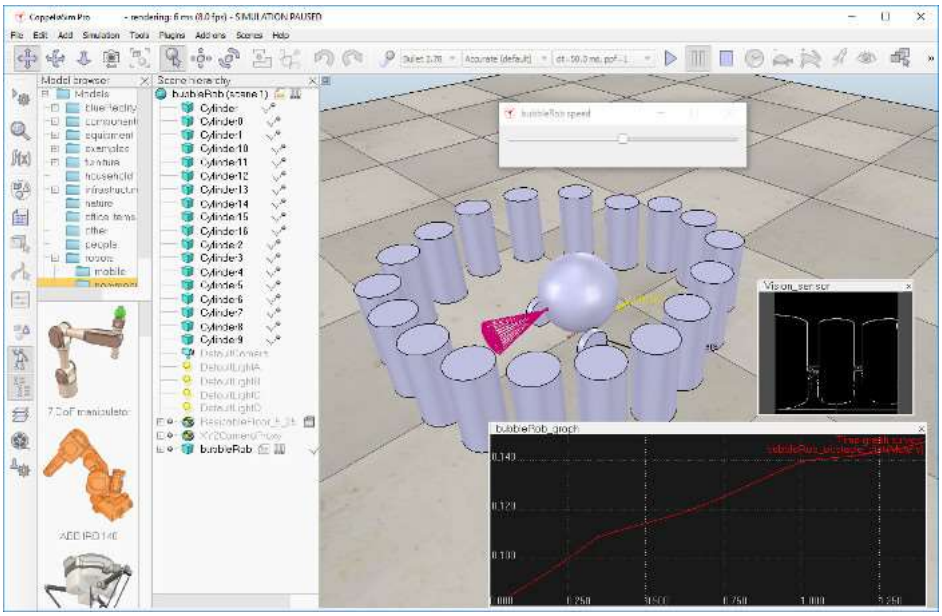
BubbleRob tutorial

This tutorial will try to introduce quite many CoppeliaSim functionalities while designing the simple mobile robot *BubbleRob*. The CoppeliaSim s tutorial is located in *scenes/tutorials/BubbleRob*. Following figure illustrates the simulation scene that we will design:



Since this tutorial will fly over many different aspects, make sure to also have a look at the other tutorials, mainly the tutorial about building a of all, freshly start CoppeliaSim. The simulator displays a default scene. We will start with the body of *BubbleRob*.

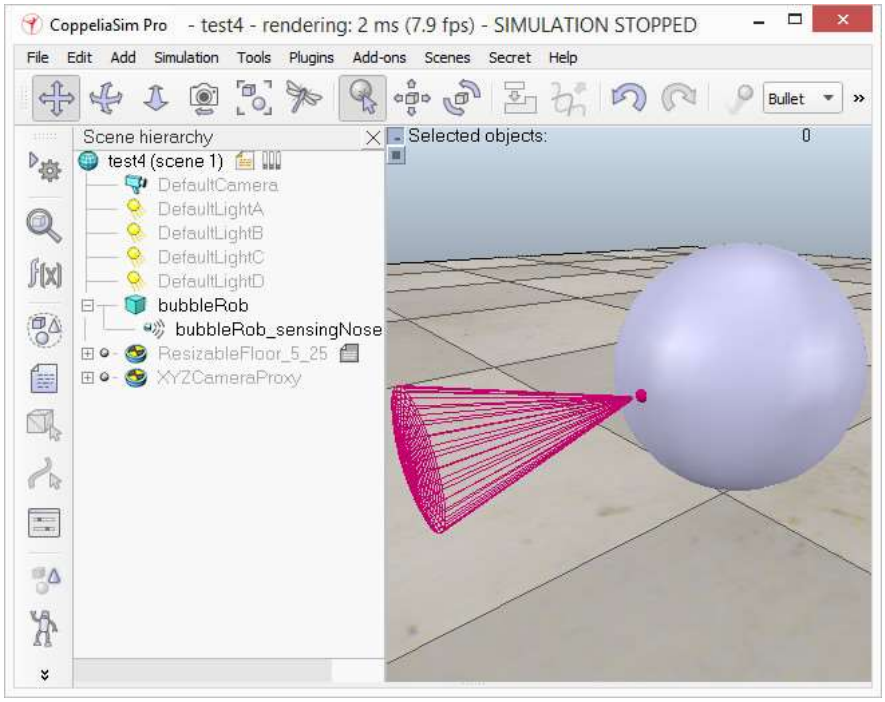
We add a primitive sphere of diameter 0.2 to the scene with [Menu bar --> Add --> Primitive shape --> Sphere]. We adjust the **X-size** item tc The created sphere will appear in the visibility layer 1 by default, and be dynamic and respondable (since we kept the item **Create dynamic a shape** enabled). This means that *BubbleRob's* body will be falling and able to react to collisions with other respondable shapes (i.e. simulated engine). We can see this is the shape dynamics properties: items **Body is respondable** and **Body is dynamic** are enabled. We start the sim button, or by pressing <control-space> in the scene window), and copy-and-paste the created sphere (with [Menu bar --> Edit --> Copy selec [Menu bar --> Edit -> Paste buffer], or with <control-c> then <control-v>): the two spheres will react to collision and roll away. We stop the s duplicated sphere will automatically be removed. This default behaviour can be modified in the simulation dialog.

We also want the *BubbleRob's* body to by usable by the other calculation modules (e.g. distance calculation). For that reason, we enable **Colli Renderable** and **Detectable** in the object common properties for that shape, if not already enabled. If we wanted, we could now also chang appearance of our sphere in the shape properties.

Now we open the position dialog on the **translation** tab, select the sphere representing *BubbleRob's* body, and enter 0.02 for **Along Z.** We m **Relative to**-item is set to **World.** Then we click **Translate selection.** This translates all selected objects by 2 cm along the absolute Z-axis, our sphere a little bit. In the scene hierarchy, we double-click the sphere's name, so that we can edit its name. We enter *bubbleRob* and press

Next we will add a proximity sensor so that *BubbleRob* knows when it is approaching obstacles: we select [Menu bar --> Add --> Proximity ser the orientation dialog on the **orientation** tab, we enter 90 for **Around Y** and for **Around Z**, then click **Rotate selection.** In the position dial tab, we enter 0.1 for **X-coord.** and 0.12 for **Z-coord.** The proximity sensor is now correctly positioned relative to *BubbleRob's* body. We doub sensor's icon in the scene hierarchy to open its properties dialog. We click **Show volume parameter** to open the proximity sensor volume dia **Offset** to 0.005, **Angle** to 30 and **Range** to 0.15. Then, in the proximity sensor properties, we click **Show detection parameters.** This ope detection parameter dialog. We uncheck item **Don't allow detections if distance smaller than** then close that dialog again. In the scene h click the proximity sensor's name, so that we can edit its name. We enter *bubbleRob_sensingNose* and press enter.

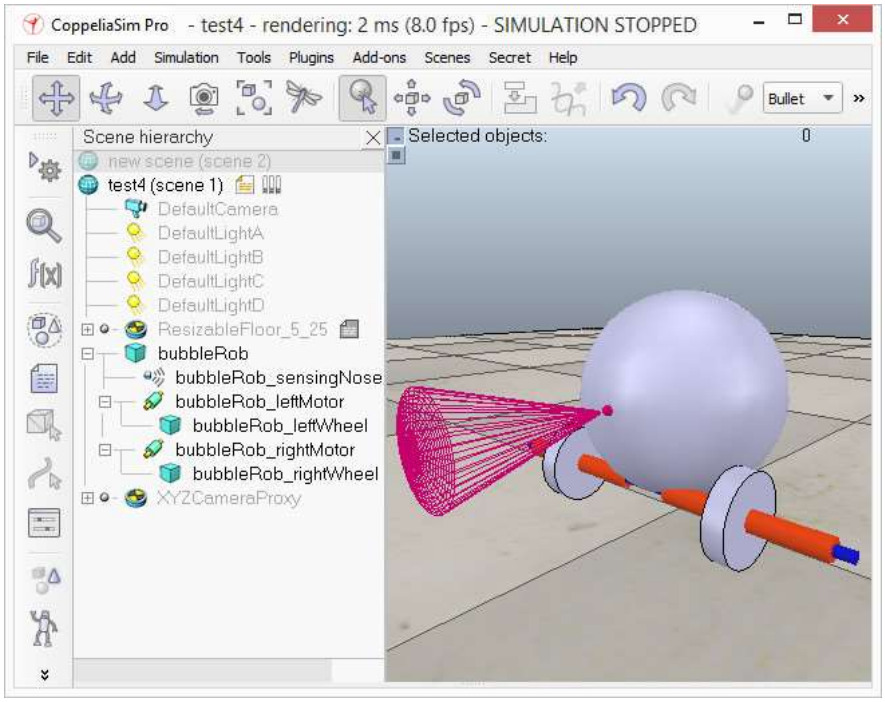
We select *bubbleRob_sensingNose*, then control-select *bubbleRob*, then click [Menu bar --> Edit --> Make last selected object parent]. This at the body of the robot. We could also have dragged *bubbleRob_sensingNose* onto *bubbleRob* in the scene hierarchy. This is what we now have:



[Proximity sensor attached to *bubbleRob's* body]

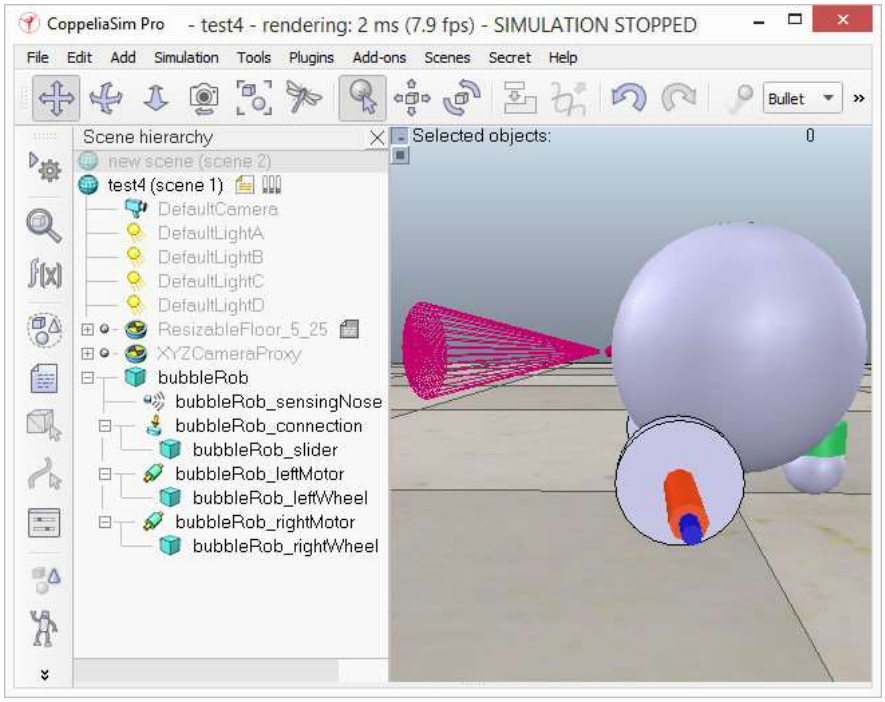
Next we will take care of *BubbleRob's* wheels. We create a new scene with [Menu bar --> File --> New scene]. It is often very convenient to w scenes, in order to visualize and work only on specific elements. We add a pure primitive cylinder with dimensions (0.08,0.08,0.02). As for the we enable Collidable, Measurable, Renderable and Detectable in the object common properties for that cylinder, if not already enabled. Then w absolute position to (0.05,0.1,0.04) and its absolute orientation to (-90,0,0). We change the name to *bubbleRob_leftWheel*. We copy and past the absolute Y coordinate of the copy to -0.1. We rename the copy to *bubbleRob_rightWheel*. We select the two wheels, copy them, then swit then paste the wheels.

We now need to add joints (or motors) for the wheels. We click [Menu bar --> Add --> Joint --> Revolute] to add a revolute joint to the scene when adding a new object to the scene, the object will appear at the origin of the world. We Keep the joint selected, then control-select *bubbl* position dialog, on the **position** tab, we click the **Apply to selection** button: this positioned the joint at the center of the left wheel. Then, in on the **orientation** tab, we do the same: this oriented the joint in the same way as the left wheel. We rename the joint to *bubbleRob_leftMot* click the joint's icon in the scene hierarchy to open the joint properties dialog. Then we click **Show dynamic parameters** to open the joint d dialog. We **enable the motor**, and check item **Lock motor when target velocity is zero**. We now repeat the same procedure for the right to *bubbleRob_rightMotor*. Now we attach the left wheel to the left motor, the right wheel to the right motor, then attach the two motors to *bul* we have:



[Proximity sensor, motors and wheels]

We run the simulation and notice that the robot is falling backwards. We are still missing a third contact point to the floor. We now add a small new scene we and add a pure primitive sphere with diameter 0.05 and make the sphere Collidable, Measurable, Renderable and Detectable (if then rename it to *bubbleRob_slider*. We set the **Material** to *noFrictionMaterial* in the shape dynamics properties. To rigidly link the slider with we add a force sensor object with [Menu bar --> Add --> Force sensor]. We rename it to *bubbleRob_connection* and shift it up by 0.05. We at force sensor, then copy both objects, switch back to scene 1 and paste them. We then shift the force sensor by -0.07 along the absolute X-axis robot body. If we run the simulation now, we can notice that the slider is slightly moving in relation to the robot body: this is because both obj *bubbleRob_slider* and *bubbleRob*) are colliding with each other. To avoid strange effects during dynamics simulation, we have to inform Coppel do not mutually collide, and we do this in following way: in the shape dynamics properties, for *bubbleRob_slider* we set the **local respondabl** and for *bubbleRob*, we set the **local respondable mask** to 11110000. If we run the simulation again, we can notice that both objects do not This is what we now have:



[Proximity sensor, motors, wheels and slider]

We run the simulation again and notice that *BubbleRob* slightly moves, even with locked motor. We also try to run the simulation with different result will be different. Stability of dynamic simulations is tightly linked to masses and inertias of the involved non-static shapes. For an explana make sure to carefully read this and that sections. We now try to correct for that undesired effect. We select the two wheels and the slider, an dynamics dialog we click three times **M=M*2 (for selection)**. The effect is that all selected shapes will have their masses multiplied by 8. W inertias of the 3 selected shapes, then run the simulation again: stability has improved. In the joint dynamics dialog, we set the **Target veloci** motors. We run the simulation: *BubbleRob* now moves forward and eventually falls off the floor. We reset the **Target velocity** item to zero fo

The object *bubbleRob* is at the base of all objects that will later form the *BubbleRob* model. We will define the model a little bit later. Next we a graph object to *BubbleRob* in order to display its clearance distance. We click [Menu bar --> Add --> Graph] and rename it to *bubbleRob_grap* to *bubbleRob*, and set the graph's absolute coordinates to (0,0,0.005).

Now we set one motor **target velocity** to 50, run the simulation, and will see *BubbleRob's* trajectory displayed in the scene. We then stop the the motor target velocity to zero.

We add a pure primitive cylinder with following dimensions: (0.1, 0.1, 0.2). We want this cylinder to be static (i.e. not influenced by gravity or exerting some collision responses on non-static respondable shapes. For this, we disable **Body is dynamic** in the shape dynamics properties. cylinder to be Collidable, Measurable, Renderable and Detectable. We do this in the object common properties. Now, while the cylinder is still s object translation toolbar button:

Now we can drag any point in the scene: the cylinder will follow the movement while always being constrained to keep the same Z-coordinate. the cylinder a few times, and move them to positions around *BubbleRob* (it is most convenient to perform that while looking at the scene from object shifting, holding down the shift key allows to perform smaller shift steps. Holding down the ctrl key allows to move in an orthogonal dire direction(s). When done, select the camera pan toolbar button again:

We set a **target velocity** of 50 for the left motor and run the simulation: the graph view now displays the distance to the closest obstacle anc is visible in the scene too. We stop the simulation and reset the target velocity to zero.

We now need to finish **BubbleRob** as a model definition. We select the model base (i.e. object *bubbleRob*) then check items **Object is mode Object/model can transfer or accept DNA** in the object common properties: there is now a stippled bounding box that encompasses all o hierarchy. We select the two joints, the proximity sensor and the graph, then enable item **Ignored by model bounding box** and click **Apply** same dialog: the model bounding box now ignores the two joints and the proximity sensor. Still in the same dialog, we disable **camera visibil** enable **camera visibility layer** 10 for the two joints and the force sensor: this effectively hides the two joints and the force sensor, since laye default. At any time we can modify the visibility layers for the whole scene. To finish the model definition, we select the vision sensor, the two the graph, then enable item **Select base of model instead**: if we now try to select an object in our model in the scene, the whole model wil which is a convenient way to handle and manipulate the whole model as a single object. Additionally, this protects the model against inadverta Individual objects in the model can still be selected in the scene by click-selecting them with control-shift, or normally selecting them in the sc finally collapse the model tree in the scene hierarchy.

Next we will add a vision sensor, at the same position and orientation as *BubbleRob's* proximity sensor. We open the model hierarchy again, th Add --> Vision sensor --> Perspective type], then attach the vision sensor to the proximity sensor, and set the local position and orientation of (0,0,0). We also make sure the vision sensor is not not visible, not part of the model bounding box, and that if clicked, the model will be select customize the vision sensor, we open its properties dialog. We set the **Far clipping plane** item to 1, and the **Resolution x** and **Resolution** 256. We add a floating view to the scene, and over the newly added floating view, right-click [Popup menu --> View --> Associate view with s (we make sure the vision sensor is selected during that process).

We attach a child script to the vision sensor by clicking [Menu bar --> Add --> Associated child script --> Non threaded]. We double-click the i to the vision sensor in the scene hierarchy: this opens the child script that we just added. We copy and paste following code into the script edi

```
function sysCall_vision(inData)
    simVision.sensorImgToWorkImg(inData.handle) -- copy the vision sensor image to the work image
    simVision.edgeDetectionOnWorkImg(inData.handle,0.2) -- perform edge detection on the work image
    simVision.workImgToSensorImg(inData.handle) -- copy the work image to the vision sensor image buffer
end

function sysCall_init()
end
```

To be able to see the vision sensor's image, we start the simulation, then stop it again.

The last thing that we need for our scene is a small child script that will control *BubbleRob's* behavior. We select *bubbleRob* and click [Menu ba Associated child script --> Non threaded]. We double-click the script icon that appeared next to *bubbleRob's* name in the scene hierarchy and following code into the script editor, then close it:

```
function speedChange_callback(ui,id,newVal)
    speed=minMaxSpeed[1]+(minMaxSpeed[2]-minMaxSpeed[1])*newVal/100
end

function sysCall_init()
    -- This is executed exactly once, the first time this script is executed
    bubbleRobBase=sim.getObjectAssociatedWithScript(sim.handle_self) -- this is bubbleRob's handle
    leftMotor=sim.getObjectHandle("bubbleRob_leftMotor") -- Handle of the left motor
    rightMotor=sim.getObjectHandle("bubbleRob_rightMotor") -- Handle of the right motor
    noseSensor=sim.getObjectHandle("bubbleRob_sensingNose") -- Handle of the proximity sensor
    minMaxSpeed={50*math.pi/180,300*math.pi/180} -- Min and max speeds for each motor
    backUntilTime=-1 -- Tells whether bubbleRob is in forward or backward mode
    robotCollection=sim.createCollection(0)
    sim.addItemToCollection(robotCollection,sim.handle_tree,bubbleRobBase,0)
    distanceSegment=sim.addDrawingObject(sim.drawing_lines,4,0,-1,1,{0,1,0})
    robotTrace=sim.addDrawingObject(sim.drawing_linestrip+sim.drawing_cyclic,2,0,-1,200,{1,1,0},nil,nil,{1,1,0})
    graph=sim.getObjectHandle('bubbleRob_graph')
    distStream=sim.addGraphStream(graph,'bubbleRob clearance','m',0,{1,0,0})
    -- Create the custom UI:
        xml = '<ui title="'..sim.getObjectName(bubbleRobBase)..' speed" closeable="false" resizeable="false" activ
        <hslider minimum="0" maximum="100" onchange="speedChange_callback" id="1"/>
        <label text="" style="* {margin-left: 300px;}"/>
        </ui>
        ]]
    ui=simUI.create(xml)
    speed=(minMaxSpeed[1]+minMaxSpeed[2])*0.5
    simUI.setSliderValue(ui,1,100*(speed-minMaxSpeed[1])/(minMaxSpeed[2]-minMaxSpeed[1]))
end

function sysCall_sensing()
    local result,distData=sim.checkDistance(robotCollection,sim.handle_all)
    if result>0 then
```

```
            sim.addDrawingObjectItem(distanceSegment,nil)
            sim.addDrawingObjectItem(distanceSegment,distData)
            sim.setGraphStreamValue(graph,distStream,distData[7])
        end
        local p=sim.getObjectPosition(bubbleRobBase,-1)
        sim.addDrawingObjectItem(robotTrace,p)
    end

    function sysCall_actuation()
        result=sim.readProximitySensor(noseSensor) -- Read the proximity sensor
        -- If we detected something, we set the backward mode:
        if (result>0) then backUntilTime=sim.getSimulationTime()+4 end

        if (backUntilTime<sim.getSimulationTime()) then
            -- When in forward mode, we simply move forward at the desired speed
            sim.setJointTargetVelocity(leftMotor,speed)
            sim.setJointTargetVelocity(rightMotor,speed)
        else
            -- When in backward mode, we simply backup in a curve at reduced speed
            sim.setJointTargetVelocity(leftMotor,-speed/2)
            sim.setJointTargetVelocity(rightMotor,-speed/8)
        end
    end

    function sysCall_cleanup()
            simUI.destroy(ui)
    end
```

We run the simulation. *BubbleRob* now moves forward while trying to avoid obstacles (in a very basic fashion). While the simulation is still runi *BubbleRob's* velocity, and copy/paste it a few times. Also try to scale a few of them while the simulation is still running. Be aware that the mini calculation functionality might be heavily slowing down the simulation, depending on the environment.

Using a script to control a robot or model is only one way of doing. CoppeliaSim offers many different ways (also combined), have a look at th tutorial.