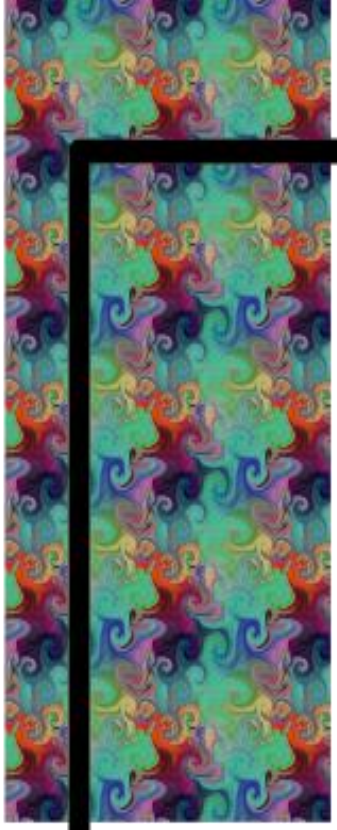


Práctica 1:
Robot en Espacio Cerrado

CE Inteligencia Artificial y Big Data

Modelos de Inteligencia Artificial

2024/2025

Daniel Marín López

Índice

1. Enunciado

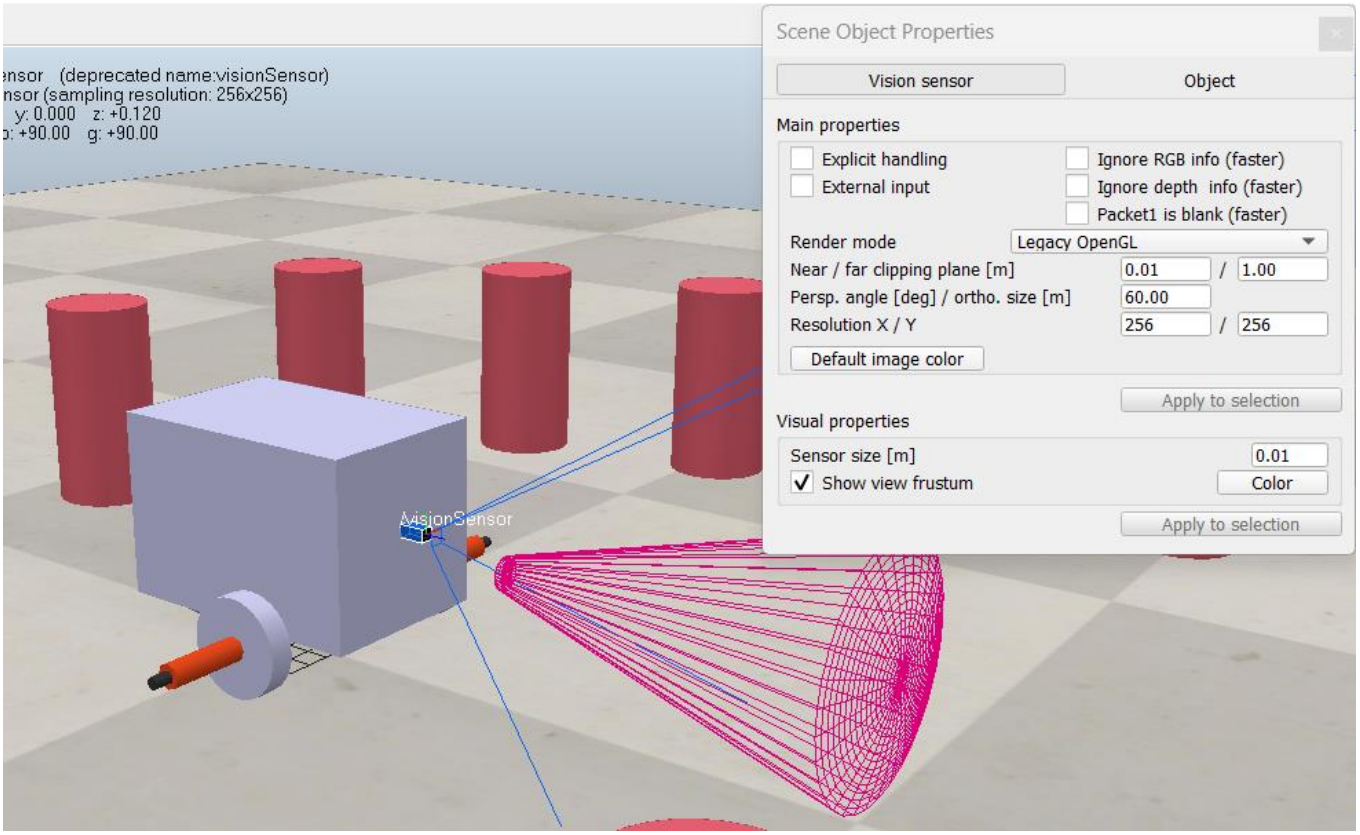
Partiendo de la Actividad 1, diseñar un robot de manera que encuentre la salida en un espacio cerrado, bien recorrer el espacio mínimo entre dos puntos definidos, o seguir una línea sin salirse. Puedes realizar una de las opciones o incluir otra a tu elección (no se permite usar los que tengan el código para descargarlo)

2. Planteamiento del problema

Ahora mismo, nuestro robot solamente cuenta con un sensor que modifica su velocidad cuando detecta un objeto. Existen otros sensores como una cámara con la cual el robot puede tener visión artificial sobre el mundo que le rodea, esto nos puede ayudar no solo viendo lo que ve el robot sino también a que este responda cuando tenga algo delante.

2.1. Cámara del robot

Para agregar una cámara, iremos a ***Add/Vision sensor/Perspective type***. Este sensor debe estar conectado al sensor de proximidad que ya pusimos con anterioridad. Debemos orientar la cámara 90º para que mire al frente. La resolución de la imagen será 256.



Con eso, nuestra cámara ya estaría lista. Pero el problema es que al darle al Play no aparece ninguna ventana con la visión artificial del robot. Para ello, tendremos que codificar nosotros mismos la ventana para la visión.

El código se divide ahora en dos scripts, el de la propia cámara y el del robot (recordemos que los códigos son en Lua debido a que no encuentra el intérprete de Python). El de la cámara es el siguiente:

```
function sysCall_init()
end

function sysCall_vision(inData)
    -- callback function automatically added for backward compatibility
    -- (vision sensor have no filters anymore, but rather a callback function where image
processing can be performed)
    local retVal={}
    retVal.trigger=false
    retVal.packedPackets={}
    simVision.sensorImgToWorkImg(inData.handle)
    simVision.edgeDetectionOnWorkImg(inData.handle,0.200000)
    simVision.workImgToSensorImg(inData.handle,false)
    return retVal
end
```

Este código debe ser un simulation script unido a la cámara. Mientras que el del robot es el siguiente:

```
-- callback para el slider de velocidad
function speedChange_callback(ui, id, newVal)
    speed = minMaxSpeed[1] + (minMaxSpeed[2] - minMaxSpeed[1]) * newVal / 100
end

function sysCall_init()
    self = {}

    -- Handles
    baseHandle = sim.getObjectHandle(sim.handle_self)
    leftMotor = sim.getObjectHandle("RevoluteLeft")
    rightMotor = sim.getObjectHandle("RevoluteRight")
    noseSensor = sim.getObjectHandle("proximitySensor")
    camHandle = sim.getObjectHandle("visionSensor")

    -- Gr?fico
    self.graphHandle = sim.getObjectHandle("/Act1Rob/graph")
    self.streamHandle = sim.addGraphStream(self.graphHandle, 'distancia_libre', 'm', 0, {1,
0, 0})

    -- Velocidades
    minMaxSpeed = { -50 * math.pi / 180, 300 * math.pi / 180 }
    backUntilTime = 0

    -- UI Slider
    local baseName = sim.getObjectName(baseHandle)
    local xmlSpeed = '<ui title="' .. baseName .. ' speed" closeable="false"
resizeable="false" activate="false">' .. [[
        <hslider minimum="0" maximum="100" onchange="speedChange_callback" id="1"/>
        <label text="" style="* {margin-left: 300px;}"/>
```

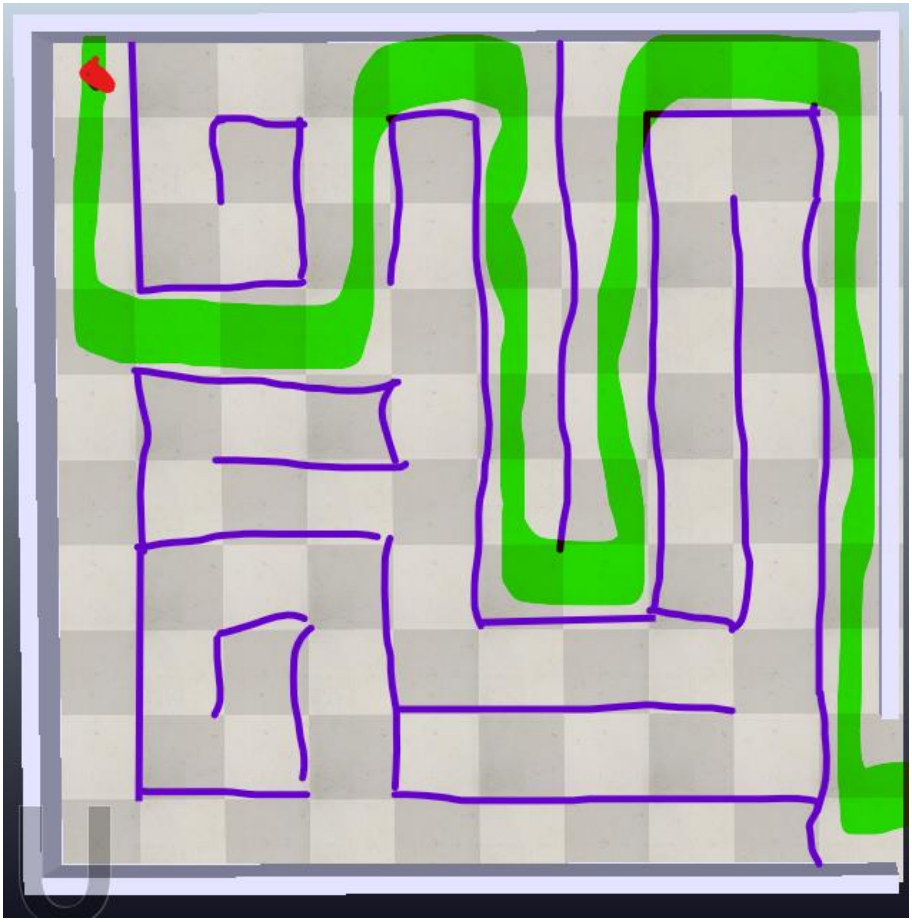
```
    </ui>]]
    uiSpeed = simUI.create(xmlSpeed)

    speed = (minMaxSpeed[1] + minMaxSpeed[2]) * 0.5
    simUI.setSliderValue(uiSpeed, 1, 100 * (speed - minMaxSpeed[1]) / (minMaxSpeed[2] -
minMaxSpeed[1]))

    -- UI C?mara
    local xmlCam = [[
        <ui title="Camera view" closeable="false" modal="false" resizable="false">
            <image id="1"/>
        </ui>
    ]]
    uiCam = simUI.create(xmlCam)
end

function sysCall_actuation()
    local result, dist = sim.readProximitySensor(noseSensor)
    local now = sim.getSimulationTime()

    if result > 0 then
        backUntilTime = now + 4
    end

    if now > backUntilTime then
        sim.setGraphStreamValue(self.graphHandle, self.streamHandle, dist)
        sim.setJointTargetVelocity(leftMotor, speed)
        sim.setJointTargetVelocity(rightMotor, speed)
    else
        sim.setJointTargetVelocity(leftMotor, -speed / 2)
        sim.setJointTargetVelocity(rightMotor, -speed / 8)
    end
end

function sysCall_sensing()
    if camHandle and uiCam then
        local imageBuffer, resolution = sim.getVisionSensorImg(camHandle, 0)
        if imageBuffer then
            simUI.setImageData(uiCam, 1, imageBuffer, resolution[1], resolution[2])
        end
    end
end

function sysCall_cleanup()
    if uiSpeed then simUI.destroy(uiSpeed) end
    if uiCam then simUI.destroy(uiCam) end
end
```

En este script se controla nuestro robot para que avance a una velocidad (del script) definida por una barra en la interfaz de usuario. Si el sensor de proximidad detecta un obstáculo, el robot retrocede brevemente y gira un poco

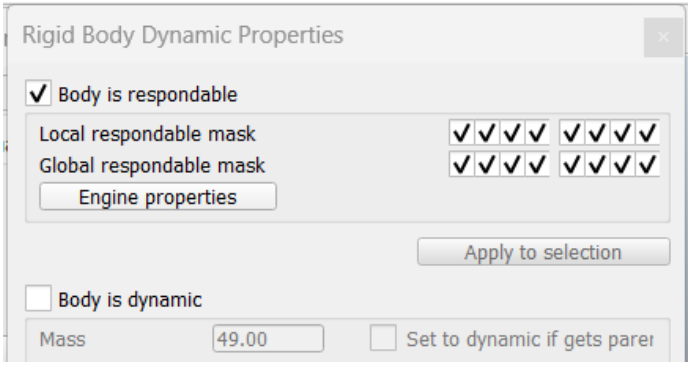
antes de intentar avanzar de nuevo. Además, se muestra la distancia detectada por el sensor en un gráfico y la vista de una cámara en otra ventana de la interfaz de usuario.
Con esto, nuestro robot ya tiene una cámara la cual podemos ver lo que recibe y, por ende, tomar una estrategia de cómo afrontar un problema. Hemos puesto varios cilindros a su alrededor y una clara salida, los resultados fueron que el robot da un giro hasta llegar a la salida y se queda atrapado por culpa del sensor. Necesita un poco de ayuda humana para salir.

3. Desafío: Salir de un espacio cerrado

El problema que se nos plantea es que el robot sea capaz de salir de un espacio cerrado, hemos propuesto como problema que el robot sea capaz de salir de un laberinto. El laberinto planteado es el siguiente:



Un laberinto simple que cubre todo el escenario de coppelia. Para ello, hemos creado un montón de cubos y una vez terminado el laberinto los unimos en un grupo, eso lo hacemos seleccionando todos los cubos y luego en ***Edit/Shape grouping/merging***. También debemos hacer que el laberinto sea estático para que el robot no lo empuje.



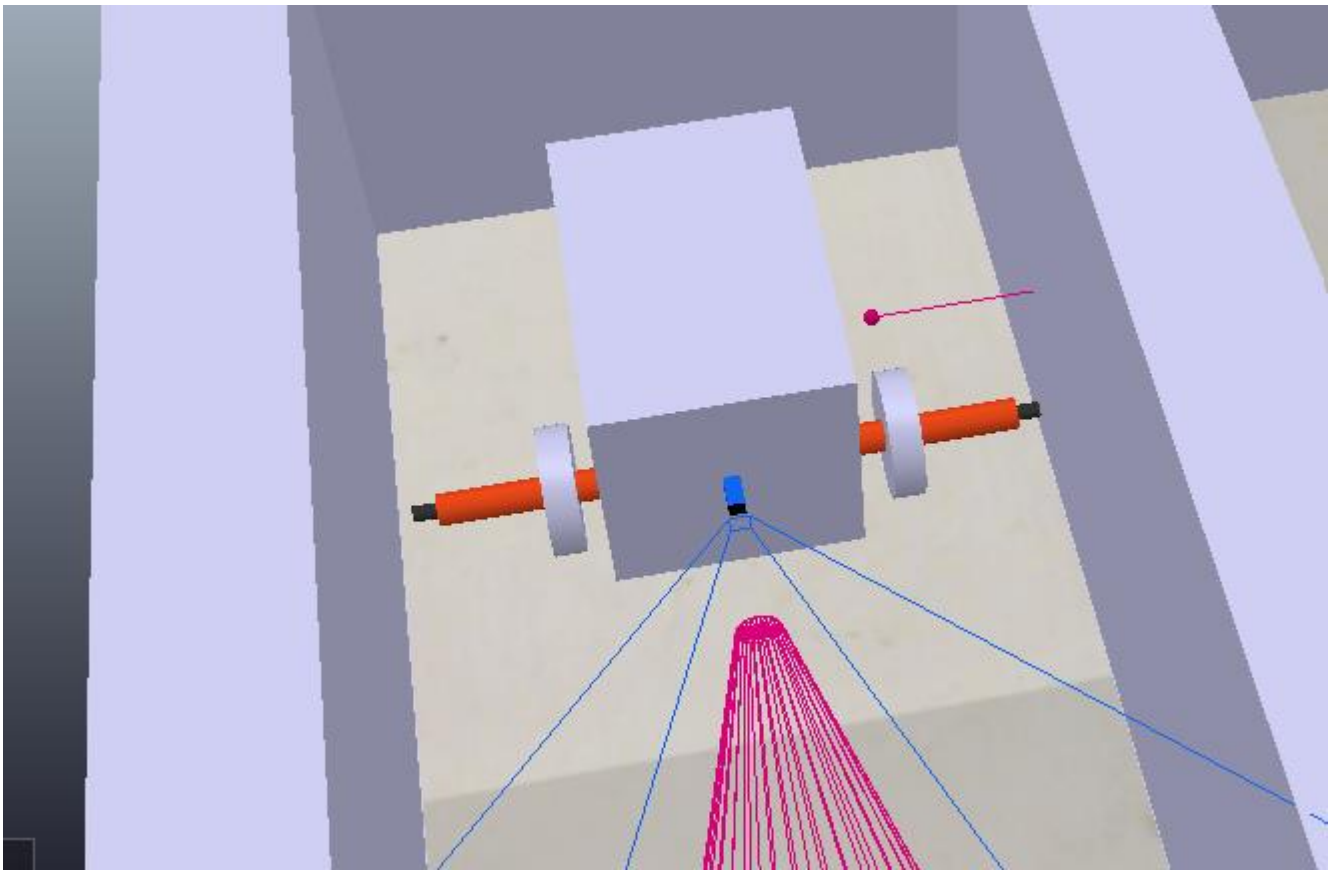
Una vez tenemos nuestro laberinto listo, el problema se puede resolver de 2 formas posibles:

- Mediante un sensor lateral que servirá como poner la mano en la pared.
- Mediante la cámara y una línea trazada en el suelo.

Hemos explorado las dos opciones para luego comparar cual es la que mejor resultado ha dado.

3.1. Sensor lateral

La idea es tener un sensor de proximidad adicional colocado en un lateral de nuestro robot para que simule que está tocando la pared. Para ello, agregaremos un sensor en forma de rayo que esté girado 90º mirando hacia la pared.



En nuestro caso, estará en el lado derecho. Incluso la cámara no sería necesaria en este caso pero la dejaremos para no modificar código. El código sería el siguiente:

```
-- Declaramos una nueva variable global
turnDirection = 1  -- 1: derecha, -1: izquierda

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

function sysCall_init()
    math.randomseed(os.time())

    leftLostTime = -1  -- tiempo en que se perdi? la pared izquierda
    turnDelay = 0.5    -- segundos que esperamos antes de girar a la izquierda


    -- Handles
```

```
    robot = {}
    robot.base = sim.getObjectHandle(sim.handle_self)
    robot.leftMotor = sim.getObjectHandle("RevoluteLeft")
    robot.rightMotor = sim.getObjectHandle("RevoluteRight")
    robot.sensor = sim.getObjectHandle("proximitySensor")
    robot.camera = sim.getObjectHandle("noseCam")
    robot.leftSensor = sim.getObjectHandle("leftSensor")

    robot.graph = sim.getObject("/Act1Rob/graph")
    robot.stream = sim.addGraphStream(robot.graph, "distancia_libre", "m", 0, {1, 0, 0})

    minMaxSpeed = { -50 * math.pi / 180, 300 * math.pi / 180 }
    speed = (minMaxSpeed[1] + minMaxSpeed[2]) / 2
    backUntilTime = 0

    -- UI con control imagen agregado
    local name = sim.getObjectName(robot.base)
    local xml =
        '<ui title="' .. name .. ' speed" closeable="false" resizeable="false"
activate="false">' ..
        [[
        <hslider minimum="0" maximum="100" onchange="speedChange_callback" id="1"/>
        <label text="" style="* {margin-left: 300px;}"/>
        <image id="2" width="320" height="240"/>
        </ui>
        ]]
    ui = simUI.create(xml)
    simUI.setSliderValue(ui, 1, 100 * (speed - minMaxSpeed[1]) / (minMaxSpeed[2] -
minMaxSpeed[1]))
end

function sysCall_actuation()
    local frontDetected, frontDist = sim.readProximitySensor(robot.sensor)
    local leftDetected, leftDist = sim.readProximitySensor(robot.leftSensor)

    local now = sim.getSimulationTime()
    local frontBlocked = frontDetected > 0 and frontDist < 0.4
    local leftWall = leftDetected > 0 and leftDist < 0.3

    -- Comportamiento principal
    if frontBlocked then
        -- ?? Pared enfrente ? gira a la derecha inmediatamente
        sim.setJointTargetVelocity(robot.leftMotor, speed)
        sim.setJointTargetVelocity(robot.rightMotor, -speed)
        leftLostTime = -1  -- reseteamos el temporizador de giro
    elseif not leftWall then
        -- ? No hay pared a la izquierda
        if leftLostTime < 0 then
            -- Registramos el momento en que se perdi? la pared
            leftLostTime = now
            -- Avanzamos recto un poco m?s
            sim.setJointTargetVelocity(robot.leftMotor, speed)
```

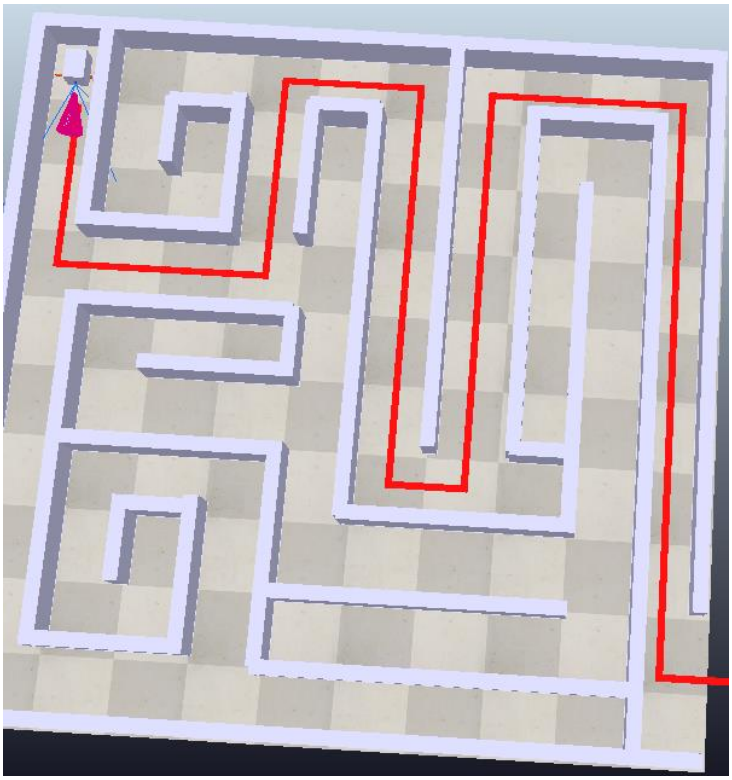
```
            sim.setJointTargetVelocity(robot.rightMotor, speed)
        elseif now - leftLostTime >= turnDelay then
            -- Ha pasado suficiente tiempo sin pared ? ahora s? giramos a la izquierda
            sim.setJointTargetVelocity(robot.leftMotor, -speed)
            sim.setJointTargetVelocity(robot.rightMotor, speed)
        else
            -- A?n dentro del tiempo de espera ? seguir recto
            sim.setJointTargetVelocity(robot.leftMotor, speed)
            sim.setJointTargetVelocity(robot.rightMotor, speed)
        end
    else
        -- ? Pared a la izquierda presente ? seguir recto
        sim.setJointTargetVelocity(robot.leftMotor, speed)
        sim.setJointTargetVelocity(robot.rightMotor, speed)
        leftLostTime = -1  -- reseteamos el temporizador
    end
end

function sysCall_sensing()
    local image, resolution = sim.getVisionSensorImg(robot.camera, 0)
    if image and type(resolution) == "table" then
        local w = math.floor(resolution[1])
        local h = math.floor(resolution[2])
        simUI.setImageData(ui, 2, image, w, h)
    end
end

function sysCall_cleanup()
    if ui and type(ui) == "number" then
        simUI.destroy(ui)
    end
end
```

Este código hará que nuestro robot siga una pared izquierda, con la capacidad de evitar obstáculos frontales. Si se detecta un obstáculo frontal, el robot gira a la derecha inmediatamente. Si el robot pierde la pared izquierda, espera un breve período antes de girar a la izquierda para re-alinearse. La velocidad del robot se puede ajustar mediante un control deslizante en la interfaz de usuario, que también muestra la vista de una cámara montada en el robot.

Cuando le damos Play, el sensor se ilumina cuando toca la pared. Esto es señal de que está detectando la pared con éxito, sin embargo no es capaz de girar correctamente cuando la pared ya no la toca. Por lo que el código falta mucho por ajustar, pero es un buen inicio.

3.2. Visión artificial y una línea trazada en el suelo

El segundo método consiste en usar nuestra cámara para que el robot siga una línea trazada en el suelo. Para ello haremos uso de los planos en coppelia, siguiendo la misma lógica que el laberinto uniremos todos los planos para formar solo un objeto. La línea será roja para que destaque del suelo.



Eliminaremos el script de la cámara y pasaremos a tener todo en un solo script. El cual es el siguiente:

```
-- Declaramos una nueva variable global
turnDirection = 1  -- 1: derecha, -1: izquierda
lineError = 0      -- desviaci?n para seguir la l?nea

-- Flag para indicar si los valores de color vienen normalizados [0..1] o en bytes [0..255]
colorsAreNormalized = false  -- cambia a true si los valores van de 0 a 1

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

function sysCall_init()
    math.randomseed(os.time())

    -- Handles
    robot = {}
    robot.base = sim.getObjectHandle(sim.handle_self)
    robot.leftMotor = sim.getObjectHandle("RevoluteLeft")
    robot.rightMotor = sim.getObjectHandle("RevoluteRight")
    robot.sensor = sim.getObjectHandle("proximitySensor")
    robot.camera = sim.getObjectHandle("noseCam")
    robot.graph = sim.getObject("/Act1Rob/graph")
    robot.stream = sim.addGraphStream(robot.graph, "distancia_libre", "m", 0, {1, 0, 0})
```

```
    minMaxSpeed = { -50 * math.pi / 180, 300 * math.pi / 180 }
    speed = (minMaxSpeed[1] + minMaxSpeed[2]) / 2
    backUntilTime = 0

    -- UI con control imagen agregado
    local name = sim.getObjectName(robot.base)
    local xml =
        '<ui title="' .. name .. ' speed" closeable="false" resizeable="false"
activate="false">' ..
        [[
        <hslider minimum="0" maximum="100" onchange="speedChange_callback" id="1"/>
        <label text="" style="* {margin-left: 300px;}"/>
        <image id="2" width="320" height="240"/>
        </ui>
        ]]
    ui = simUI.create(xml)
    simUI.setSliderValue(ui, 1, 100 * (speed - minMaxSpeed[1]) / (minMaxSpeed[2] -
minMaxSpeed[1]))
end

function sysCall_sensing()
    local image, resolution = sim.getVisionSensorImg(robot.camera, 0)
    if image and type(resolution) == "table" then
        local w = resolution[1]
        local h = resolution[2]

        -- Mostrar imagen en interfaz
        simUI.setImageData(ui, 2, image, w, h)

        -- Convertir imagen a tabla de bytes reales
        local imageData = sim.unpackUInt8Table(image)

        -- Seguimiento de l?nea
        local rowY = math.floor(h / 2)  -- fila central de la imagen
        local sumX = 0
        local count = 0

        for x = 1, w do
            local idx = 3 * ((rowY - 1) * w + (x - 1))
            local r = imageData[idx + 1]
            local g = imageData[idx + 2]
            local b = imageData[idx + 3]

            if r > 150 and g < 80 and b < 80 then
                sumX = sumX + (x - w / 2)
                count = count + 1
            end
        end

        if count > 0 then
            lineError = sumX / count
```

```
        else
            lineError = 0
        end
    end
end

function sysCall_actuation()
    local result, distance = sim.readProximitySensor(robot.sensor)
    local now = sim.getSimulationTime()

    if result > 0 and distance < 0.3 then  -- menor distancia para evitar frenado prematuro
        backUntilTime = now + 2
        turnDir = math.random(0, 1) == 0 and "left" or "right"
    end

    if now < backUntilTime then
        -- Modo evasi?n
        if turnDir == "left" then
            sim.setJointTargetVelocity(robot.leftMotor, -speed / 4)
            sim.setJointTargetVelocity(robot.rightMotor, -speed / 2)
        else
            sim.setJointTargetVelocity(robot.leftMotor, -speed / 2)
            sim.setJointTargetVelocity(robot.rightMotor, -speed / 4)
        end
    else
        -- Seguimiento de l?nea con correcci?n proporcional
        local Kp = 0.006  -- puedes ajustar este valor
        local correction = lineError * Kp
        correction = math.max(-speed / 2, math.min(speed / 2, correction))  -- limitar
correcci?n

        local leftSpeed = speed - correction
        local rightSpeed = speed + correction

        sim.setJointTargetVelocity(robot.leftMotor, leftSpeed)
        sim.setJointTargetVelocity(robot.rightMotor, rightSpeed)
    end

    if result > 0 then
        sim.setGraphStreamValue(robot.graph, robot.stream, distance)
    end
end

function sysCall_cleanup()
    if ui and type(ui) == "number" then
        simUI.destroy(ui)
    end
end
```

Este script hará que nuestro robot siga una línea, evitando obstáculos. El robot utiliza una cámara para detectar la posición de la línea y ajustar las velocidades de sus motores para mantenerse centrado sobre ella, con una corrección proporcional basada en el error de posición. Además, el robot emplea un sensor de proximidad para detectar obstáculos; cuando se detecta uno, el robot retrocede y gira en una dirección aleatoria antes de intentar seguir la línea nuevamente.

Con este resultado, el robot no lo hace del todo mal pero no puede funcionar sin la ayuda humana, ya que le ocurren problemas como engancharse en la pared al girar demasiado rápido al ver la línea girar a un lado. Podemos decir que es el típico robot que depende del factor humano para poder continuar su labor.

4. Conclusiones

En ambos casos todavía queda mucho por hacer para que el robot pueda salir sin problemas. Los resultados obtenidos son los siguientes:

Método	Solución	Problema	Mejoras
Método para seguir una pared	Usar un sensor lateral	No reacciona a tiempo cuando deja de detectar la pared.	Aumentar más la reacción del sensor.
Método para seguir una línea trazada en el suelo	Usar la cámara y la visión artificial	Gira demasiado rápido cuando la línea gira y se engancha.	Aumentar más el tiempo de espera para que gire en el momento oportuno.