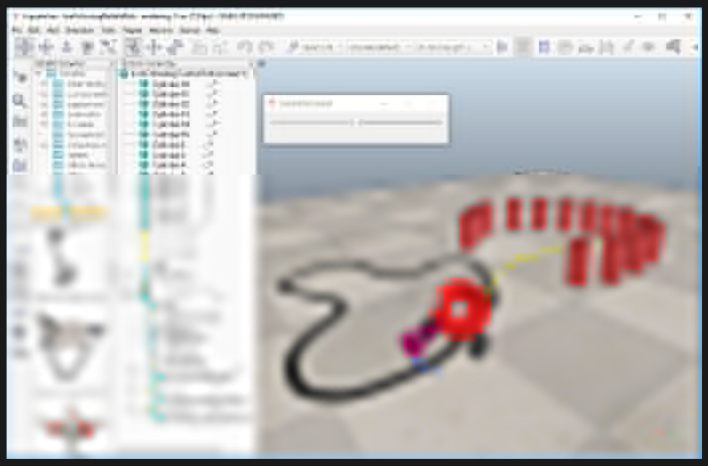
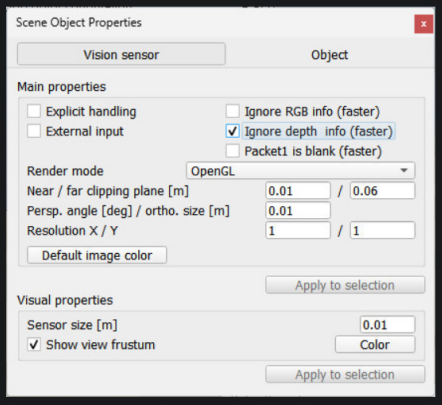
Line following BubbleRob tutorial

In this tutorial we aim at extending the functionality of BubbleRob to let it follow a line on the ground. Make sure you have fully read and understood the first BubbleRob tutorial. This tutorial is courtesy of Eric Rohmer.

Load the scene of the first BubbleRob tutorial located in *scenes/tutorials/BubbleRob*. The scene file related to this tutorial is located in *scenes/tutorials/LineFollowingBubbleRob*. Following figure illustrates the simulation scene that we will design:

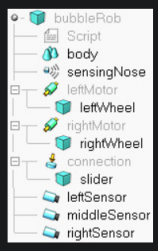


We first create the first of 3 vision sensors that we will attach to the *bubbleRob* object. Select [Add > Vision sensor > Orthographic type]. Edit its properties, by double-clicking on the newly created vision sensor icon in the scene hierarchy, and change the parameters to reflect following dialog:



The vision sensor has to be facing the ground, so select it, and in the orientation dialog, on the **orientation** tab, set [180;0;0] for the *Alpha-Beta-Gamma* items.

We have several possibilities to read a vision sensor. Since our vision sensor has just one pixel and operates in an easy way, we will simply query the average intensity value of the image read by our vision sensor. For more complex cases, we could have set-up a vision callback function. Now copy and paste the vision sensor twice, and adjust its aliases to *eftSensor*, *middleSensor* and *rightSensor*. Make *bubbleRob* their parent (i.e. attach them to the *bubbleRob* object). Your sensors should now look like this in the scene hierarchy:



Let's position the sensors correctly. For that use the position dialog, on the **position** tab, and set following absolute coordinates:

- left sensor: [0.2;0.042;0.018]
- middle sensor: [0.2;0;0.018]
- right sensor: [0.2;-0.042;0.018]

Now let's modify the environment. We can remove a few cylinders in front of BubbleRob. Next, we will build the path that the robot will try to follow: click [Add > Path > Closed]. You have several possibilities to adjust the shape of the path, by manipulating its control points: you can delete or duplicate them, and you

can shift/reorient them. Enable the object movement with the mouse, and adjust the path to your liking.

Once you are satisfied with the geometry of the path (you can always modify it at a later stage), open the customization script attached to it and replace its content with:

```
--lua

sim = require('sim')
path = require('models.path_customization')

function path.shaping(path, pathIsClosed, upVector)
    local section = {-0.02, 0.001, 0.02, 0.001}
    local color = {0.3, 0.3, 0.3}
    local options = 0
    if pathIsClosed then
        options = options | 4
    end
    local shape = sim.generateShapeFromPath(path, section, options, upVector)
    sim.setShapeColor(shape, nil, sim.colorcomponent_ambient_diffuse, color)
    return shape
end
```

Restart the customization script for the changes to take effect, then open the path's user configuration dialog and check the **Generate extruded shape** checkbox.

The last step is to adjust the controller of BubbleRob, so that it will also follow the black path. Open the simulation script attached to *bubbleRob*, and replace it with following code:

Python Lua

```
#python

import math

# Initialization
def sysCall_init():
    sim = require('sim')
    simUI = require('simUI')

    self.bubbleRobBase = sim.getObject('..')
    self.leftMotor = sim.getObject("../leftMotor")
    self.rightMotor = sim.getObject("../rightMotor")
    self.noseSensor = sim.getObject("../sensingNose")

    self.minMaxSpeed = [50 * math.pi / 180, 300 * math.pi / 180]
    self.backUntilTime = -1

    self.floorSensorHandles = [-1, -1, -1]
    self.floorSensorHandles[0] = sim.getObject("../leftSensor")
    self.floorSensorHandles[1] = sim.getObject("../middleSensor")
    self.floorSensorHandles[2] = sim.getObject("../rightSensor")

    self.robotTrace = sim.addDrawingObject(
        sim.drawing_linestrip + sim.drawing_cyclic, 2, 0, -1, 200,
        [1, 1, 0], None, None, [1, 1, 0]
    )

    xml = f'<ui title="{sim.getObjectAlias(self.bubbleRobBase, 1)} speed" closeable="false" ' + \
      'resizeable="false" activate="false"> <hslider minimum="0" ' + \
      'maximum="100" on-change="speedChange_callback" id="1"/>' + \
      '<label text="" style="* {{margin-left: 300px;}}"/>' + \
      '</ui>'

    self.ui = simUI.create(xml)

    self.speed = (self.minMaxSpeed[0] + self.minMaxSpeed[1]) * 0.5
    simUI.setSliderValue(
        self.ui, 1,
        100 * (self.speed - self.minMaxSpeed[0]) / (self.minMaxSpeed[1] - self.minMaxSpeed[0])
    )

# Sensing callback
def sysCall_sensing():
    p = sim.getObjectPosition(self.bubbleRobBase)
    sim.addDrawingObjectItem(self.robotTrace, p)

# UI callback
def speedChange_callback(ui, id, newVal):
    self.speed = self.minMaxSpeed[0] + (self.minMaxSpeed[1] - self.minMaxSpeed[0]) * newVal / 100

# Actuation callback
def sysCall_actuation():
    result, *_ = sim.readProximitySensor(self.noseSensor)
    if result > 0:
        self.backUntilTime = sim.getSimulationTime() + 4

    sensorReading = [False, False, False]
    for i in range(3):
        result = sim.readVisionSensor(self.floorSensorHandles[i])
        if isinstance(result, tuple):
            result, data, *_ = result
        if result >= 0:
            sensorReading[i] = (data[10] < 0.5)  # Indexing adjusted for 0-based Python

    rightV = self.speed
    leftV = self.speed
    if sensorReading[0]:
        leftV = 0.03 * self.speed
    if sensorReading[2]:
        rightV = 0.03 * self.speed
    if sensorReading[0] and sensorReading[2]:
        self.backUntilTime = sim.getSimulationTime() + 2

    if self.backUntilTime < sim.getSimulationTime():
        sim.setJointTargetVelocity(self.leftMotor, leftV)
        sim.setJointTargetVelocity(self.rightMotor, rightV)
    else:
        sim.setJointTargetVelocity(self.leftMotor, -self.speed / 2)
        sim.setJointTargetVelocity(self.rightMotor, -self.speed / 8)

# Cleanup callback
def sysCall_cleanup():
    simUI.destroy(self.ui)
```

You can easily debug your line following vision sensors: select one, then in the scene view select [Right-click --> Add --> Floating view], then in the newly added floating view select [Right click --> View --> Associate view with selected vision sensor].