

MASTER PROFESSIONNEL RAIA

TRAVAUX PRATIQUES

Atelier Robots Mobiles

TPN°2 : Concevoir et programmer un robot suiveur de ligne avec CoppeliaSim Edu et Lua script.

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 et programmer, dans CoppeliaSim Edu, un robot capable de suivre une ligne noire sur le sol en utilisant des capteurs.
- Ajuster les paramètres tels que la vitesse et la distance entre les capteurs pour optimiser les performances du robot, en utilisant le script Lua et les fonctions Regular API.
- Utiliser des algorithmes plus avancés pour contrôler le mouvement du robot.

I- Introduction :

CoppeliaSim Edu (anciennement V-REP Edu, Virtual Robot Experimentation Platform) [1] est un logiciel de simulation robotique 3D qui permet de créer et de tester des scénarios de simulation pour différents types de robots et de systèmes.

Lua et Python sont deux langages de programmation intégrés à CoppeliaSim Edu, qui peuvent être utilisés pour personnaliser et automatiser les simulations. Voici quelques exemples d'utilisation de Lua et Python dans CoppeliaSim Edu :

Personnalisation de scénarios de simulation : Lua et Python peuvent être utilisés pour créer des scénarios de simulation personnalisés en définissant les propriétés des objets et des robots, en contrôlant les mouvements et les interactions entre les objets, et en définissant les comportements des robots.

Contrôle de robots : Lua et Python peuvent être utilisés pour programmer les mouvements et les actions des robots dans la simulation. Les scripts Lua ou Python peuvent être utilisés pour définir les trajectoires de mouvement, les comportements en cas de collision, les capteurs et les actionneurs.

Analyse de données : Lua et Python peuvent être utilisés pour collecter et analyser des données générées par la simulation. Les scripts Lua ou Python peuvent être utilisés pour extraire des informations telles que la trajectoire des robots, les collisions, les temps de réaction, etc.

Création de plugins : Lua et Python peuvent être utilisés pour créer des plugins pour CoppeliaSim Edu. Les plugins permettent d'ajouter de nouvelles fonctionnalités à CoppeliaSim Edu en utilisant le langage de programmation de son choix.

En résumé, l'utilisation de Lua et Python dans CoppeliaSim Edu permet aux utilisateurs de personnaliser et d'automatiser leurs simulations, ce qui peut aider à accélérer le processus de développement et de test des algorithmes de contrôle robotique et des scénarios de simulation.

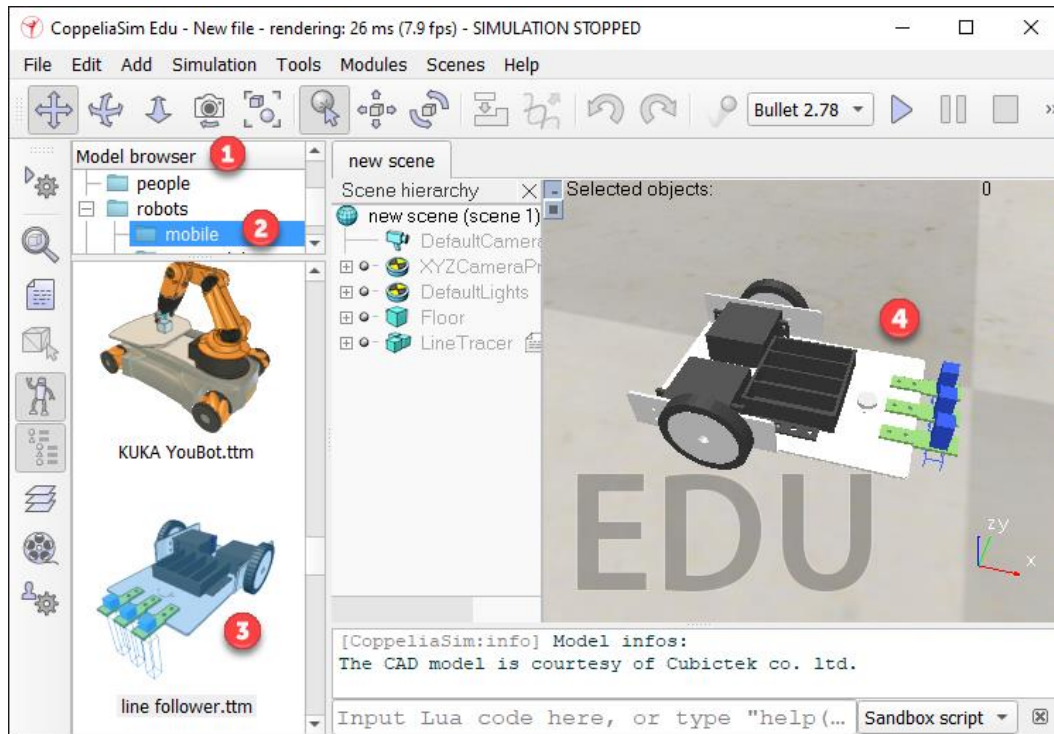
Durant la suite du TP, nous ferons usage à des fonctions de l'API régulière (Regular API functions) [2] pour permettre aux scripts Lua d'interagir avec la simulation et d'effectuer des actions spécifiques selon les besoins du scénario de simulation.

II- Mise en œuvre d'un robot suiveur de ligne :

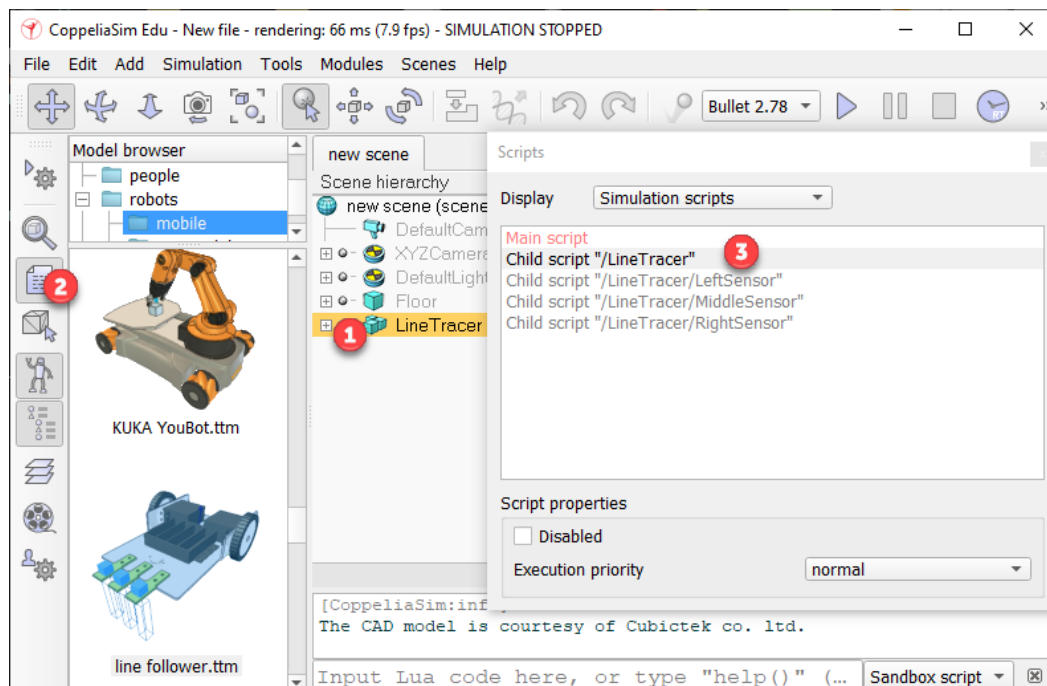
Cliquer sur l'icône CoppeliaSim Edu (ou V-REP PRO EDU).



Dans l'espace du « model browser », répertoire « robots », sous-répertoire « mobile » choisir le robot « line follower.ttm » puis le déposer dans la scène.



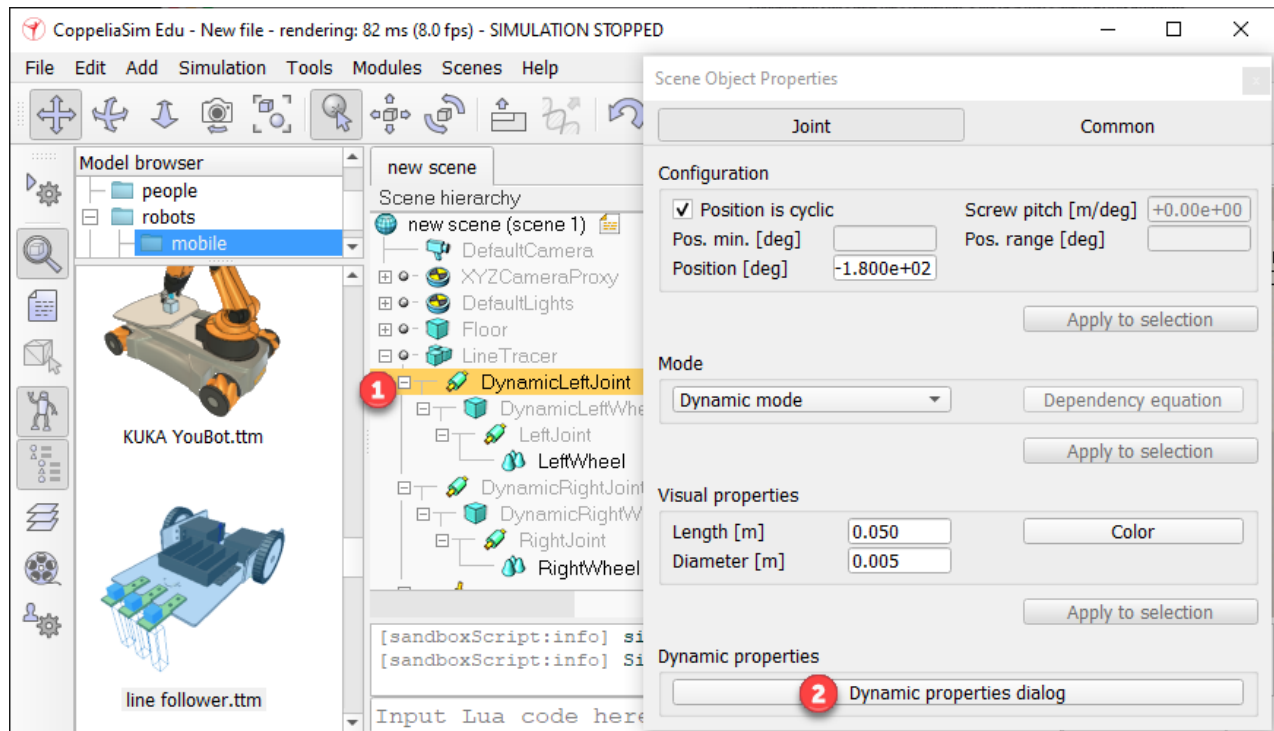
Pour créer ultérieurement votre Script, aller dans « new scene », sélectionner «LineTracer » puis « Scripts ». Quand la fenêtre GUI (Graphical User Interface) « Scripts » s'affiche, pointer sur « Child script "/>



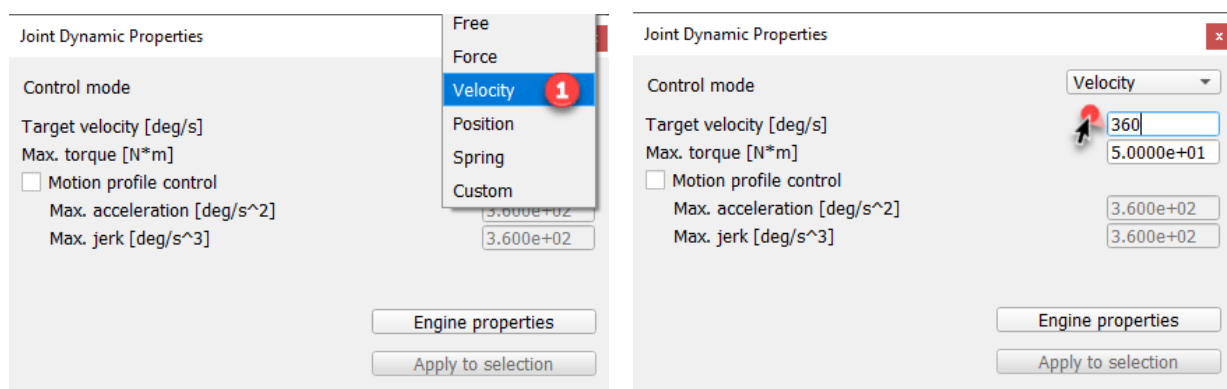
On peut contrôler le robot manuellement (vitesse, position, orientation, etc.) en utilisant les fenêtres GUI (Graphical User Interface).

Exemple :

- ⇒ Aller dans « new scene », sélectionner « LineTracer », Cliquer sur « DynamicLeftJoint », quand la fenêtre « Scene Object Properties » s'affiche cliquer sur « Dynamic properties dialog ».



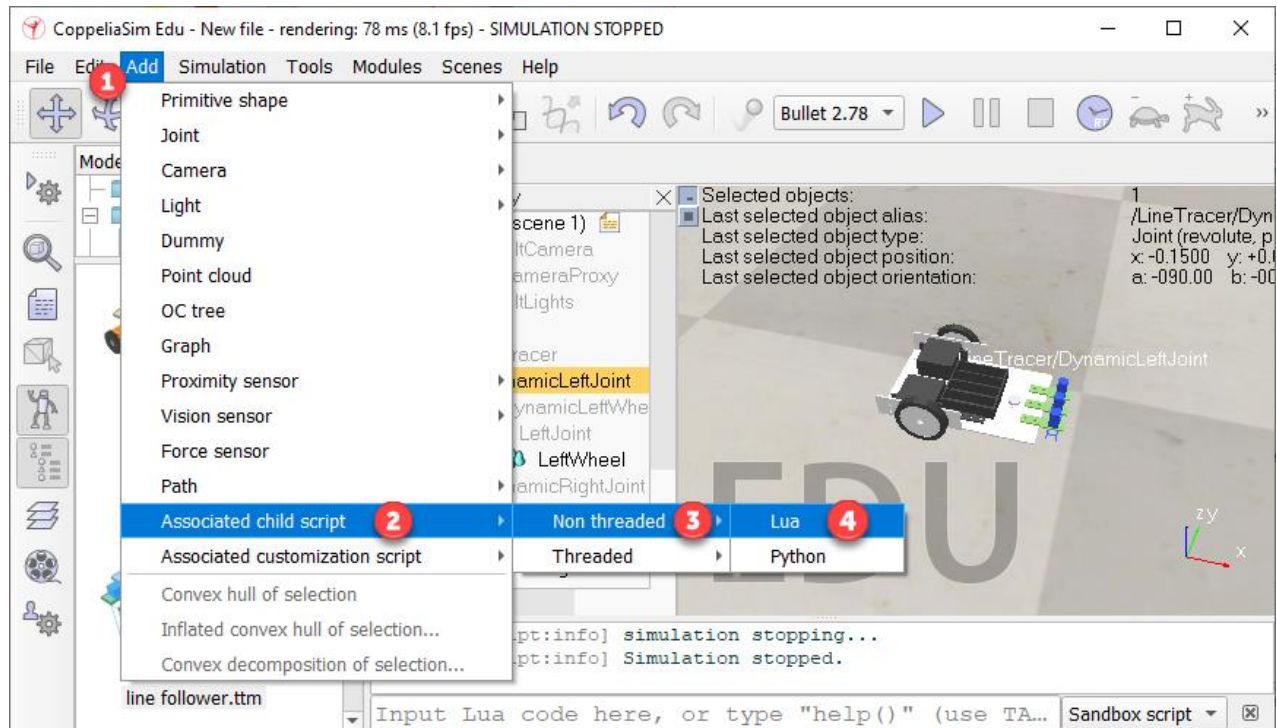
- ⇒ La fenêtre « Joint Dynamic Properties » apparaît et contient les différents contrôles du **moteur droit** du robot. Sélectionner « Velocity » puis dans « Target velocity [deg/s] » taper 360 (vous pouvez essayer d'autres valeurs) et valider.




- ⇒ Cliquer sur « Start/resume simulation »
- ⇒ Visualiser la simulation et interpréter.
- ⇒ Refaire les mêmes étapes pour « DynamicRightJoint »
- ⇒ Enregistrer votre scène : cliquer sur « File », « Save scene as », « CoppeliaSim scene » puis donner un nom. **Exemple :** « *Suiveur de ligne.ttt* »

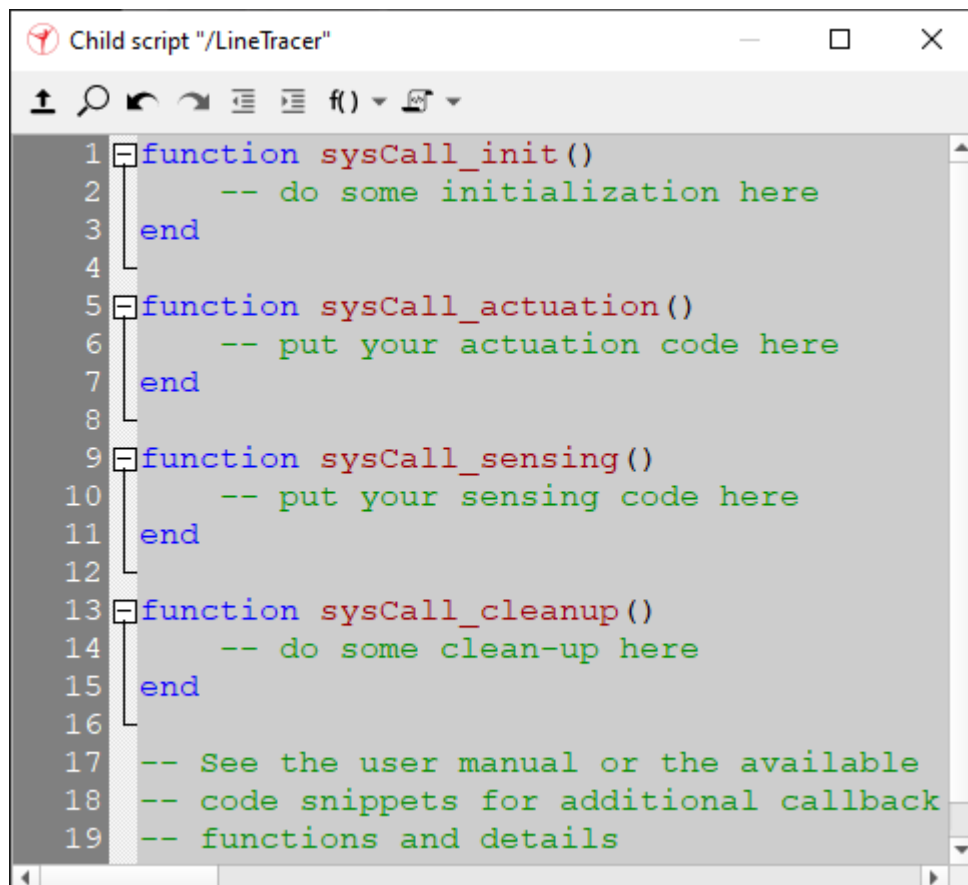


Dans la scene « Suiveur de ligne », sélectionner «LineTracer », cliquer sur «Add », sélectionner « Associated child script », ensuite « Non threaded » puis « Lua »



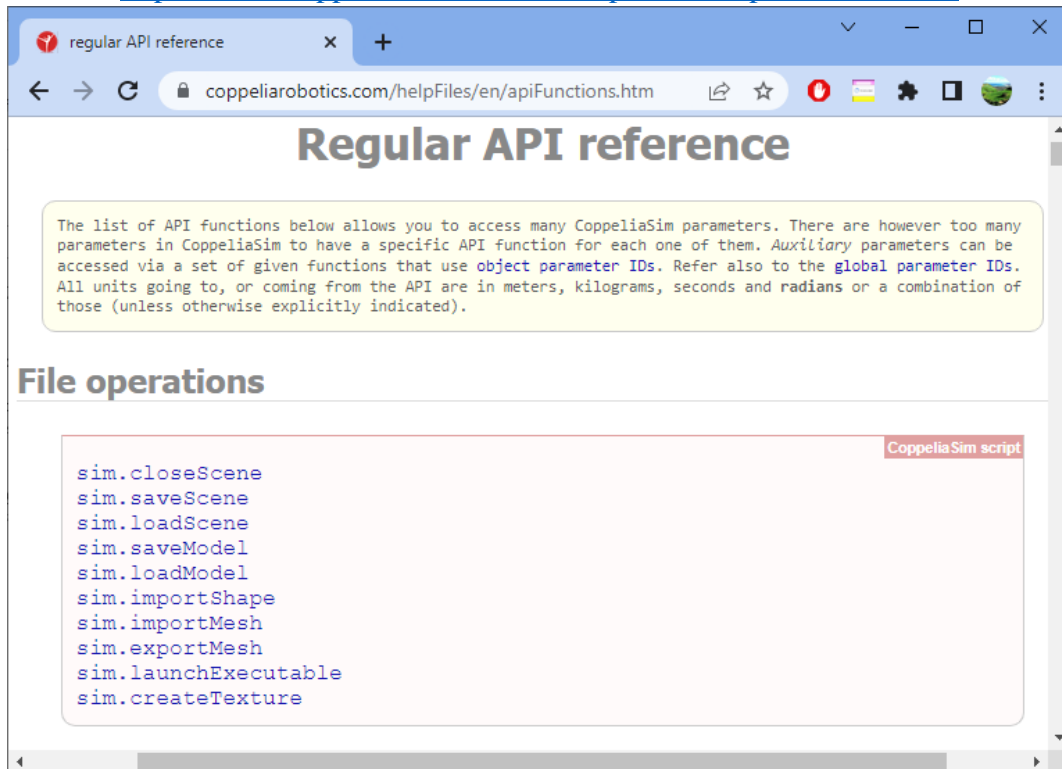
Cliquer sur l'icône du Child script près du «LineTracer »,  LineTracer

La fenêtre du script Lua s'affiche.

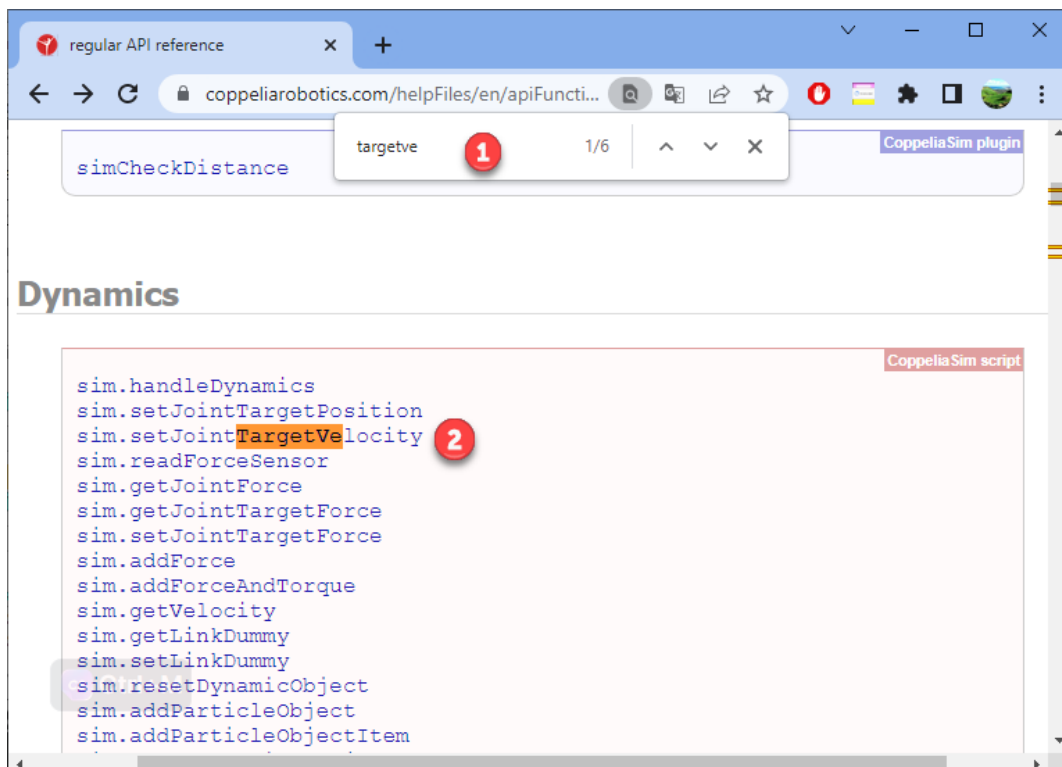


Afin de maîtriser efficacement les diverses fonctionnalités du langage "Lua", il est possible de se référer aux fichiers d'aide disponibles à partir du lien suivant ou de consulter les documents fournis avec ce TP :

<https://www.coppeliarobotics.com/helpFiles/en/apiFunctions.htm>



Exemple : Pour la suite du TP, il est nécessaire de disposer de la fonction qui contrôle la vitesse du moteur. Pour la trouver, il convient d'appuyer sur la touche F3 du clavier, puis rechercher la fonction souhaitée dans le navigateur.



The screenshot shows the 'Regular API function' page for `simSetJointTargetVelocity` on the Coppelia Robotics website. The browser address bar shows the URL: `coppeliarobotics.com/helpFiles/en/regularApi/simSetJoint...`. The page title is 'Regular API function'. Below the title, the function name is listed: `simSetJointTargetVelocity / sim.setJointTargetVelocity`. The page contains a table with the following information:

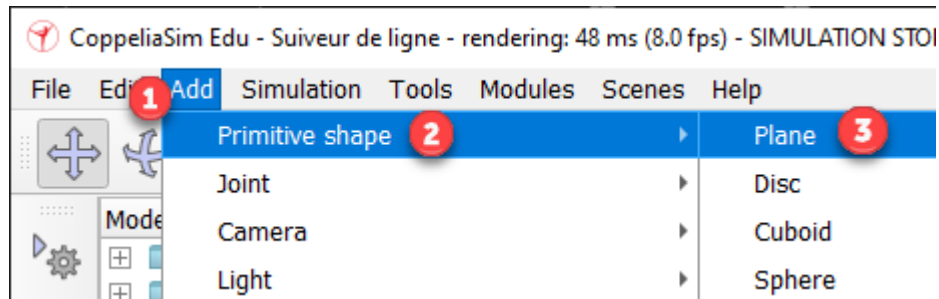
Description	Sets the target linear/angular velocity of a non-spherical joint. When in kinematic mode, the joint will move according to a motion profile that respects maximum acceleration and jerk values. In dynamic and velocity control mode, the controller is instructed about the desired velocity. See also sim.getJointTargetVelocity .
C/C++ synopsis	<code>simInt simSetJointTargetVelocity(simInt objectHandle, simFloat targetVelocity)</code>
C/C++ parameters	objectHandle: handle of the joint object targetVelocity: target velocity of the joint
C/C++ return value	-1 if operation was not successful
Lua synopsis	<code>sim.setJointTargetVelocity(int objectHandle, float targetVelocity, float[] motionParams={})</code> 1
Lua parameters	objectHandle: handle of the joint object targetVelocity: target velocity of the joint motionParams: when the joint is in kinematic mode: the maximum allowed acceleration and jerk. When in dynamic mode with motion profile control enabled: the maximum allowed acceleration and

Il est indispensable d'avoir une autre fonction pour identifier la couleur correspondante à partir d'un capteur approprié.

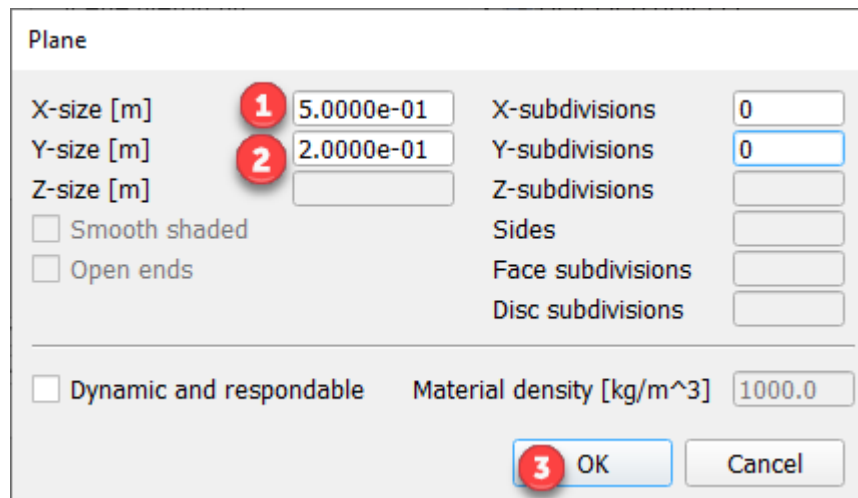
The screenshot shows the 'Regular API function' page for `simReadVisionSensor` on the Coppelia Robotics website. The browser address bar shows the URL: `coppeliarobotics.com/helpFiles/en/regularApi/simReadVis...`. The page title is 'Regular API function'. Below the title, the function name is listed: `simReadVisionSensor / sim.readVisionSensor`. The page contains a table with the following information:

Description	Reads the state of a vision sensor. This function doesn't perform detection, it merely reads the result from a previous call to <code>sim.handleVisionSensor</code> (<code>sim.handleVisionSensor</code> is called in the default main script). See also sim.checkVisionSensor , sim.checkVisionSensorEx and sim.resetVisionSensor .
C/C++ synopsis	<code>simInt simReadVisionSensor(simInt visionSensorHandle, simFloat** auxValues, simInt** auxValuesCount)</code>
C/C++ parameters	visionSensorHandle: handle of a vision sensor object auxValues: by default CoppeliaSim returns one packet of 15 auxiliary values: the minimum of {intensity, red, green, blue, depth value}, the maximum of {intensity, red, green, blue, depth value}, and the average of {intensity, red, green, blue, depth value}. Additional packets can be appended in the vision callback function . auxValues can be nullptr. The user is in charge of releasing the auxValues buffer with <code>simReleaseBuffer(*auxValues)</code> . auxValuesCount: contains information about the number of auxiliary value packets and packet sizes returned in auxValues . The first value is the number of packets, the second is the size of packet1, the third is the size of packet2, etc. Can be nullptr if auxValues is also nullptr. The user is in charge of releasing the auxValuesCount buffer with <code>simReleaseBuffer(*auxValuesCount)</code> . See simHandleVisionSensor for a usage example
C/C++ return value	detection state (0 or 1), or -1 in case of an error, or if <code>simHandleVisionSensor</code> was never called, or if <code>simResetVisionSensor</code> was previously called.
Lua synopsis	<code>int result, float[] auxiliaryValuePacket1, float[] auxiliaryValuePacket2, etc. = sim.readVisionSensor(int visionSensorHandle)</code> 1
Lua parameters	visionSensorHandle: handle of a vision sensor object

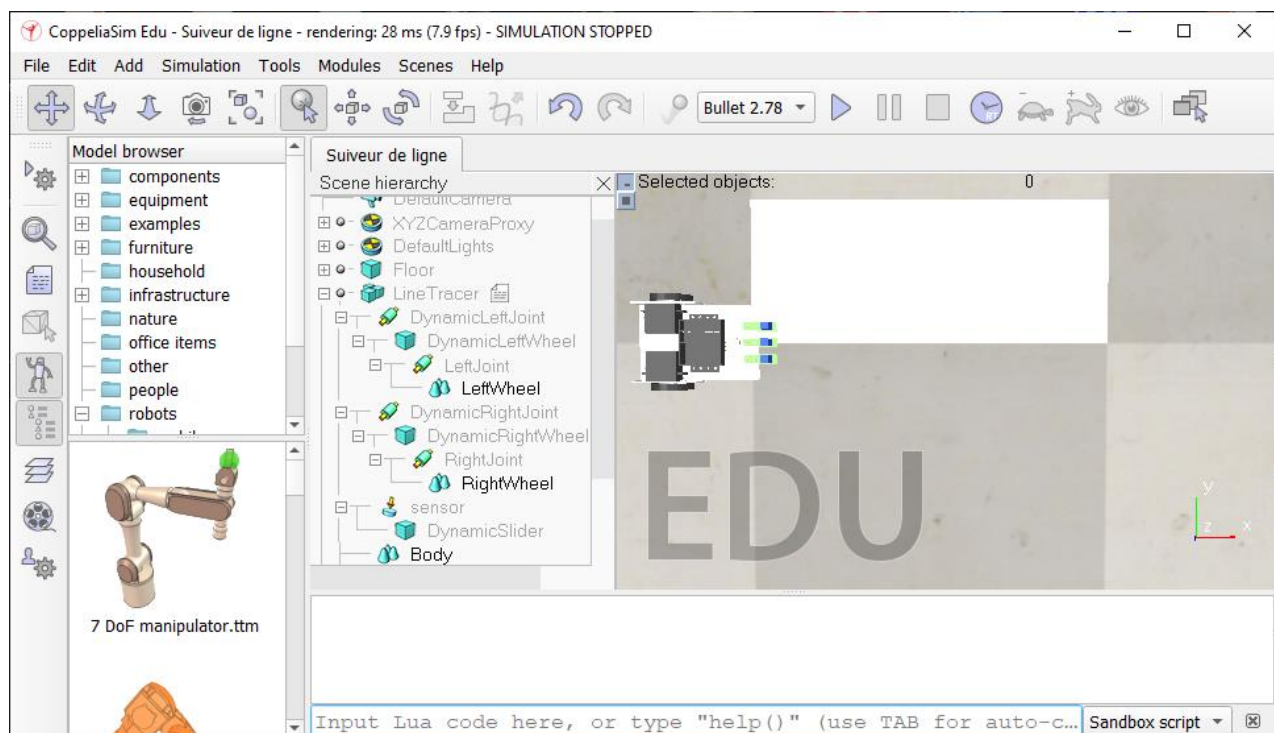
Pour obtenir une indication des mesures données par le capteur de vision, il suffit de cliquer « Add », puis « Primitive shape » et ajouter « Plane » à la scène.



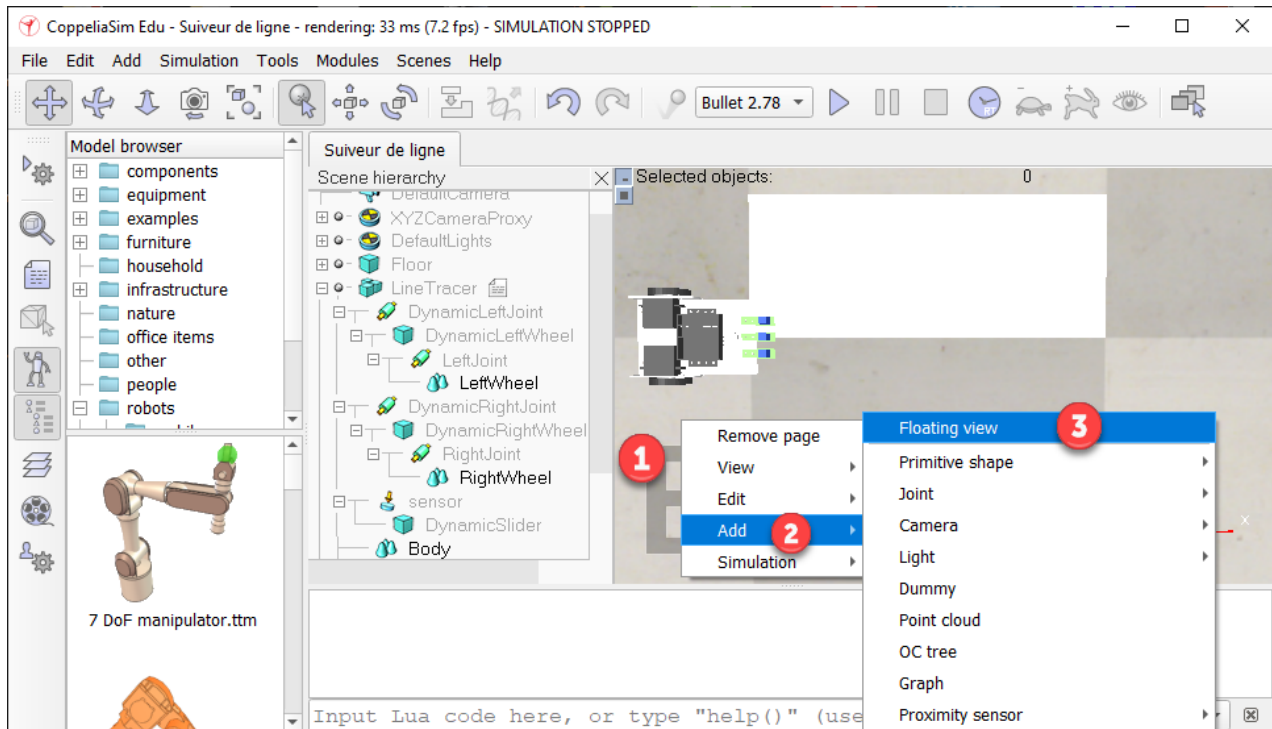
Les dimensions en « X-size[m] » et « Y-size[m] » d'un plan « Plane » indiquées dans la figure suivante seront ajoutées.



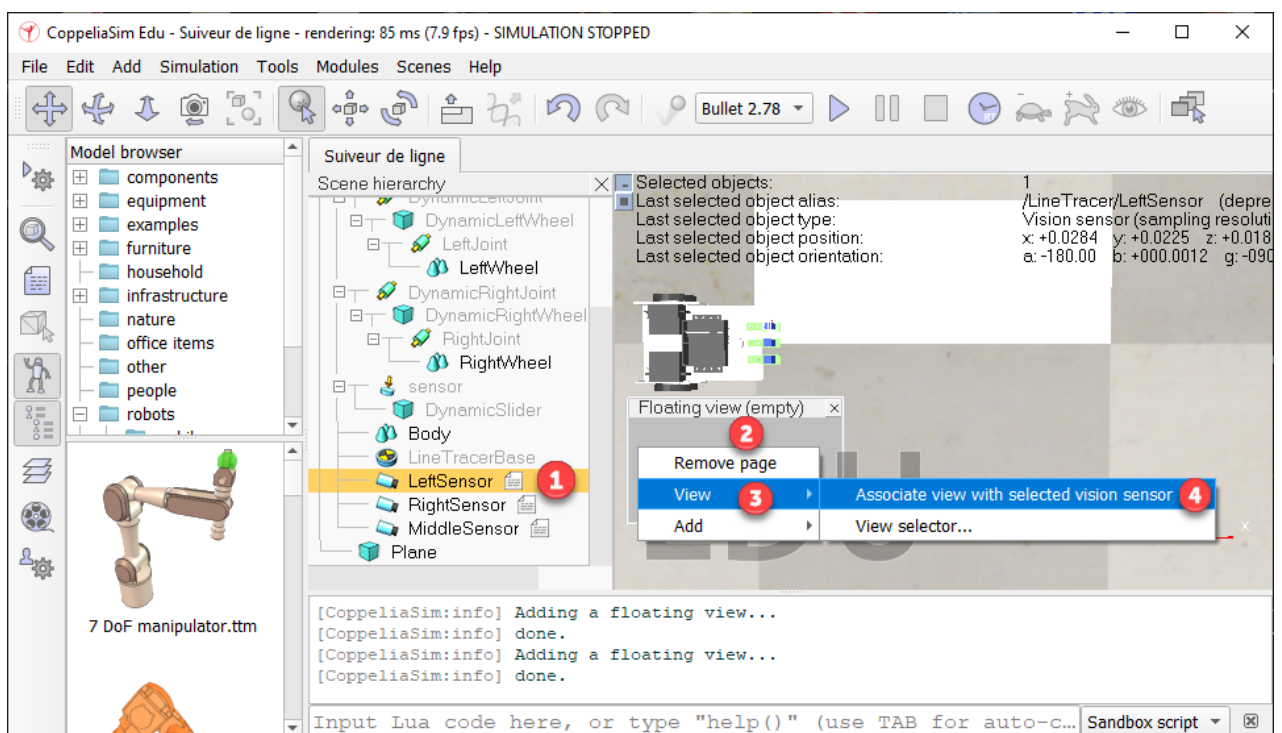
Positionner « Plane » au milieu du robot comme indique la figure suivante :



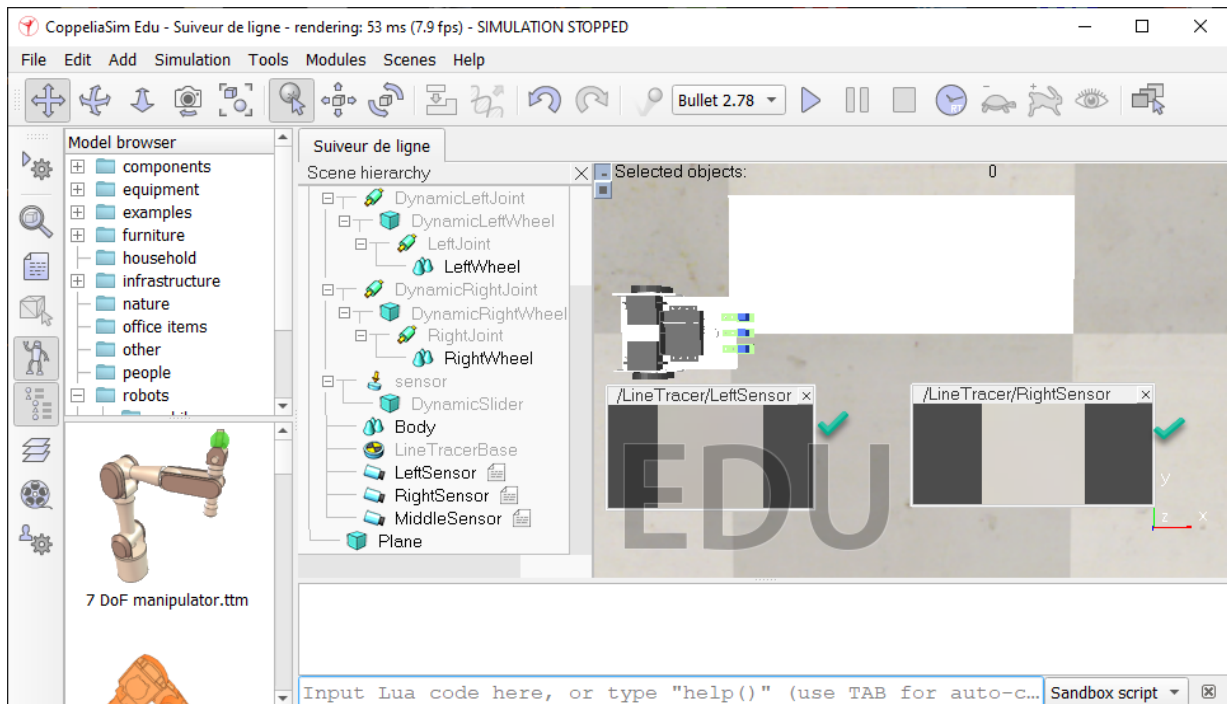
Par la suite, ajouter une vue flottante, par un clic droit sur la scène, puis sélectionner « Add », « Floating view »



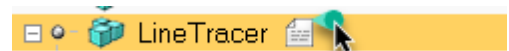
Dans la « scène hierarchy » sélectionner « LeftSensor ». Ensuite cliquer sur « Floating view (empty) », « View » puis « Associate view with selected vision sensor »



Refaites les mêmes étapes pour ajouter une vue flottante pour le capteur de vision « RightSensor ».



Cliquer sur l'icône du Child script près du «LineTracer»,



Ecrire dans l'éditeur «Child script /'LineTracer' » le code suivant :

```

Child script "/LineTracer"

1 function sysCall_init()
2     -- do some initialization here
3     leftJoint = sim.getObject('/DynamicLeftJoint')
4     rightJoint = sim.getObject('/DynamicRightJoint')
5     leftSensor = sim.getObject('/LeftSensor')
6     rightSensor = sim.getObject('/RightSensor')
7     -- print(leftJoint)
8 end
9
10 function sysCall_actuation()
11     -- put your actuation code here
12     vl=2
13     vr=2
14     sim.setJointTargetVelocity( leftJoint,vl)
15     sim.setJointTargetVelocity( rightJoint,vr)
16
17 end
18
19 function sysCall_sensing()
20     -- put your sensing code here
21     result,data_left=sim.readVisionSensor(leftSensor)
22     print (result)

```

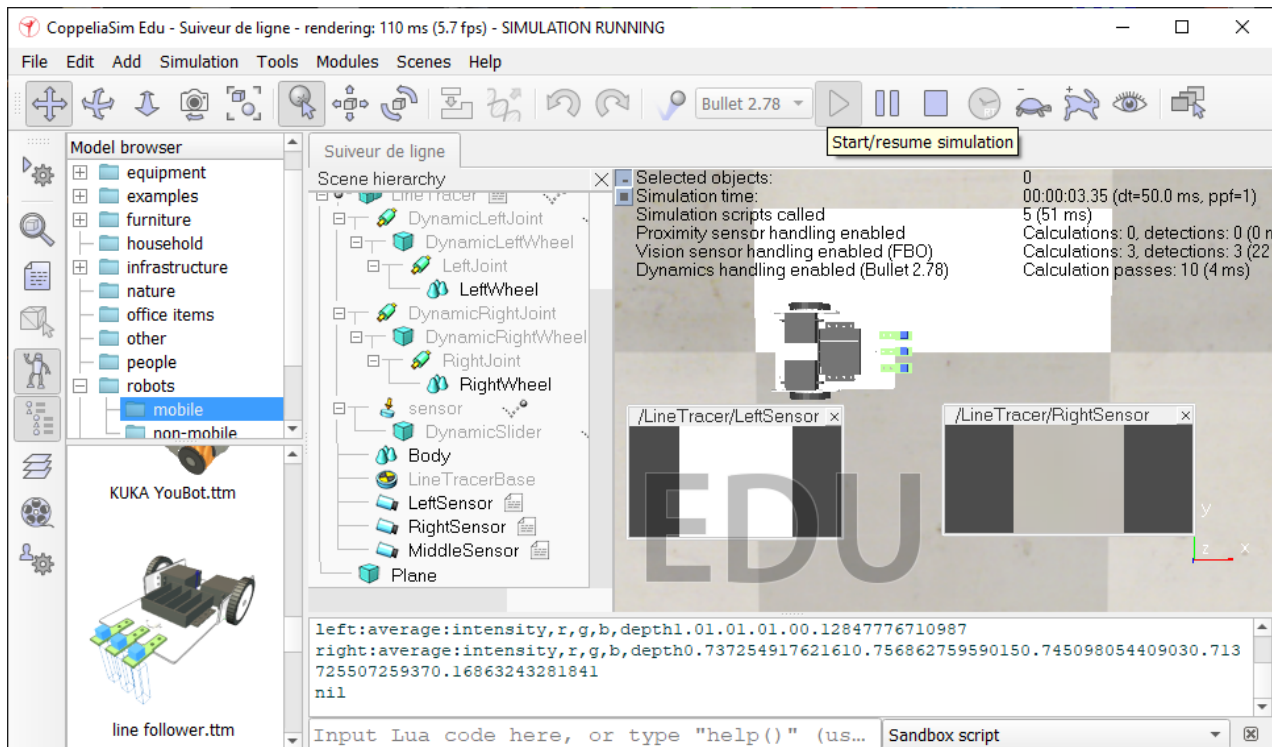
⇒ La suite du code :

```

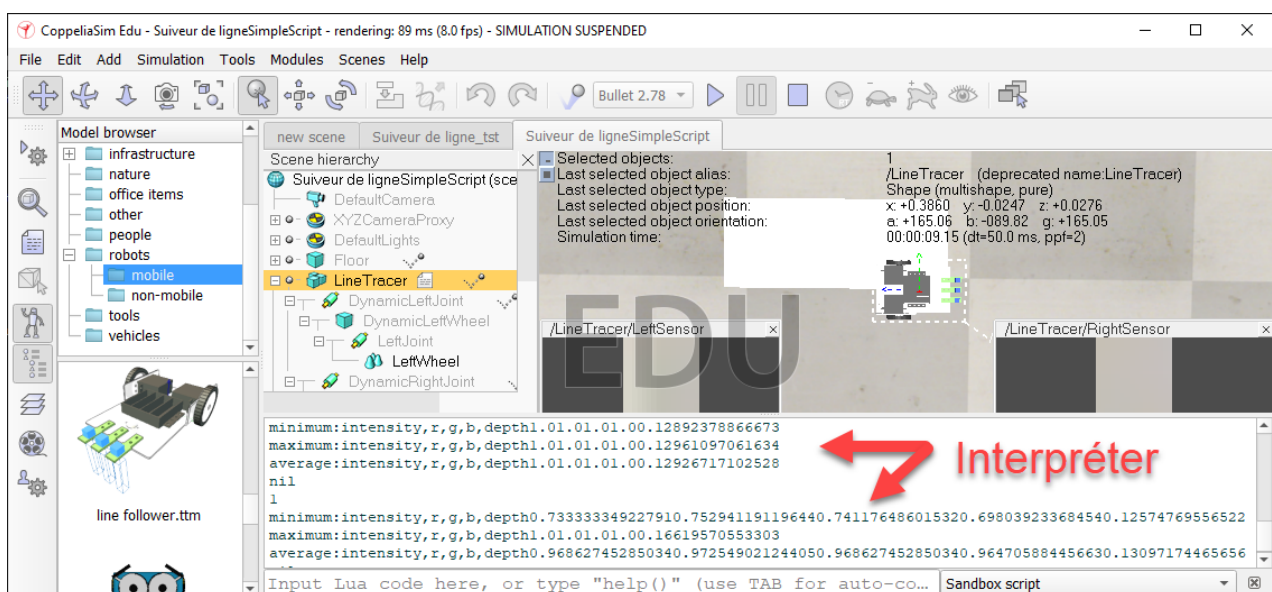
23
24 if (data_left ~=nil)then
25   print ('minimum:intensity,r,g,b,depth' ..data_left[1] ..data_left[2] ..data_left[3] ..data_left[4] ..data_left[5])
26   print ('maximum:intensity,r,g,b,depth' ..data_left[6] ..data_left[7] ..data_left[8] ..data_left[9] ..data_left[10])
27   print ('average:intensity,r,g,b,depth' ..data_left[11] ..data_left[12] ..data_left[13] ..data_left[14] ..data_left[15])
28   print ()
29 end
30
31 end
32

```

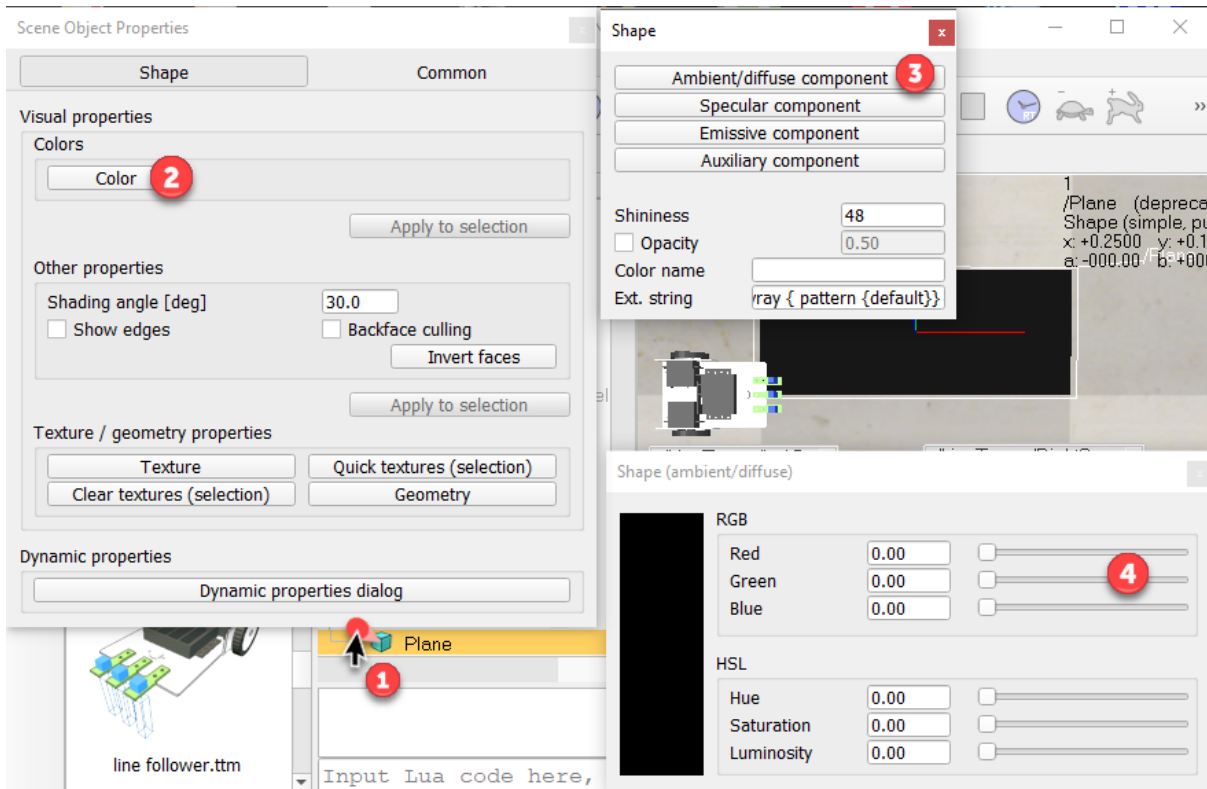
⇒ Cliquer sur « Start/resume simulation »



⇒ Interpréter les mesures obtenues dans « Sandbox script »

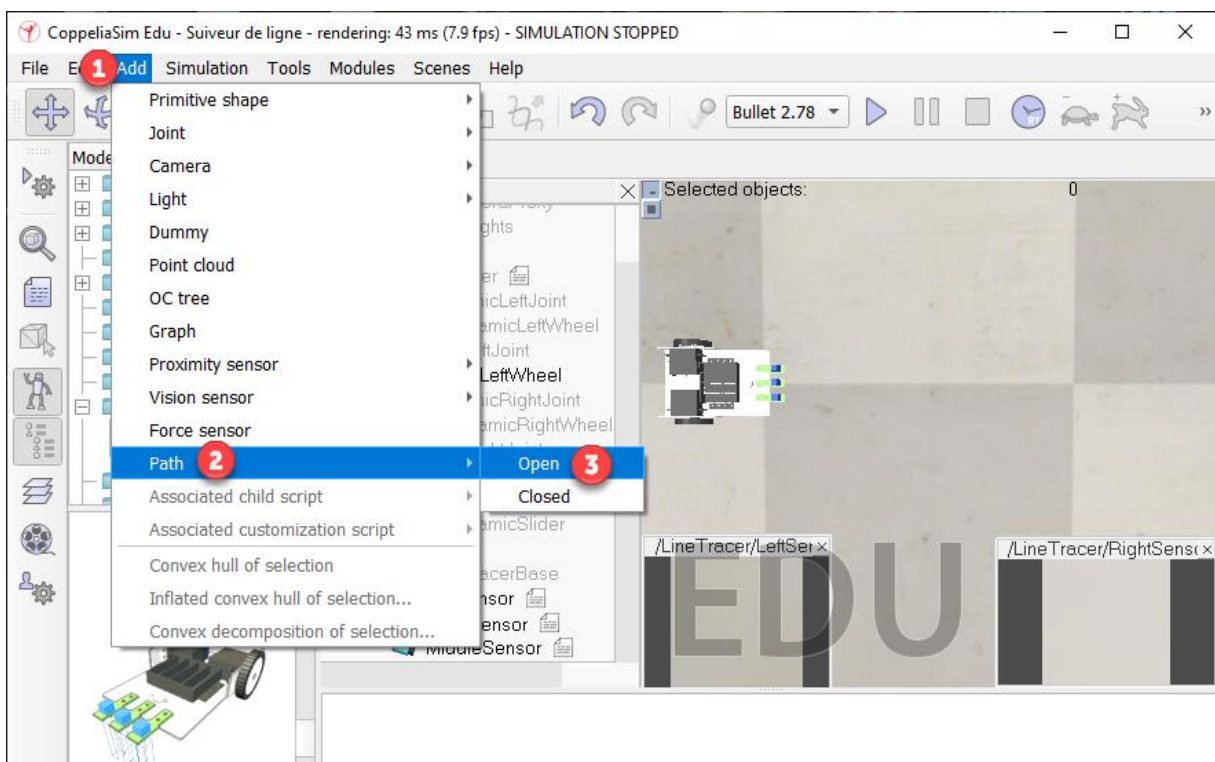


Changer la couleur du plan « Plane » en noir comme l'indique la figure suivante et interpréter les mesures après simulation.



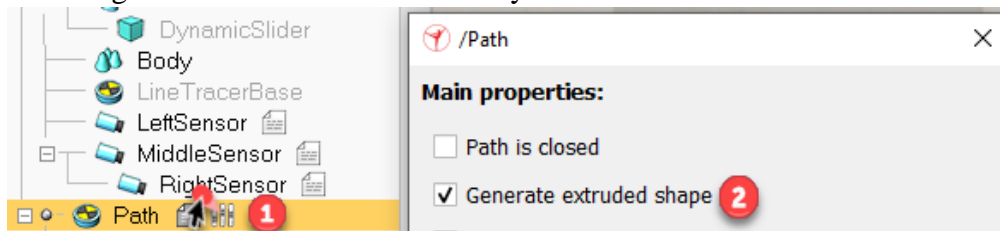
Pour une meilleure conception de parcours à suivre par le robot, on peut remplacer un plan « Plane » par un chemin « Path » ouvert ou fermé.

Dans la « scène hierarchy » sélectionner « plane » puis l'effacer, ensuite cliquer sur « Add », « Path » et choisir « Open »

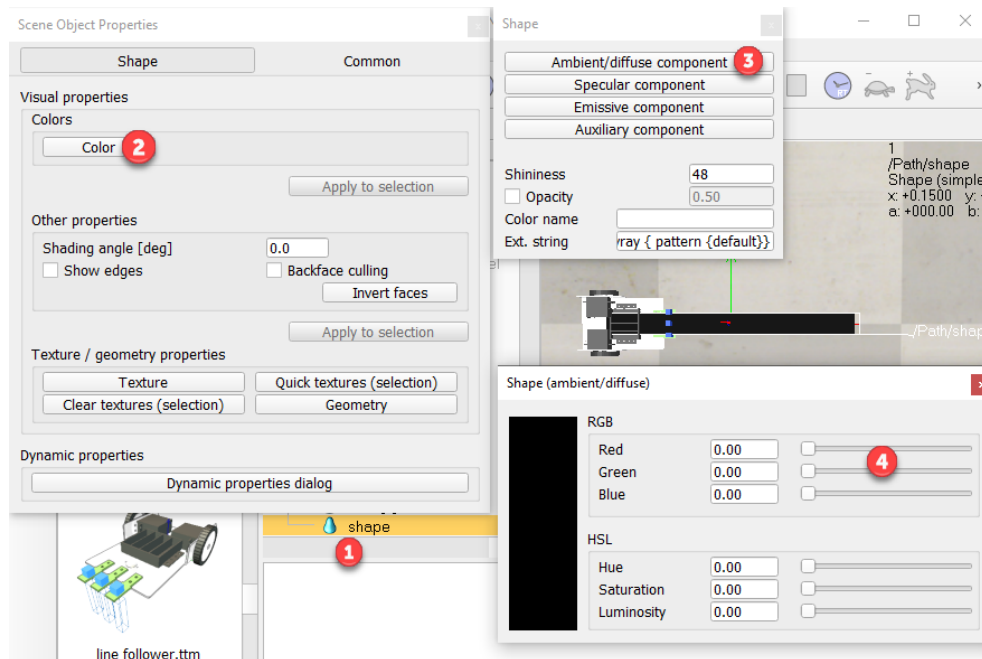


Dans la « scène hierarchy » sélectionner « Path », cliquer sur « Trigger callback » puis cocher « Generate extruded shape ».

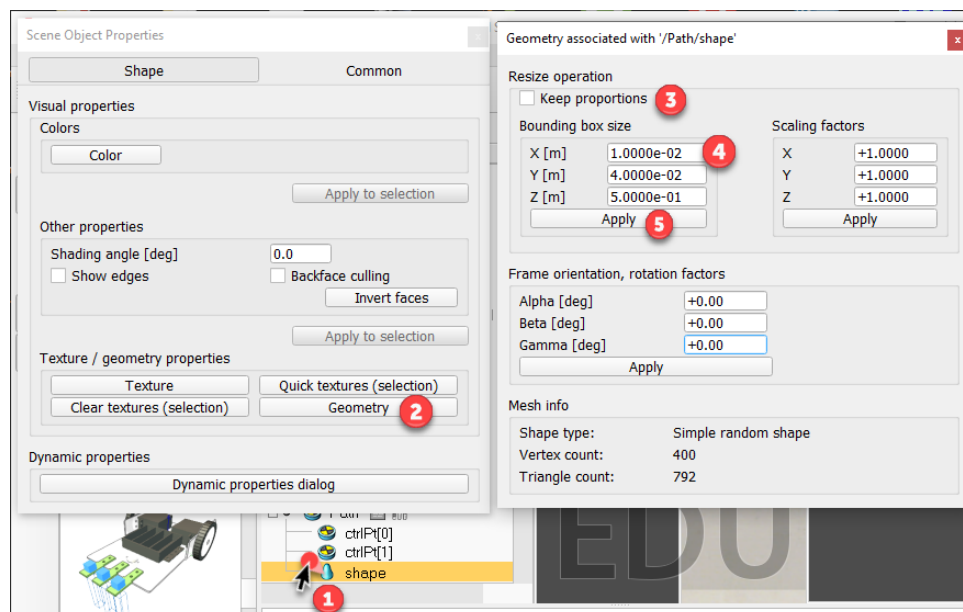
Un « shape » sera généré dans la « scène hierarchy ».



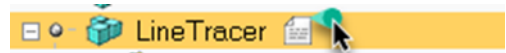
Dans la « scène hierarchy » sélectionner « shape », et cliquer son icône et choisir la couleur noire comme l'indique la figure suivante :



Pour changer l'épaisseur du « shape » sélectionner « Geometry », décocher « keep proportions » puis changer la valeur de « X[m] » en 0.01



Cliquer sur l'icône du Child script près du «LineTracer»,



Ecrire dans l'éditeur «Child script /'LineTracer' » le code suivant :

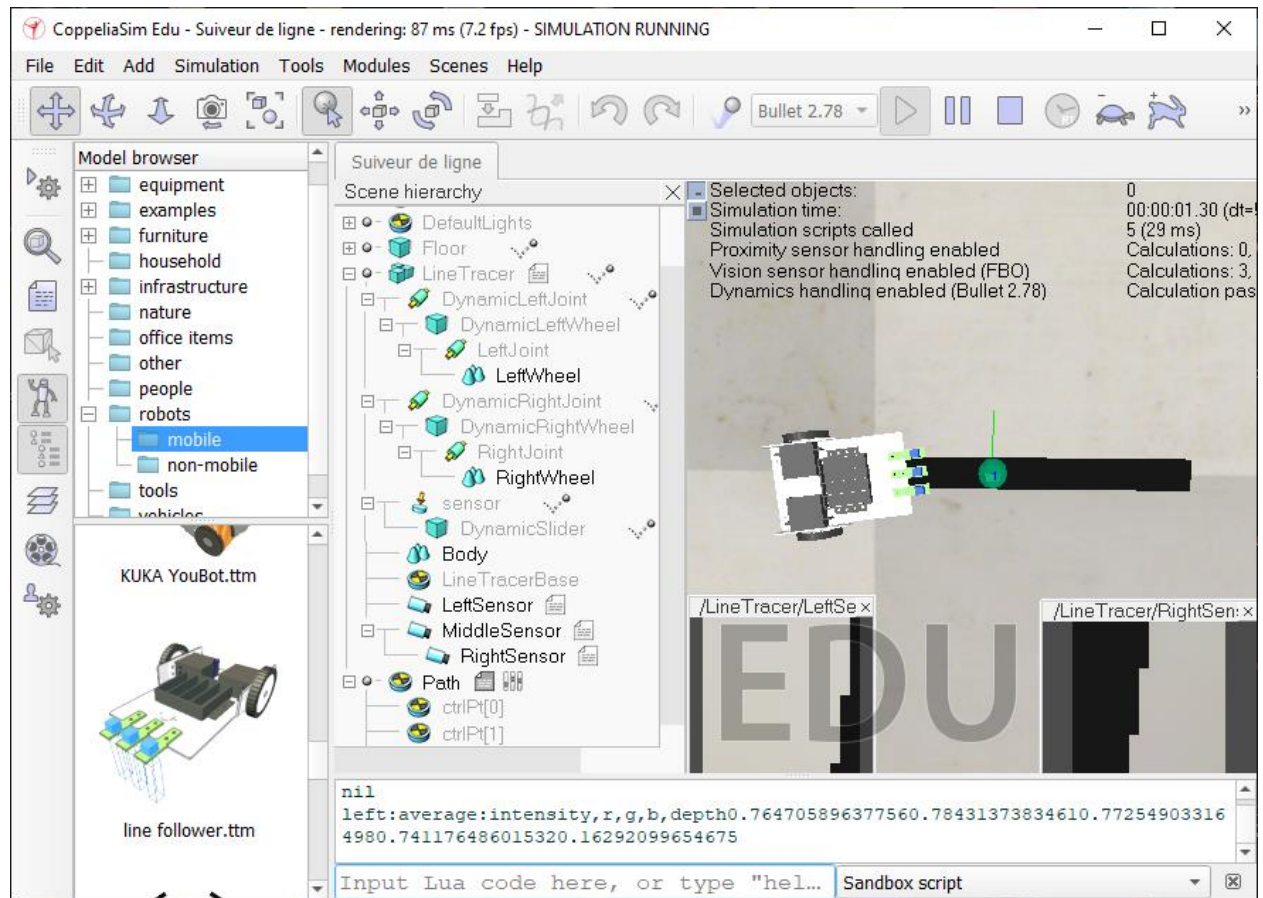
```

Child script "/LineTracer"

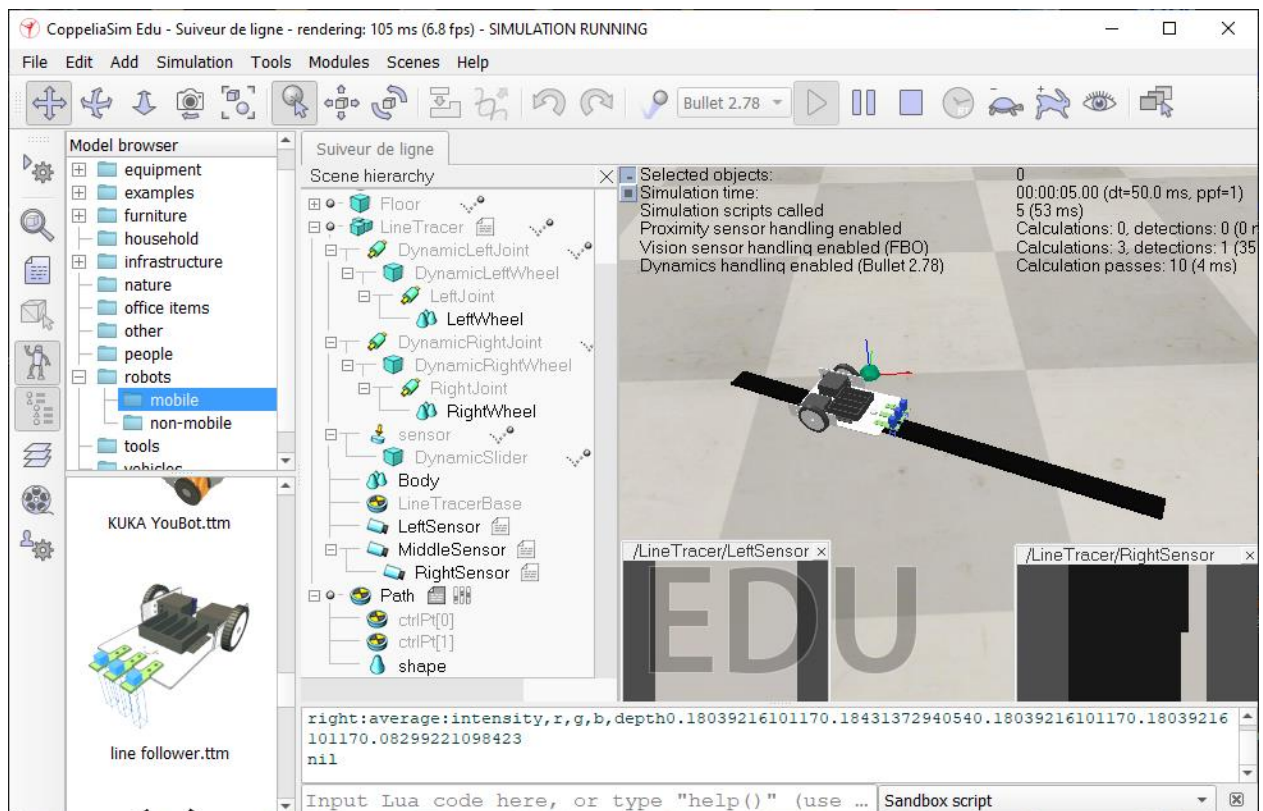
1 function sysCall_init()
2     -- do some initialization here
3     leftJoint = sim.getObject('/DynamicLeftJoint')
4     rightJoint = sim.getObject('/DynamicRightJoint')
5     leftSensor = sim.getObject('/LeftSensor')
6     rightSensor = sim.getObject('/RightSensor')
7     -- print(leftJoint)
8 end
9
10 function sysCall_actuation()
11     -- put your actuation code here
12     vl=2
13     vr=2
14     sim.setJointTargetVelocity( leftJoint,vl)
15     sim.setJointTargetVelocity( rightJoint,vr)
16     if (data_left ~=nil)then
17         intensity_left=data_left[11]
18         intensity_right=data_right[11]
19         if (intensity_right > 0.1 and intensity_left <0.1) then
20             print('Turn right')
21             sim.setJointTargetVelocity( rightJoint,vl+vr)
22         end
23         if (intensity_left > 0.1 and intensity_right <0.1) then
24             print('Turn left')
25             sim.setJointTargetVelocity( leftJoint,vl+vr)
26         end
27     end
28 end
29 end
30
31 function sysCall_sensing()
32     -- put your sensing code here
33     result,data_left=sim.readVisionSensor(leftSensor)
34     result,data_right=sim.readVisionSensor(rightSensor)
35     -- print (result)
36     --if (data_left ~=nil)then
37
38     --print ('minimum:intensity,r,g,b,depth' ..data_left[1] ..data_left[2] ..data_left[3] ..data_left[4] ..data_left[5])
39     --print ('maximum:intensity,r,g,b,depth' ..data_left[6] ..data_left[7] ..data_left[8] ..data_left[9] ..data_left[10])
40     --print ('average:intensity,r,g,b,depth' ..data_left[11] ..data_left[12] ..data_left[13] ..data_left[14] ..data_left[15])
41     print ('left:average:intensity,r,g,b,depth' ..data_left[11] ..data_left[12] ..data_left[13] ..data_left[14] ..data_left[15])
42     print ('right:average:intensity,r,g,b,depth' ..data_right[11] ..data_right[12] ..data_right[13] ..data_right[14] ..data_right[15])
43     print()
44     -- end
45
46 end
47
48 function sysCall_cleanup()
49     -- do some clean-up here
50 end
51
52 -- See the user manual or the available
53 -- code snippets for additional callback
54 -- functions and details
55

```


Lancer la simulation et interpréter le résultat.



Changer la direction du « path », lancer la simulation et interpréter le résultat.



Dans la « scène hierarchy » supprimer le parcours qui ouvert « path », « Open » et le remplacer par un parcours fermé « path », « Closed » en suivant les mêmes étapes.

Créer par la suite votre propre chemin, lancer la simulation et interpréter le résultat.

