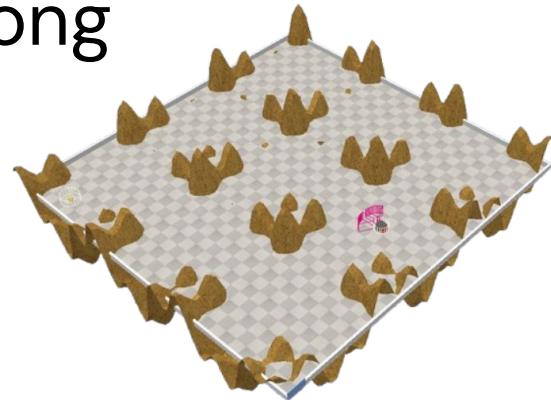
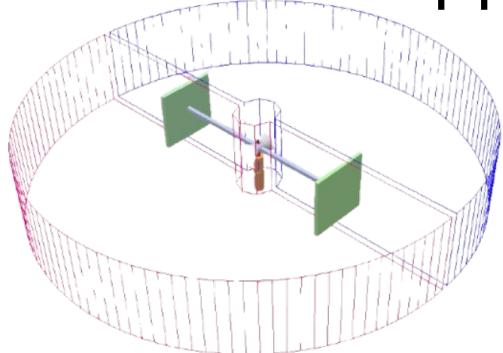


Embodied AI Class Experiments

Prof. Poramate Manoonpong



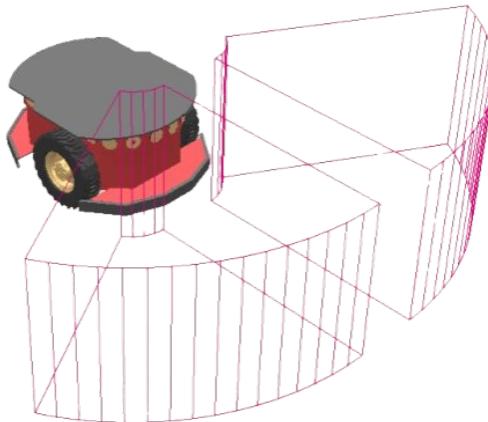
Introduction

These assignments constitute an integral component of the **Embodied AI** course taught by Prof. Poramate Manoonpong. They are specifically designed to ensure that students develop a thorough understanding of the theoretical principles and methodological foundations of Embodied Artificial Intelligence. Furthermore, the assignments aim to enable students to effectively apply the acquired knowledge and skills to the implementation of computational tasks as well as to practical real-world applications, thereby reinforcing the connection between theory and practice.

To support this objective, the assignments will be conducted using the **CoppeliaSim simulation** environment, with control algorithms implemented through **Lua scripting**, allowing students to design, test, and evaluate embodied controllers in a realistic and interactive simulated setting.

Table of Content

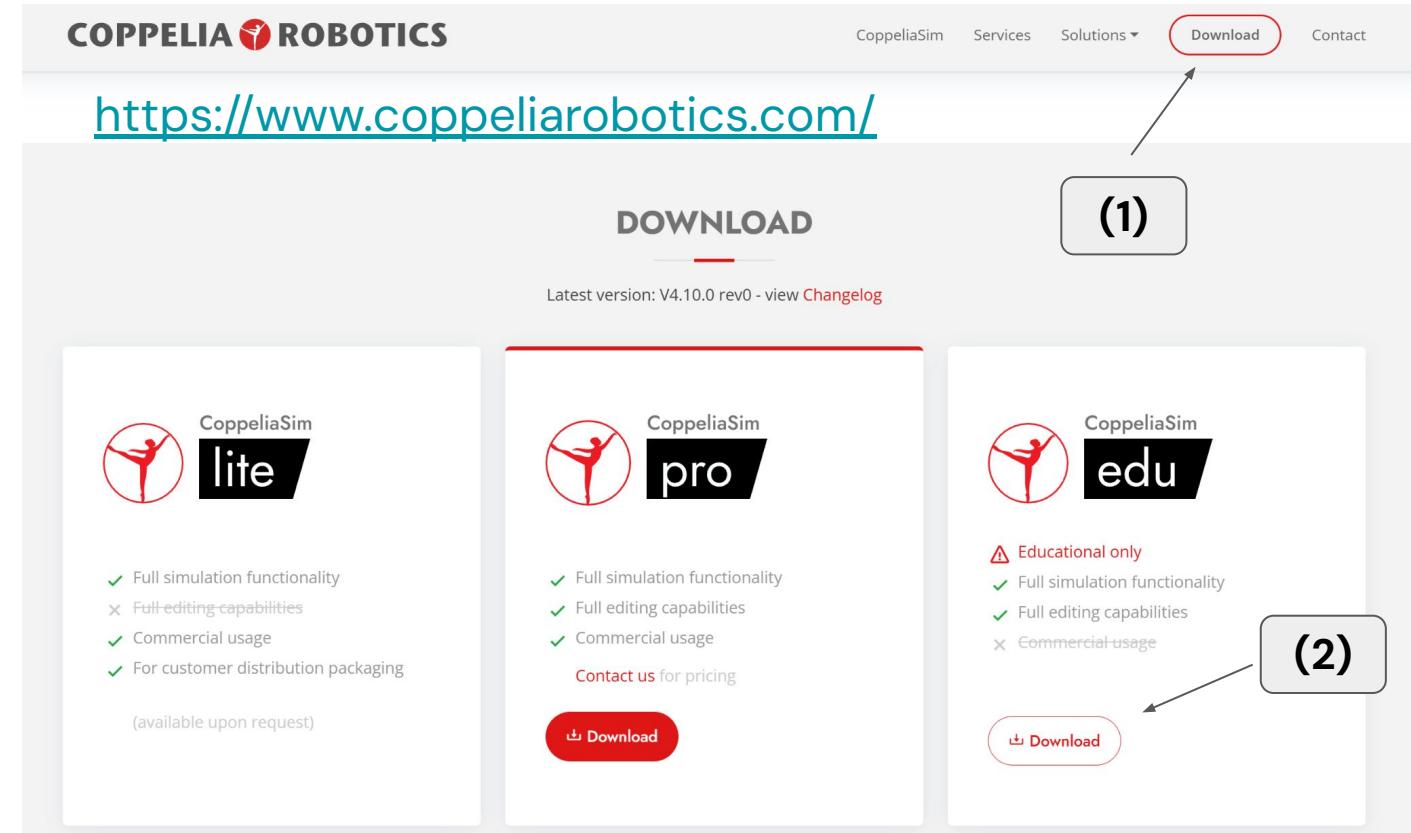
1. CoppeliaSim Simulation
 - a. [Download](#)
 - b. [Shortcuts](#)
 - c. [Opening lua script](#)
 - d. [Script structure](#)
 - e. Run and stop simulation
2. [Download Scene](#)
3. [Lua supporting function](#)
4. Tasks
 - a. [ICO learning](#)
 - b. [Q-learning](#)
 - c. [XOR](#)
5. [Record data to CSV file](#)



Download CoppeliaSim simulation

Download
click here

- On the web page
Goto (1) and (2)
- Then following the
installation



The screenshot shows the CoppeliaRobotics website homepage. At the top, there is a navigation bar with links for "CoppeliaSim", "Services", "Solutions", "Download" (which is highlighted with a red border), and "Contact". Below the navigation bar, the URL <https://www.coppeliarobotics.com/> is displayed. A large "DOWNLOAD" button is centered on the page, with the text "Latest version: V4.10.0 rev0 - view Changelog" underneath it. Three download options are shown in separate boxes:

- (1) CoppeliaSim lite**: Includes icons for a person dancing and a robot. It lists features: ✓ Full simulation functionality, ✗ Full editing capabilities, ✓ Commercial usage, and ✓ For customer distribution packaging. It also notes "(available upon request)". A "Download" button is at the bottom.
- (2) CoppeliaSim pro**: Includes icons for a person dancing and a robot. It lists features: ✓ Full simulation functionality, ✓ Full editing capabilities, ✓ Commercial usage. It includes a "Contact us for pricing" link and a "Download" button.
- (2) CoppeliaSim edu**: Includes icons for a person dancing and a robot. It lists features: ✓ Full simulation functionality, ✓ Full editing capabilities, ✗ Commercial usage. It includes a "Download" button.

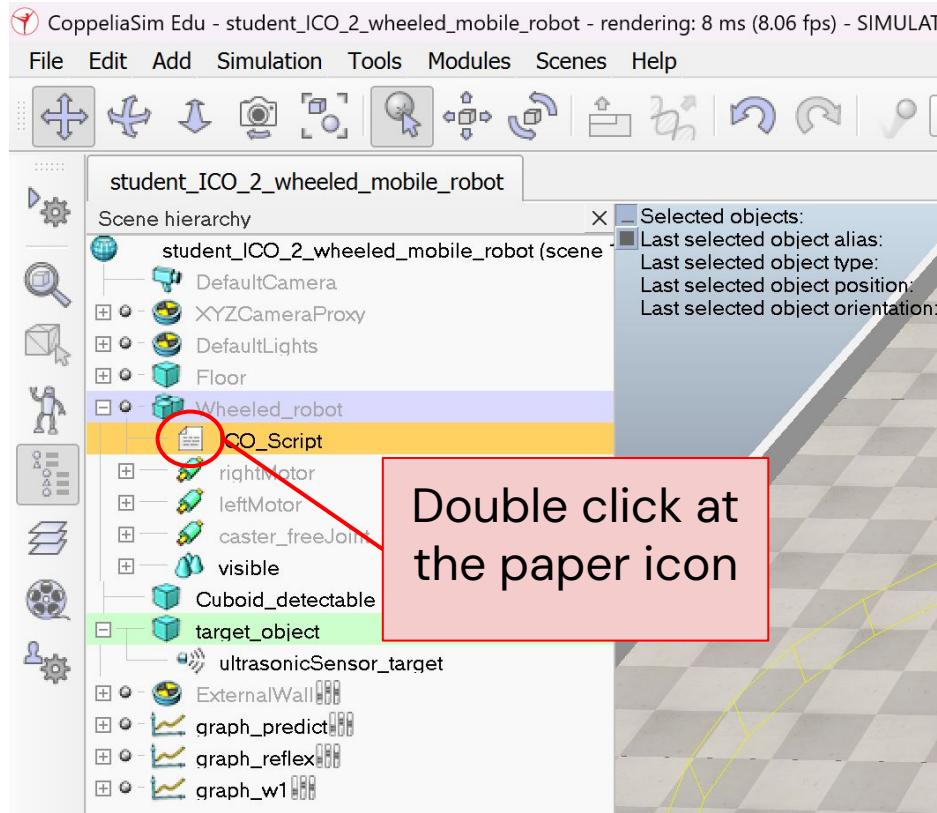
CoppeliaSim shortcuts

Shortcuts

When the focus is on the [scene hierarchy](#) or a [page](#), following shortcut keys are supported:

- **CTRL+A**: select all
- **<esc>**: clear the selection
- **CTRL+C**: copy the selection
- **CTRL+V**: paste the copy buffer
- **CTRL+X**: cut the selection
- **<delete>**: delete the selection
- **<backspace>**: delete the selection
- **CTRL+O**: open a scene
- **CTRL+N**: open a new scene
- **CTRL+S**: save the scene
- **CTRL+W**: close a scene
- **CTRL+Q**: quit the application
- **CTRL+<space>**: start/stop the simulation
- **CTRL+E**: toggle between 1) normal, 2) object translation and 3) object rotation mouse mode
- **CTRL+D**: open the object property dialog
- **CTRL+G**: open the calculation module dialog
- **CTRL+B**: adjust the view to fit selected objects, or the whole scene if no object is selected. The focus needs to be on a view.
- **CTRL+ALT+C**: set focus on the commander
- **CTRL+L**: clear status bar (when focus is on the commander)

To open lua script



The screenshot shows the 'Simulation script' window with the file path 'Wheeled_robot/ICO_Script'. The code in the window is as follows:

```
1 sim=require'sim'  
2  
3 function sysCall_init()  
4     local robot=sim.getObject('...')  
5     local obstacles=sim.createCollection(0)  
6     sim.addItemToCollection(obstacles,sim.handle_all,-1,0)  
7     sim.addItemToCollection(obstacles,sim.handle_tree,robot,1)  
8  
9     -- Add robot  
10    robot_object = sim.getObject("/Wheeled_robot")  
11  
12    -- Add sensor  
13    left_sensor = sim.getObject("../ultrasonicSensor_left")  
14    right_sensor = sim.getObject("../ultrasonicSensor_right")  
15  
16    -- Add motor  
17    motorLeft=sim.getObject("../leftMotor")  
18    motorRight=sim.getObject("../rightMotor")  
19  
20    -- Add target  
21    target_object = sim.getObject("/target")  
22    target_sensor = sim.getObject("/ultrasonicSensor_target")  
23  
24    -- Add environment  
25    v0=2  
26  
27  
28  
29  
30    -- Add Graph  
31    graph_predict = sim.getObject("/graph_predict")  
32    graph_val_omega = sim.addGraphStream(graph_predict, 'omega', 'rad/s', 0, {1,0,0})  
33  
34    graph_reflex = sim.getObject("/graph_reflex")  
35    graph_val_target = sim.addGraphStream(graph_reflex, 'Near target', '', 0, {0,1,0})  
36  
37    graph_w1 = sim.getObject("/graph_w1")  
38    graph_val_w1 = sim.addGraphStream(graph_w1, 'weight predict', '', 0, {0,0,1})  
39  
40    -- ##### Global valuable #####  
41    --
```

A red box surrounds the right side of the code window, and inside this box is the text 'The script window will pop up'.

Lua script structure

Simulation script "/script"

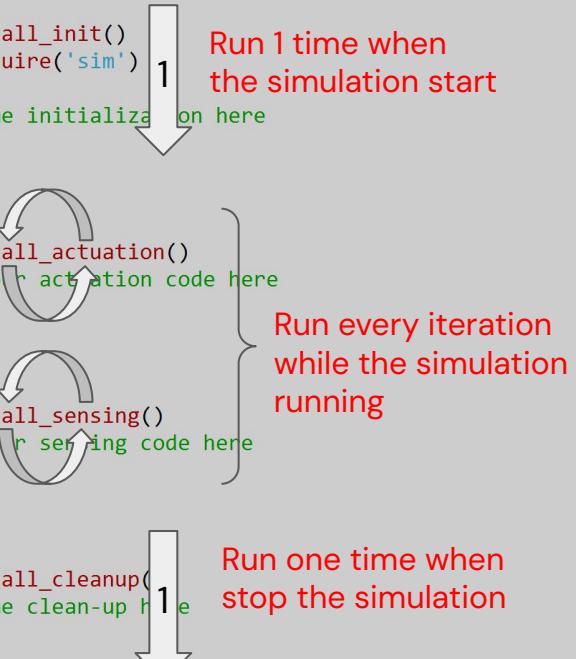
LUA

```
1
2
3 function sysCall_init()
4     sim = require('sim')
5     -- do some initialization here
6 end
7
8
9
10 function sysCall_actuation()
11     -- put your actuation code here
12 end
13
14
15 function sysCall_sensing()
16     -- put your sensing code here
17 end
18
19
20
21 function sysCall_cleanup()
22     -- do some clean-up here
23 end
24
25
26
```

Run 1 time when the simulation start

Run every iteration while the simulation running

Run one time when stop the simulation



SysCall_init():

define global parameter, set initial value

SysCall_actuation():

command motor, set object position/orientation

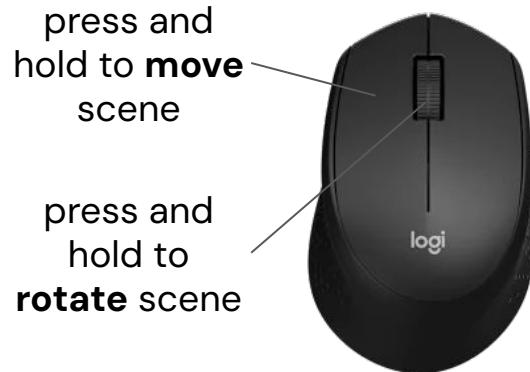
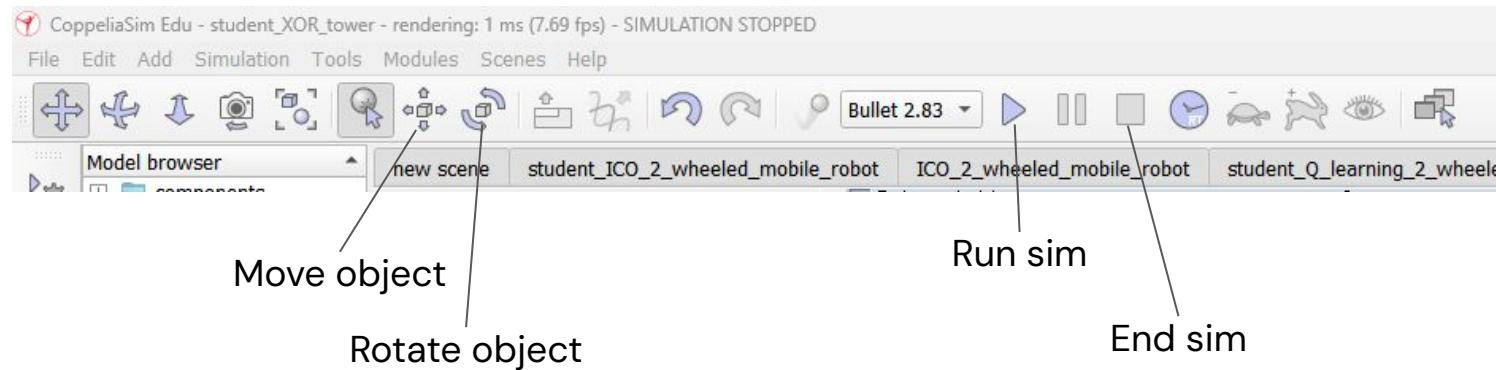
SysCall_sensing():

Get sensor value, read object position/orientation

SysCall_cleanup():

clean the scene, save record value to file

Run and stop simulation



Download Task scenes

2-wheeled robot

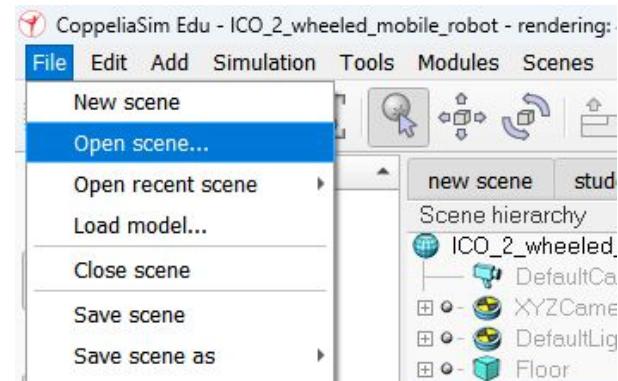
- ICO learning
- Q-learning and SARSA

Simple tower defence

- XNOR neural network

Download

Once you've finished loading the simulation, you can open the task scene by
1) double-clicking the scene file (file.ttt) directly
—or—
2) opening the simulation > Files > open scene... >
[choose your scene]



Supporting function in Lua

Normal functions in Lua → they are in math library

- `math.abs()`
- `math.cos()`
- `math.sin()`
-

Function you might need but there is not in Lua → we have created and you can use

- `max_value_index = argmax(array)`
- `val = randomFloatNumber(minVal, maxVal)`
- `val = sigmoid(val)`
- `val = dsigmoid(val)` → differential of sigmoid function
- `val = tanh(val)`
- `val = dtanh(val)` → differential of tanh function

Record data to CSV file

```
9    -- Add record File
10   logFile = false
11   fileHeader = 'Time, data1, data2, data3, data4, data5' --<<<<
12
13   date_time = os.date("%Y-%m-%d %H-%M-%S")
14   file = ''
15   if logFile then
16     print('Data recording ...')
17     scenePath = sim.getStringParam(sim.stringparam_scene_path)
18
19     file = io.open(scenePath .. '/' .. string.format(date_time) .. '_log.csv',
20     -- write header
21     file:write(fileHeader)
22   end
23
```

```
195  function sysCall_sensing()
196    -- Record data to CSV file
197    simTime = sim.getSimulationTime()
198    if logFile then
199      dataArray = {simTime, 1, 2, 3, 4, 5}
200      recordData(dataArray)
201    end
202
```

- 1) Enable logging file by changing
logFile = true
- 2) You can change or add header name of each data (please follow the format)
fileHeader = "header1, header1, ... headerN"

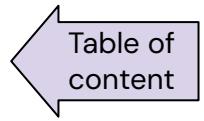
- 3) Change or add the data you need to record following the defined header in the **dataArray**

For example

- **dataArray = {myVal1, myVal2, ... myValN}**

Note: The number of members in **dataArray** must be equal to the members in the defined headers.

This function will record the data



ICO learning

ICO learning task

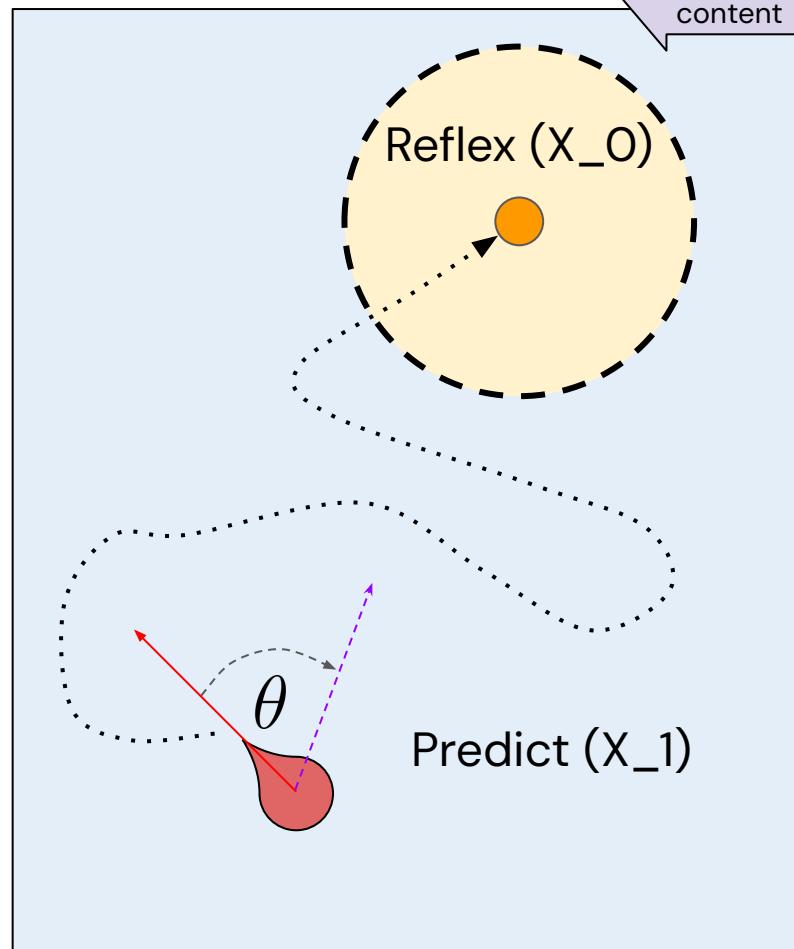
The robot know the theta (angle direction heading to the target). If the robot turns towards the goal, theta is zero (it means the robot doesn't have to turn left (-theta) or turn right (+theta)).

Predict signal (X_1) = theta all the time

Reflex signal (X_0) = theta (if the robot is in yellow area, if not $X_0 = 0$)

Reflex weight (w_0) = 1

Predict weight (w_1) = find it from ICO !!!

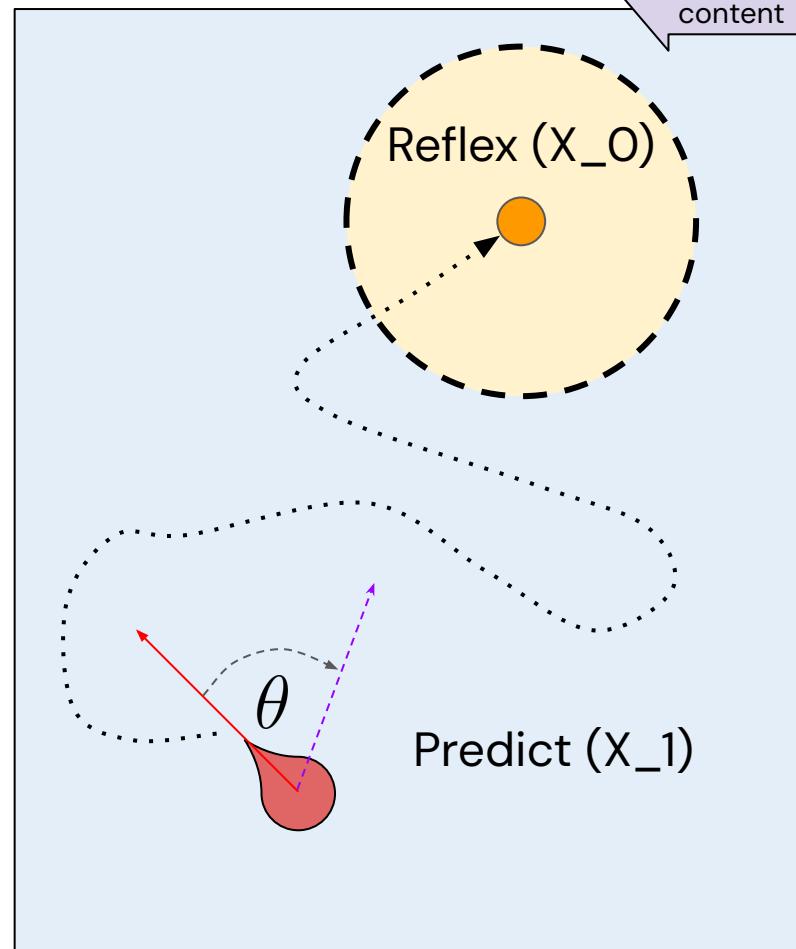


ICO learning task

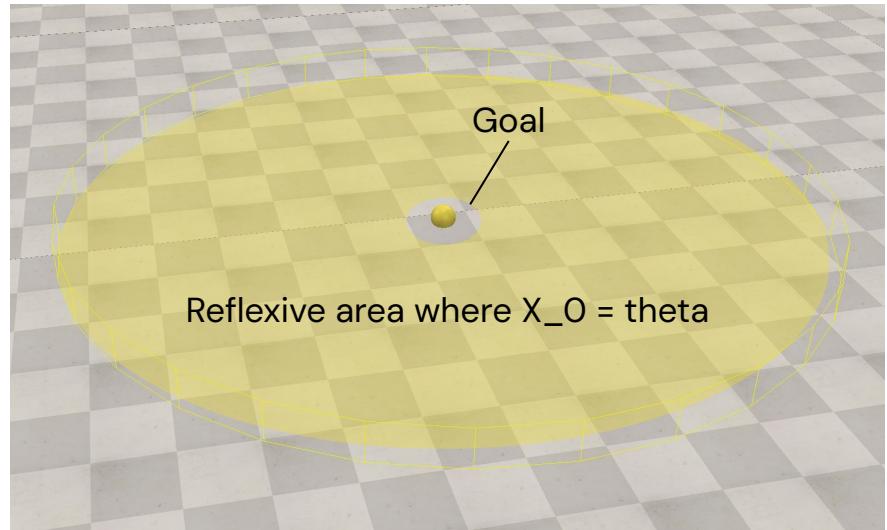
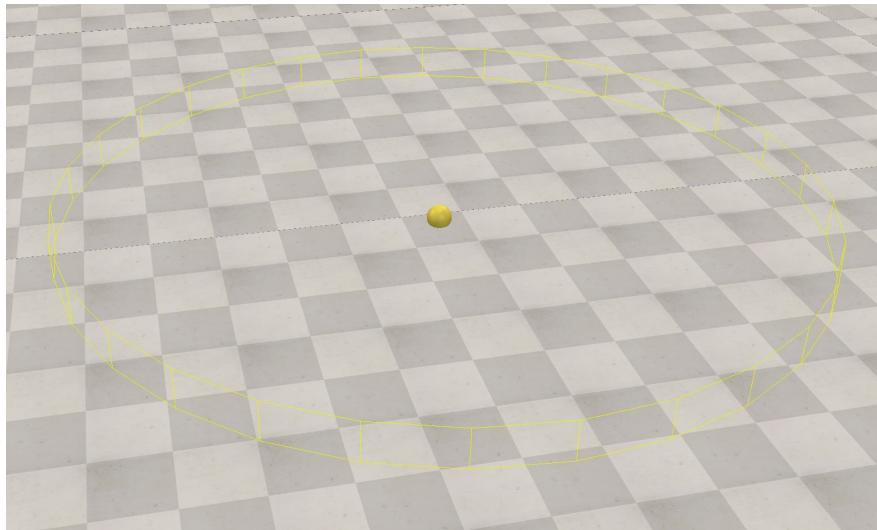
The robot is initiated move forward and random turn. Once, the robot move to yellow area, the robot will move directly to the target because the reflex signal and its weight (=1)

Your task is code the ICO equation to learn the predict weight (W_1) to drive the robot to the target

Ps. If the robot hits an obstacle (target or wall), the robot and target will respawn at a new random position and direction.



ICO learning Goal



ICO learning task

```

82 -- ##### ICO #####
83 -- ICO
84 -- #####
85 --
86 --
87 -- Predict(X1)-----\-----[X]<--\
88 --           \   /   \
89 --           (X)---/   \
90 --           [dx0/dt]--/   \
91 --           |   /   \
92 -- Reflex(x0)---|   /
93 --
94
95 output_angle = 0
96 neural_output = 0
97 noise = 0
98
99 -----
100 -- Student Initial parameter Here
101 -----
102 -- Must use parameter
103
104 x0 = 0      -- reflec signal
105 x1 = 0      -- predict signal
106 alpha = 0.01 -- learning rate
107
108 -- You can edit and add parameters
109 example_val_1 = 0
110 example_val_2 = 2.1
111 example_val_3 = 44
112 example_val_4 = 100
113
114 -----
115 -----
116

```

Initial parameters in this zone

```

124 function ICO_controller()
125
126     -- make noise every 4 seconds
127     if math.floor(simTime)%4 then
128         noise = randomFloatNumber(-2,2)
129     end
130
131
132     -- add theta to signals
133     if sensor_target_result == 1 then
134         reflex_sig = robot_target_omega
135     else
136         reflex_sig = 0
137     end
138
139     x0 = reflex_sig          -- Reflex signal
140     x1 = robot_target_omega -- predictive signal
141
142 -----
143 -- Student Edit code Here
144 -----
145
146
147
148
149
150
151     -- add noise to output
152     output_angle = neural_output + noise*math.max(0, (1-w1))
153
154     -- ====== drive to the robot wheel ======
155     -- forward velocity
156     v0 = 2
157     wheel_left = -output_angle + v0
158     wheel_right = output_angle + v0
159
160
161 end -- ICO controller
162
163

```

Edit code only in ICO_controller function

Get predict and reflex signals

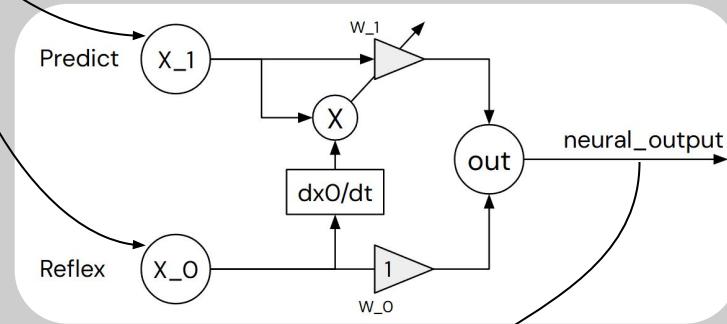
Add ICO equation here

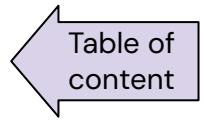
add noise to neural output

drive the output to the robot wheels with forward movement

Hint 1

```
141 x0 = reflex_sig          -- Reflex signal
142 x1 = robot_target_omega -- predictive signal
143
144 -----
145 Student Edit code Here
146 -----
147
148
149 Predict   X_1 -->| w_1 |---->| out |
150           |           |           |
151           |           |           |-----> neural_output
152           |           |           |
153           |           |           |----->| dx0/dt |
154           |           |           |           |----->| 1 |
155           |           |           |           |----->| w_0 |----->| X |
156           |           |           |           |----->| X_0 |----->| Reflex |
157
158 -----
159 -----
160
161 -- add noise to output
162 output_angle = neural_output + noise*math.max(0, (1-wl))
163
```



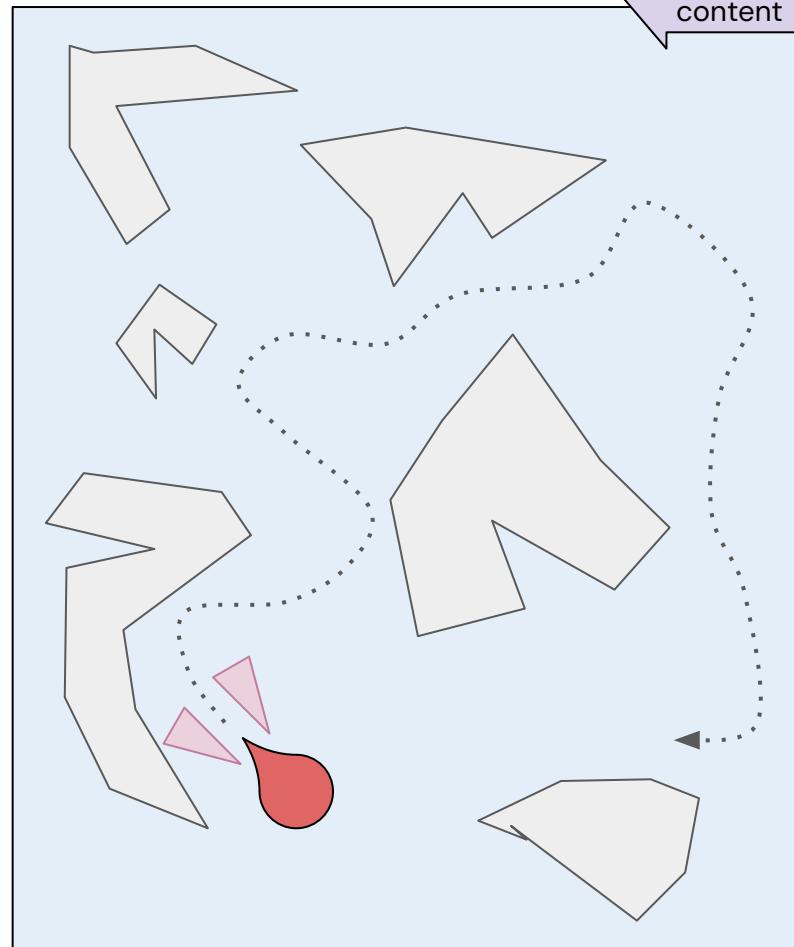
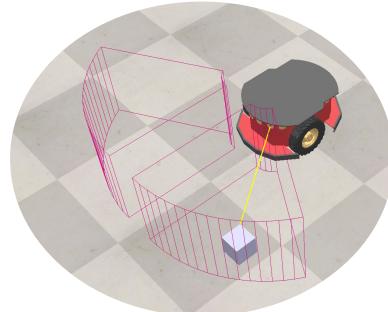


Q-learning

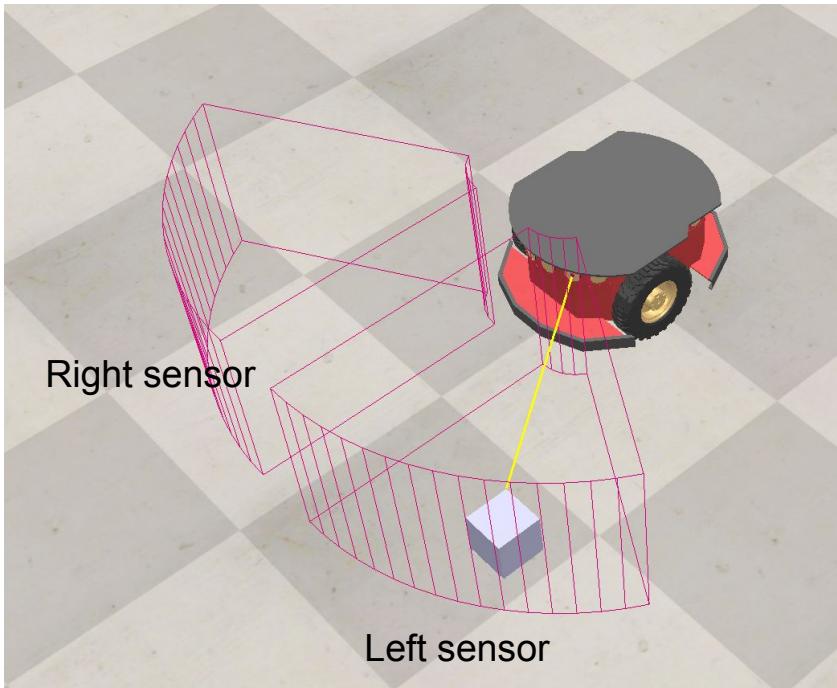
Q-learning task

The robot has ultrasonic sensors on left and right side (see next slide). If a sensor detects an obstacle (obstacle in sensor area), its value is one, if not, the value is zero.

Your task is Use Q-learning for controlling the robot avoid hitting the obstacle.

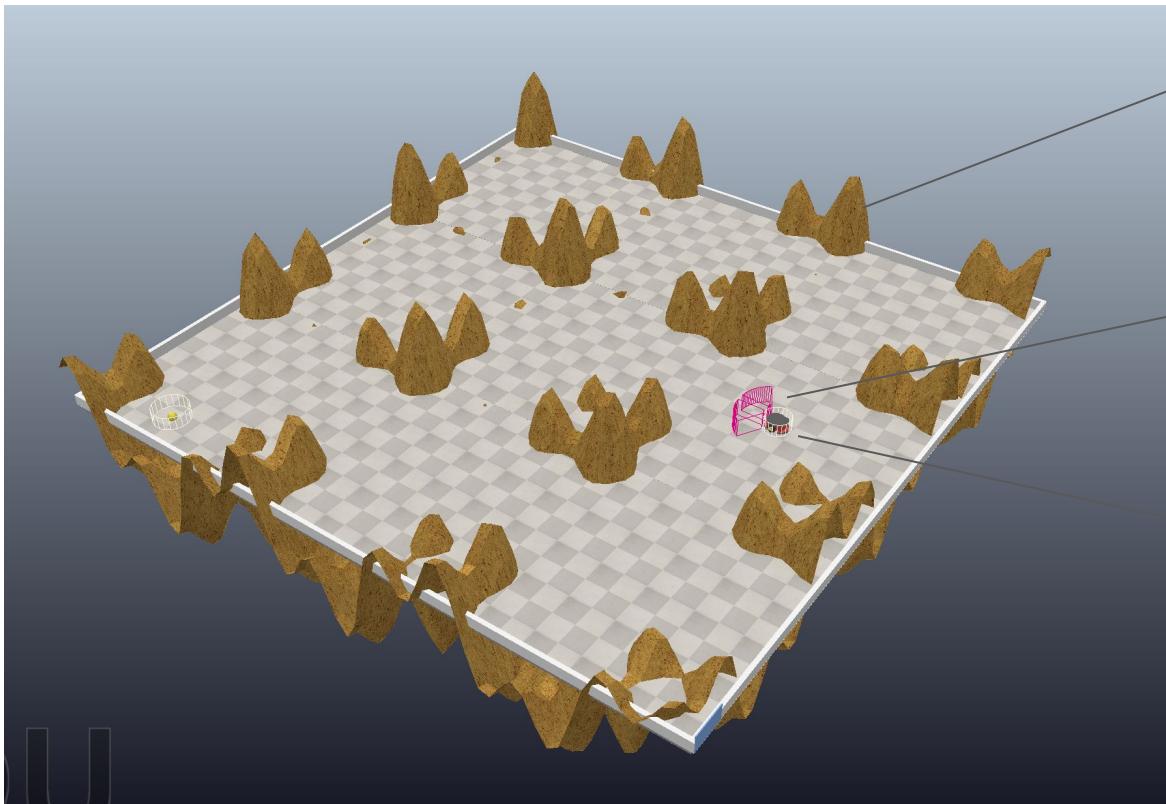


2-wheeled robot

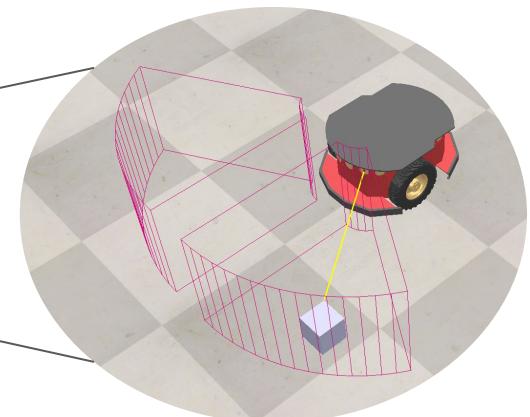


The wheeled robot consists of two wheels and two ultrasonic sensors (left and right sensors). **The red lines** indicate detecting area, if there is an object being in the area, you can see **a yellow line** pointing to the object. In this case, the left sensor value is equal to one, while the right sensor value is zero.

Scene in Simulation



Obstacle



Robot with sensors

Global parameter initiation

```
71  -- #####  
72  -- Q-Learning or SARSA  
73  -- #####  
74  
75  -----  
76  --      Student Initial parameter Here  
77  -----  
78  
79  -- state|      action (move)  
80  -- | forward | left | right | back|  
81  -- {0,0} | 0     | 0    | 0    | 0  |  
82  -- {0,1} | 0     | 0    | 0    | 0  |  
83  -- {1,0} | 0     | 0    | 0    | 0  |  
84  -- {1,1} | 0     | 0    | 0    | 0  |  
85  
86  Q = {{0,0,0,0},  
87  -- {0,0,0,0},  
88  -- {0,0,0,0},  
89  -- {0,0,0,0}}}  
90  
91  
92  
93  alpha = 0.1 -- learning rate  
94  gamma = 0.9 -- discount factor  
95  epsilon = 0.1 --exploration rate  
96  reward = 0  
97  max_episodes = 100  
98  current_episodes = 0  
99  
100 state = 0  
101 next_state = 0  
102 action = 0  
103  
104 -----  
105 -----  
106  
107 end -- sysCall_init
```

For example

Initial parameters in
this zone

Hint: Create Array in Lua code

```
Array_2D = { {1, 2, 3},  
             {4,5, 6, } }
```

Editable code in sysCall_thread()

```
116 function sysCall_thread()
117
118     local dt = 0.2 -- 200 ms
119     while sim.getSimulationState() ~= sim.simulation_advancing_abouttostop do
120         -- =====
121         -- Student Edit code Here
122         -- =====
123
124         -- ===== 1. READ SENSORS =====
125         sensor_left, sensor_right = get_sensor()
126         print(sensor_left, sensor_right)
127
128         -- ===== 2. SELECT ACTION =====
129
130
131         -- ===== 3. EXECUTE ACTION =====
132
133
134         -- ===== 4. WAIT CONTROL PERIOD =====
135         sim.wait(dt) -- Don't delete this
136         -- =====
137
138         -- ===== 5. OBSERVE NEXT STATE & REWARD =====
139
140
141         -- ===== 6. Q-LEARNING UPDATE =====
142
143
144         -- =====
145         -- =====
146
147     end
148 
```

You have to code only in this while-loop in **the sysCall_thread()** function. This loop will be run every 200 ms. You don't need to create any for-loop.

We have provided you a procedural step 1 - 6, which you can follow or not.

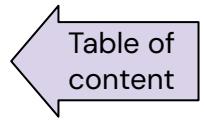
You can create a new function for operating in this while-loop such as `get_action()`, and `get_state()`.

Create custom function

```
155 -----  
156 -- Student Create Function Here  
157 -----  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172 |  
173 -----  
174 function example_create_function(a,b,c)  
175     print(a,b,c)  
176     return a+b+c, a-b-c  
177 end  
178
```

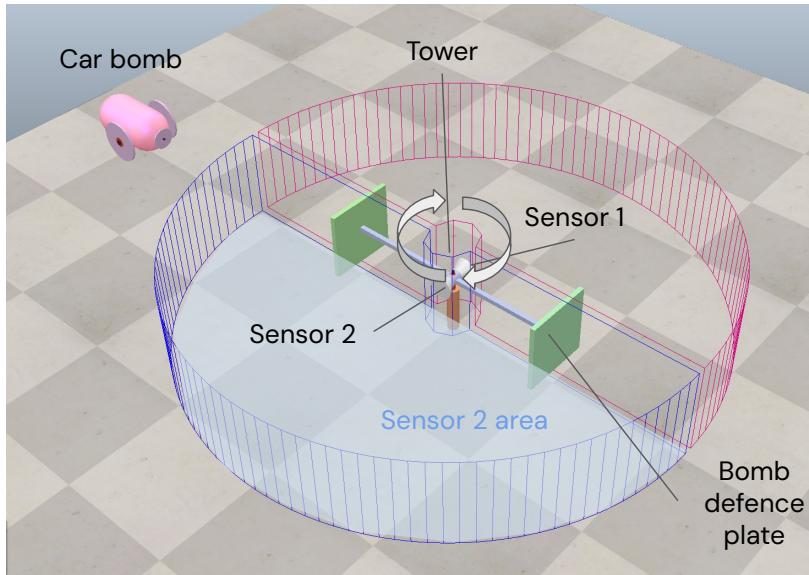
Example Function with
two returning values

You can create a new function and use it in this while-loop such as `get_action()`, and `get_state()`.



XOR tower defence

Short story for tower defence



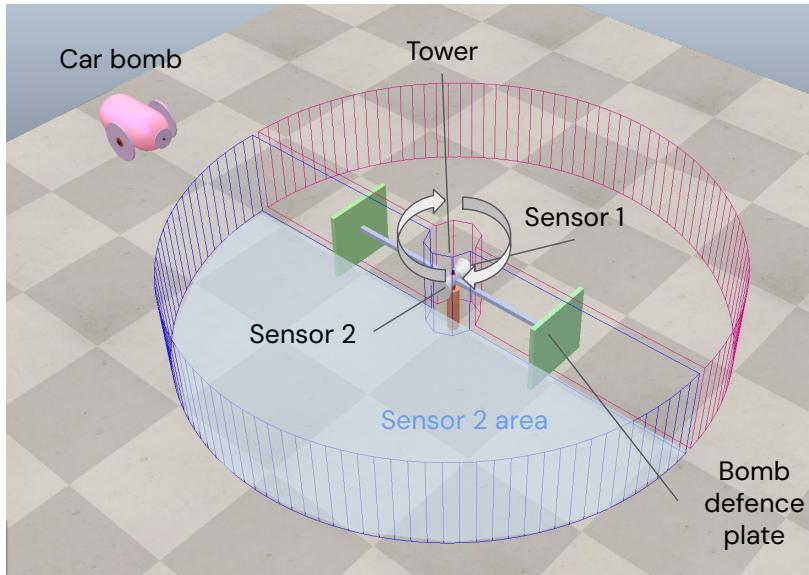
During World War, enemy forces invaded your territory and attempted to destroy a critical communication tower used to contact allied intelligence units. To stop this, the enemy sent explosive vehicles toward the tower.

Due to limited time and resources, you built two small movable titanium walls that can withstand explosions. Using a one-directional motor salvaged from a damaged fighter aircraft, the walls rotate around the tower to block incoming explosives.

Two damaged radars are installed on the tower. Although each radar can scan up to 180 degrees, they can only detect the presence of an object, not its exact position.

Your task is to develop a neural network that controls the rotating walls using only the limited input from the two radars to protect the communication tower.

Short story for tower defence



The tower consists of one directional motor, two ultrasonic sensors with 180 degrees range detection, and two bomb defence plates for stopping and bombing the car bomb.

Your task is rotate the bomb defence plate to the car bomb direction to protect the tower by using XOR logic.

XOR logic

Hint:

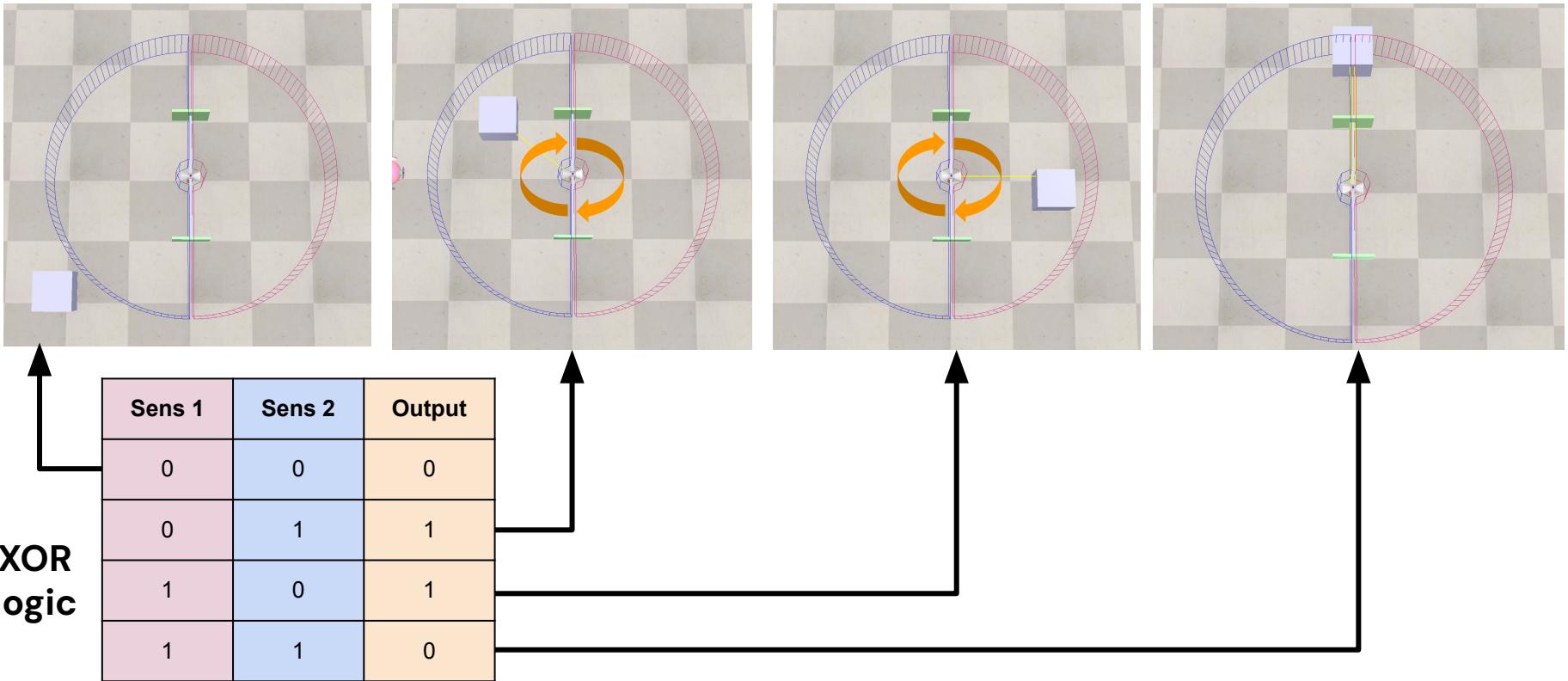
The motor should stop when both sensor can detect object simultaneously

||

The plate will fit perfectly in front of the car bomb.

Sens 1	Sens 2	Output
0	0	0
0	1	1
1	0	1
1	1	0

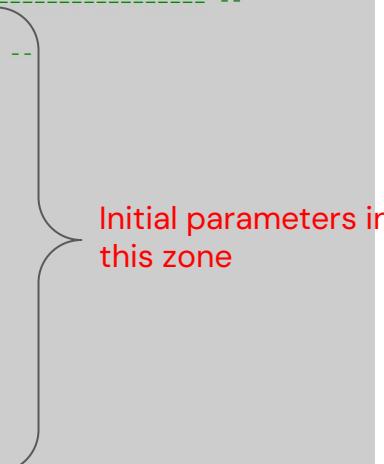
XOR logic with tower action



CoppeliaSim code (Parameter Initiation)

Parameter initiation

```
13
14
15      -- ====== Student Initial parameter Here ===== --
16
17      -- ===== XOR ===== --
18
19      -- A, B, Output of XOR logic
20
21      training_data = {{0,0,0},
22          {0,1,1},
23          {1,0,1},
24          {1,1,0}}
25
26      training = true
27
28
29
30
31
32
33
34
35      -- ===== End of SysCall Initial
36
37 end -- End of SysCall Initial
38
```



Initial parameters in
this zone

CoppeliaSim code (Training and testing functions)

```
43 function XOR_training()
44     -- Neural
45     -- [In1]---w_1-----\           Target
46     --   \                   \
47     --   W_2\                 \
48     --       [h1]---W_5---> [Out]--->(+)--> error
49     --       /   /           /   /
50     --   W_3/   W_6         /   /W_7
51     --   /   /           /   /
52     -- [In2]---/---W_4-----/           \
53     --   /           /           \
54     --   [B1]         [B2]---/
55
56
57
58     -- ===== Student Edit code Here =====
59
60
61
62 for epoch = 1, 10000 do
63     -- print(sen_val_1, sen_val_2)
64     for data_i=1 , 4 do
65
66         -- Neural output equation --
67
68         -- Find Error & delta error --
69
70         -- Update weights --
71
72     end -- each data
73     if epoch % 1000 == 0 then
74         print('epoch = ' .. epoch)
75     end
76 end -- each epoch
77
78     -- ===== Student Edit code Here =====
79
80
81 end -- end of XOR_training function
82
```

Code here for
training neural
network

```
85 function XOR_testing(input_1, input_2)
86     neural_output = 0
87     -- ===== Student Edit code Here =====
88
89     -- Neural equation with updated weights
90
91     -- Example output, student should delete thses when do the task
92     if input_1 == input_2 then
93         neural_output = 0
94     else
95         neural_output = 1
96     end
97
98
99
100    -- ===== Student Edit code Here =====
101
102
103
104    return neural_output
105
106
107 end -- end of XOR_testing function
108
```

Code here for
Testing neural
network

CoppeliaSim code (driving neural output)

You don't need to edit these function, just understand what they do in the program

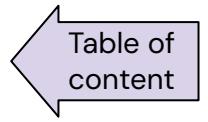
```
111 -- This funciton will enable the XOR_testing fucntion
112 -- when XOR_training funciton have finished training process
113
114 function sysCall_thread()
115   if training == true then
116     print("training")
117     XOR_training()
118     training = false
119   end
120 end
121
122
123
```

```
158 -----
159 -----
160
161 function sysCall_actuation()
162   -- When training process end, XOR_testing function will be executed
163   drive = 0
164   if training == false then
165     drive = XOR_testing(sen_result_1, sen_result_2)
166   end
167
168   -- Drive to the motor
169   sim.setJointTargetVelocity(joint, drive*0.5)
170
171
172
```

This function will enable XOR_testing function after training process finishes

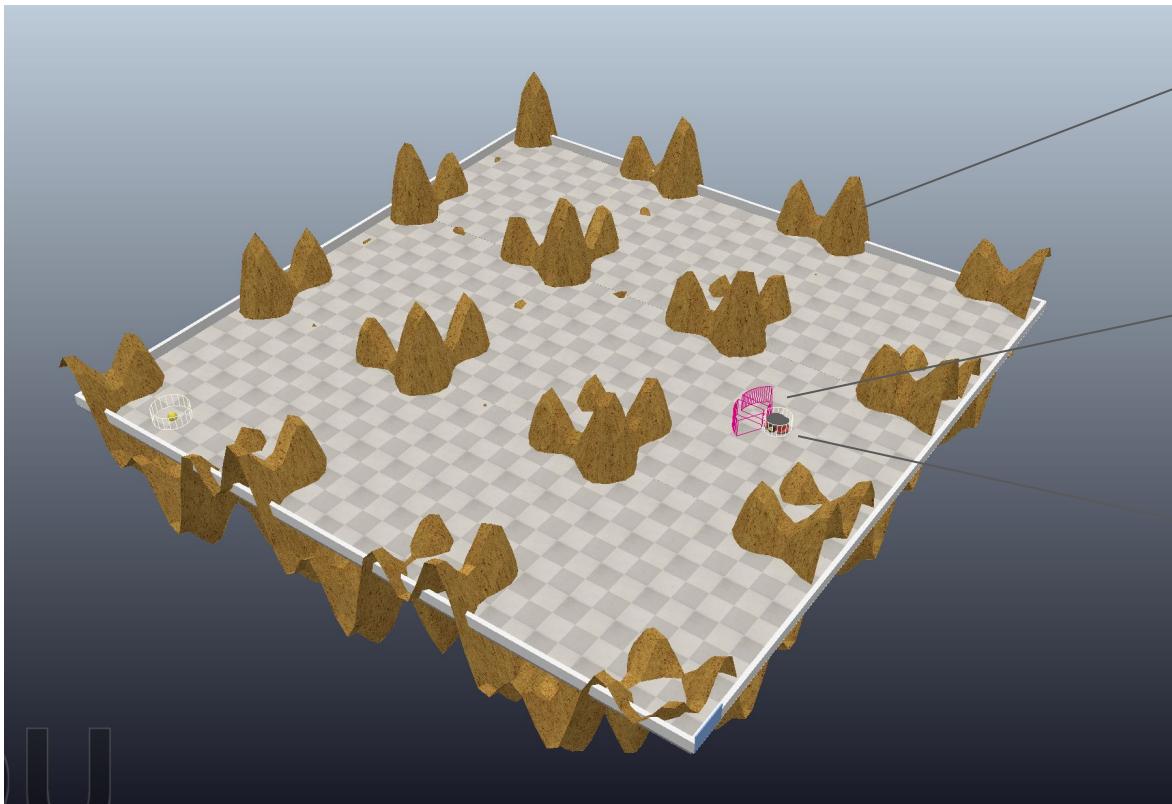
This function execute XOR_testing and drive the neural output to the motor

End

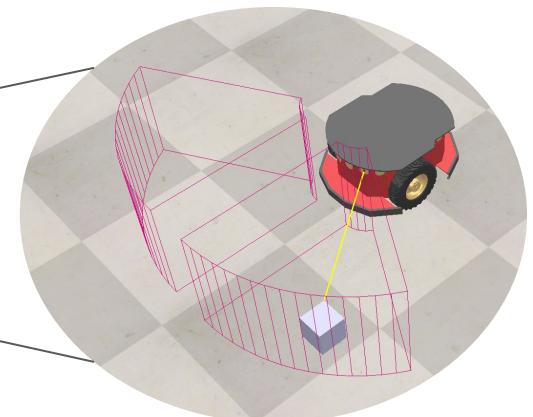


SARSA

Scene in Simulation



Obstacle



Robot with sensors

ICO Answer

