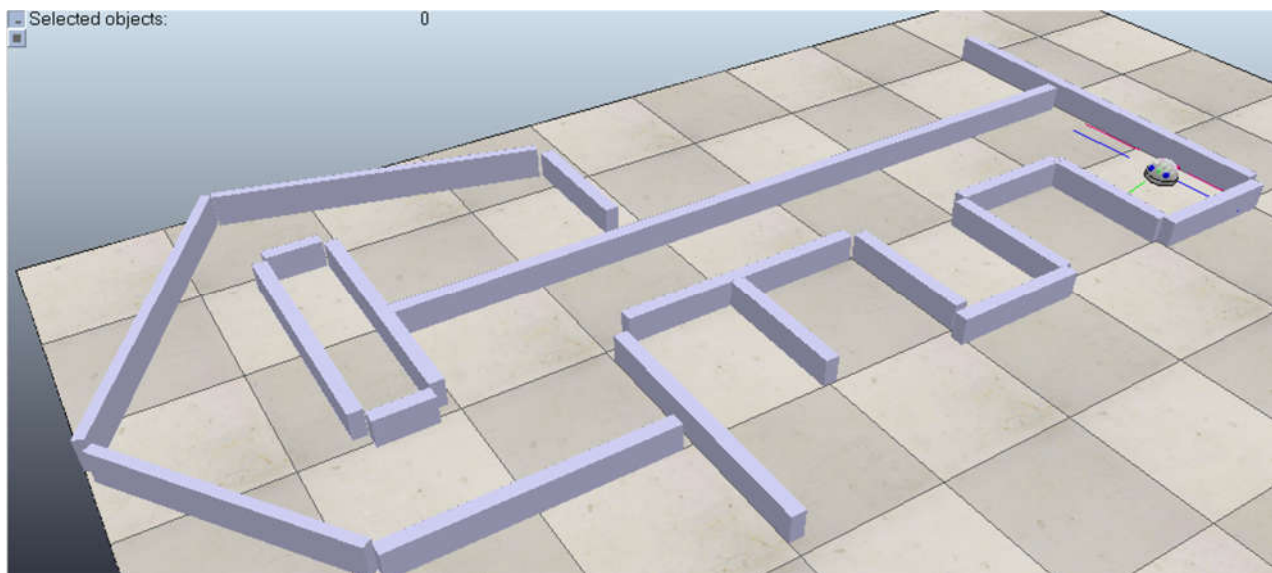
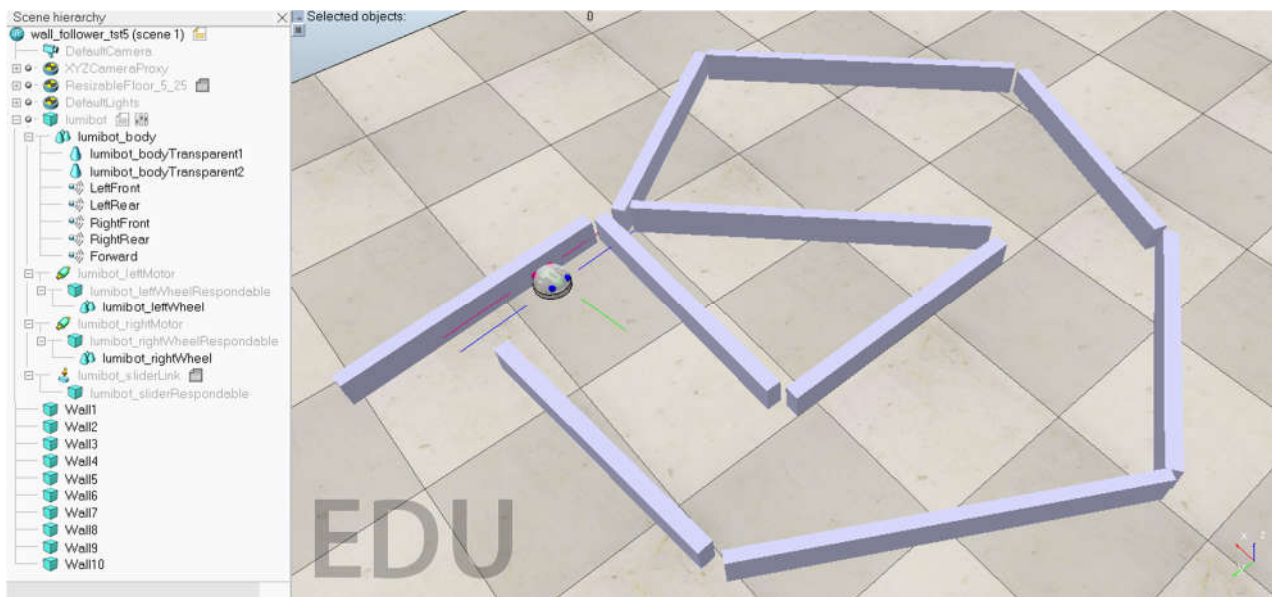
⇒ Ajouter un deuxième mur puis relancer la simulation et interpréter les résultats.



⇒ Avec plusieurs murs construire un labyrinthe.

⇒ Relancer la simulation et interpréter les résultats.

⇒ Exemples de Labyrinthe.



- (1) "A* search algorithm", Wikipédia : https://en.wikipedia.org/wiki/A*_search_algorithm
- (2) "Q-learning", Wikipédia : <https://en.wikipedia.org/wiki/Q-learning>
- (3) "Deep Reinforcement Learning for Robot Navigation in a Large Environment", IEEE Robotics and Automation Letters, 2019 :
<https://ieeexplore.ieee.org/abstract/document/8697506>
- (4) "Monte Carlo Methods in Artificial Intelligence", Springer Science & Business Media, 2006 : <https://link.springer.com/book/10.1007/978-3-540-33236-7>
- (5) https://pedagogie.ac-montpellier.fr/sites/default/files/ressources/Grain%20d%27usage%20Labyrinthe%20SAM_S_0.pdf
- (6) <https://hal.science/hal-00765934/document>
- (7) <https://core.ac.uk/download/pdf/213621765.pdf>
- (8) <https://interstices.info/lalgorithme-de-pledge/>
- (9) Labyrinthes et représentations mathématiques :
https://fr.wikipedia.org/wiki/Mod%C3%A9lisation_math%C3%A9matique_de_labyrinthe
- (10) Générateur de labyrinthes automatiques : <http://www.mazegenerator.net/>
permet de générer des labyrinthes quelconques.