

TAREA 5

Robótica con Python - Coppelia Sim



**COPPELIA
ROBOTICS**

ÍNDICE

Archivos del proyecto.....	3
Repositorio.....	3
1. Robótica con Python-Primeros pasos en Coppelia Sim-Diseño de un robot simple..	4
Paso 1: Descargar e instalar el software de coppelia robotics.....	4
Paso 2: Crear el primer cilindro - base del robot:.....	4
Paso 3: Anclar la primera articulación - Joint rotacional.....	5
Paso 4: Primer eslabón - Cuboid.....	6
Paso 5: Añadir más componentes.....	7
Paso 6: Cambiar nombres de los elementos añadidos.....	7
Paso 7: Estructura cinemática del robot - juntar piezas para que sean dependientes....	8
Paso 8: Cambiar propiedades dinámicas.....	8
Paso 9: Cambiar propiedades dinámicas de los joint 1 y 2.....	11
2. Robótica con Python - Primeros pasos en Coppelia Sim - Uso de Remote API.....	12
Paso 10: Descargar el cuaderno 4 del github y mover archivos python.....	12
Paso 11: Abrir puerto para conectar con python.....	13
Paso 12: Obtener los handlers.....	14
Paso 13: Posición de las articulaciones.....	14
Paso 14: Mover el robot con código.....	15
Paso 15: Fallos y revisiones.....	16

Archivos del proyecto

Nombre del archivo principal: project_ivan (subido con este pdf para probar funcionamiento)

```
└── Video/
    └── resultado_final.mp4
├── robot-ivan.ttt
├── sim.py
├── simConst.py
└── __pycache__
    ├── sim.cpython-313.pyc
    └── simConst.cpython-313.pyc
└── remoteApi.dll
```

Repositorio

Aquí están todos los archivos:

Github: https://github.com/IvanFalconMonzon/TA5_CoppeliaSim_IvanFalconMonzon.git

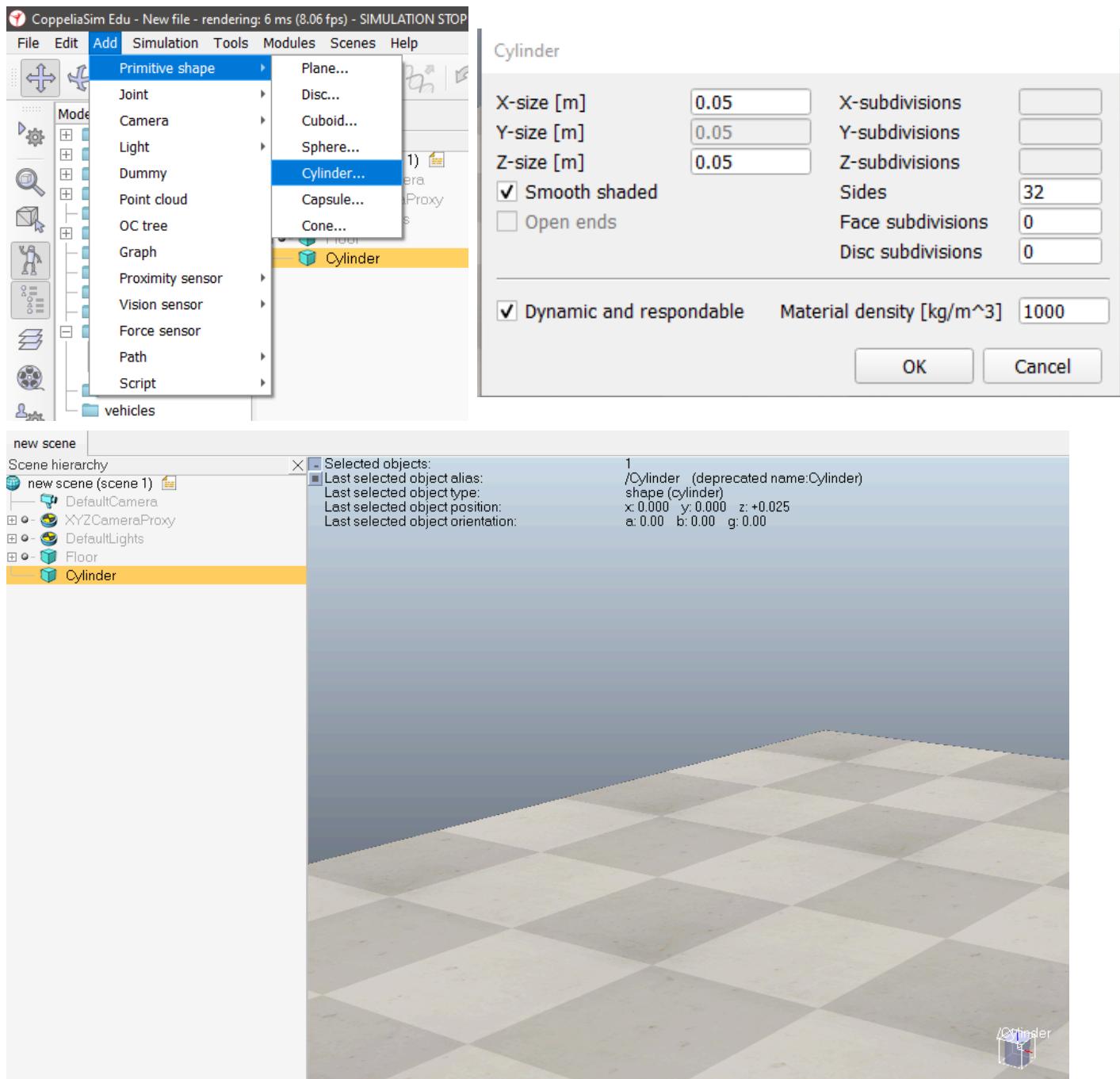
1. Robótica con Python-Primeros pasos en Coppelia Sim-Diseño de un robot simple

→ Enlace vídeo: https://youtu.be/m4_cXLoreXg?si=PUmqBRadDLNud-AX

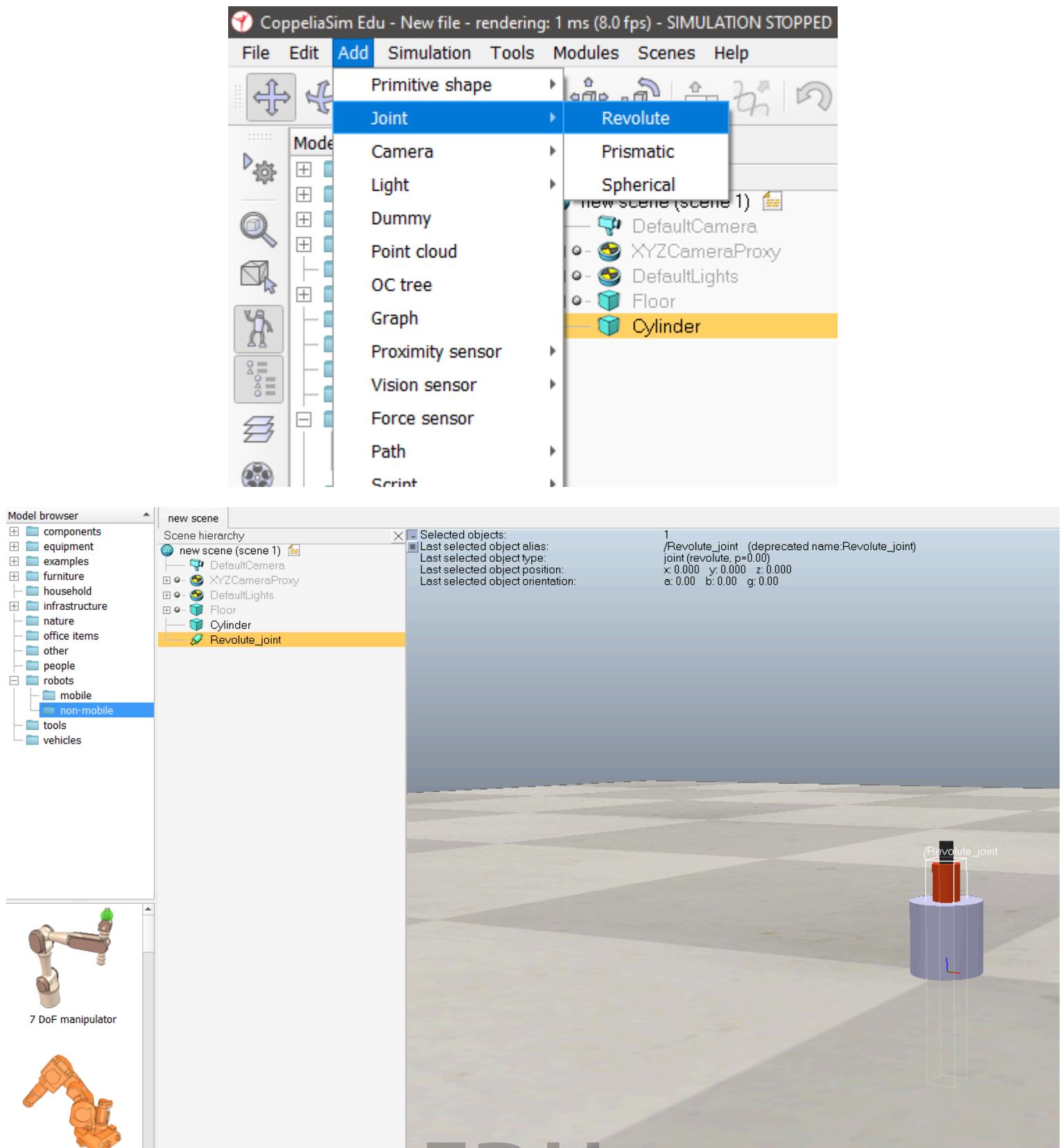
Paso 1: Descargar e instalar el software de coppelia robotics

→ Enlace descarga: <https://www.coppeliarobotics.com/>

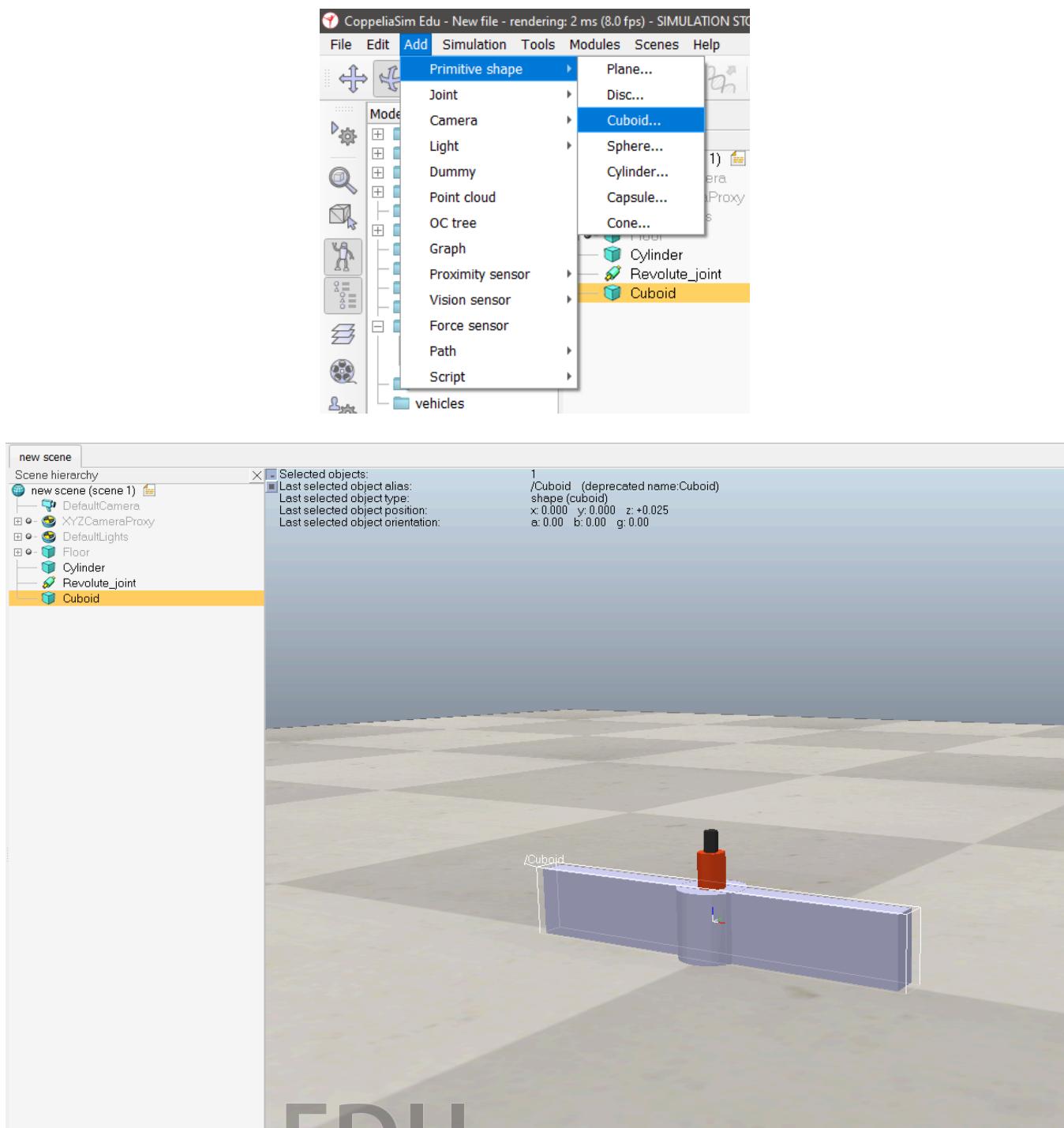
Paso 2: Crear el primer cilindro - base del robot:



Paso 3: Anclar la primera articulación - Joint rotacional



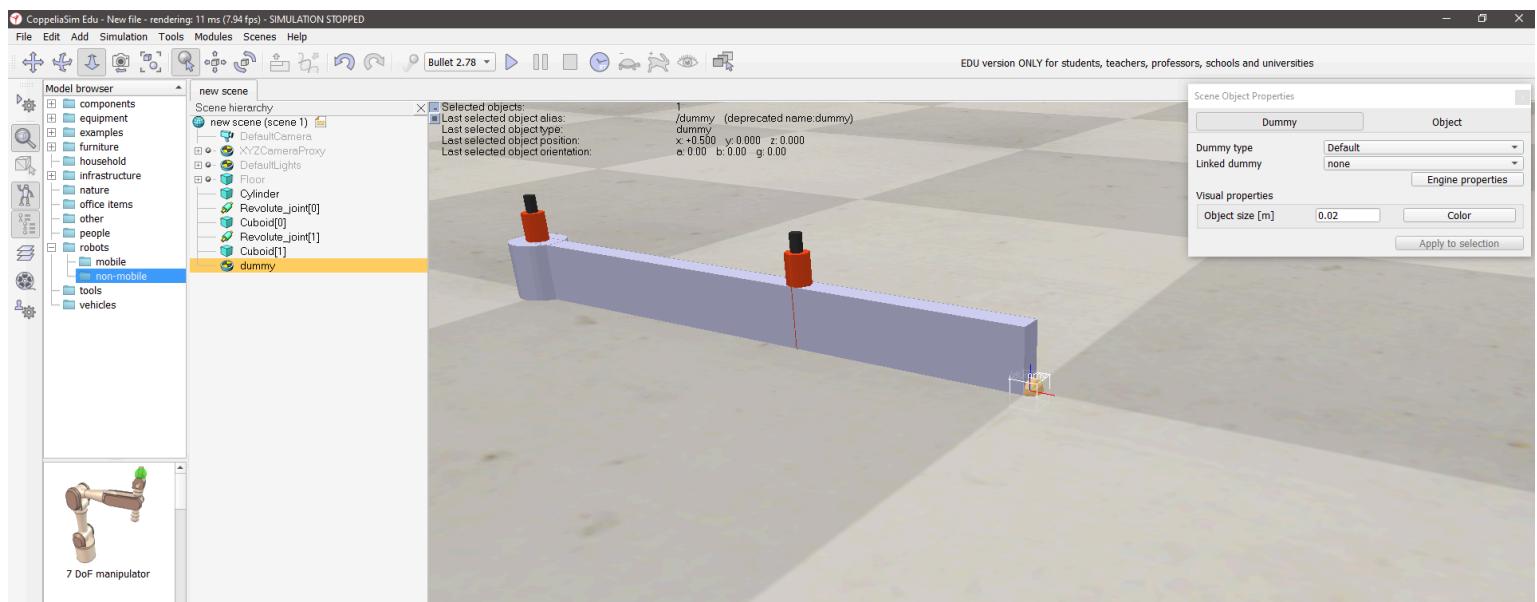
Paso 4: Primer eslabón - Cuboid



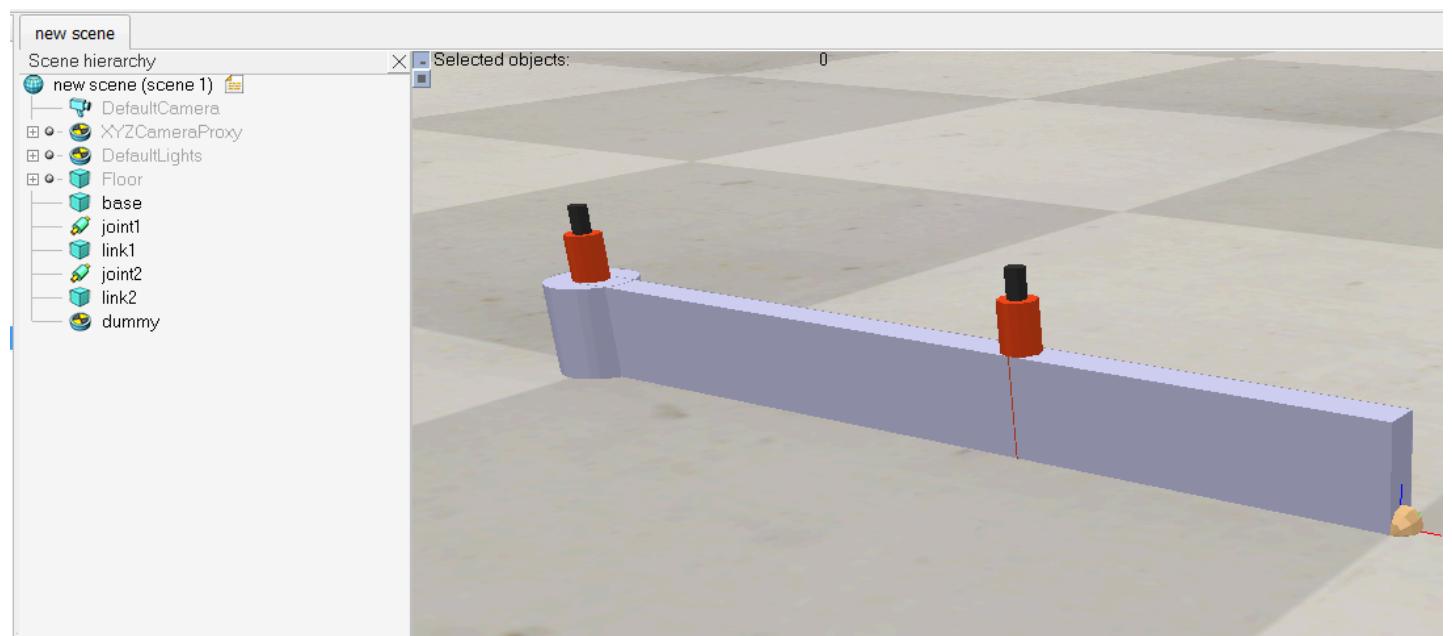
Paso 5: Añadir más componentes

Para que la tarea no se haga muy larga hago varios pasos y pongo una captura de pantalla de estos cambios realizados.

1. Mover la barra al centro
2. Añadir otra articulación rotacional
3. Añadir otro cuboid y moverlo
4. Añadir dummy y agrandarlo

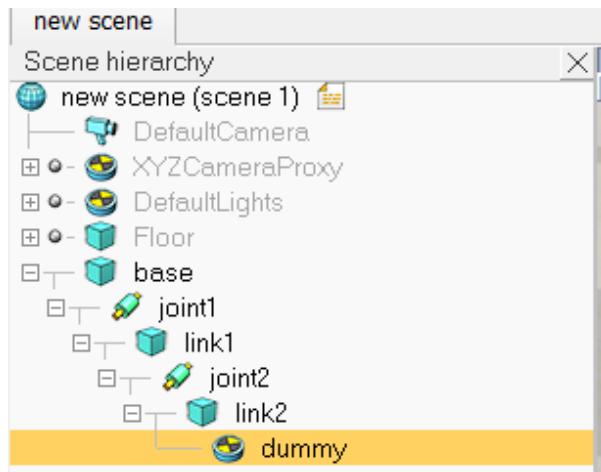


Paso 6: Cambiar nombres de los elementos añadidos



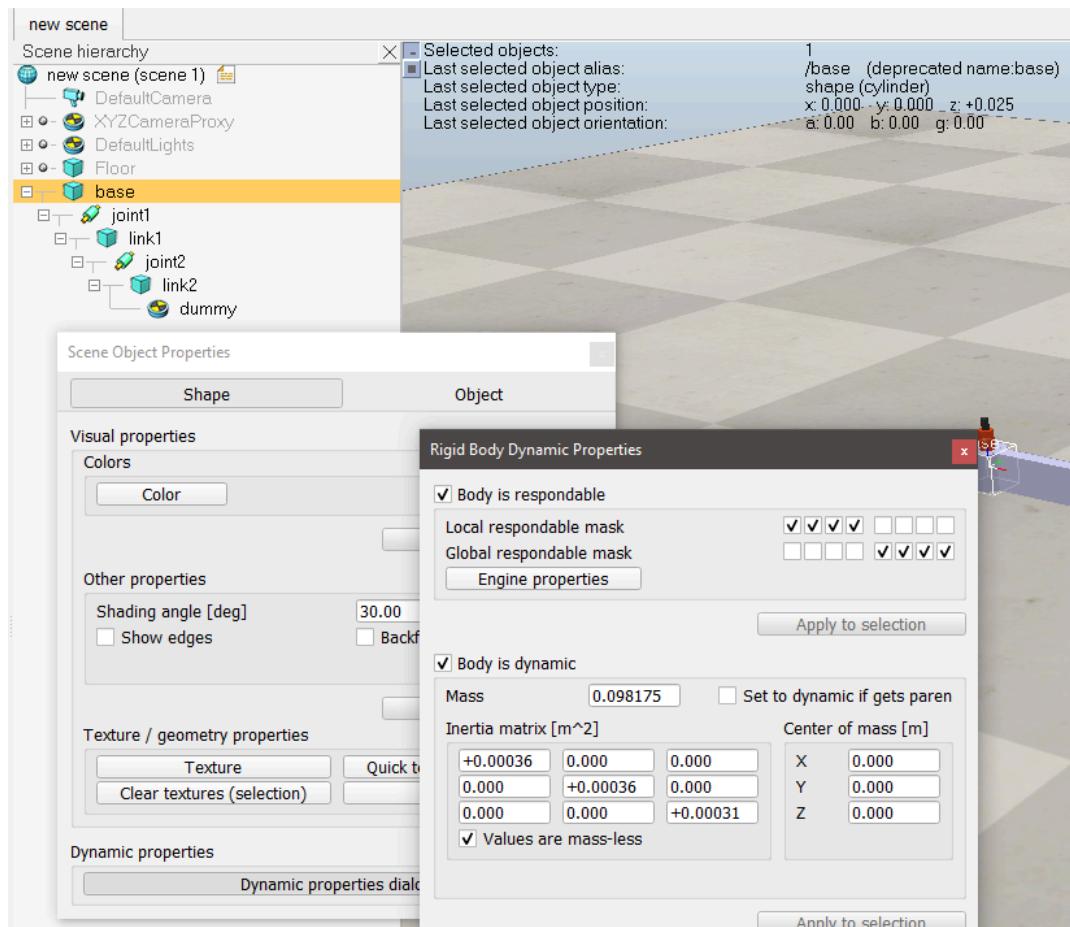
Paso 7: Estructura cinemática del robot - juntar piezas para que sean dependientes

Cada pieza depende jerárquicamente de la anterior.

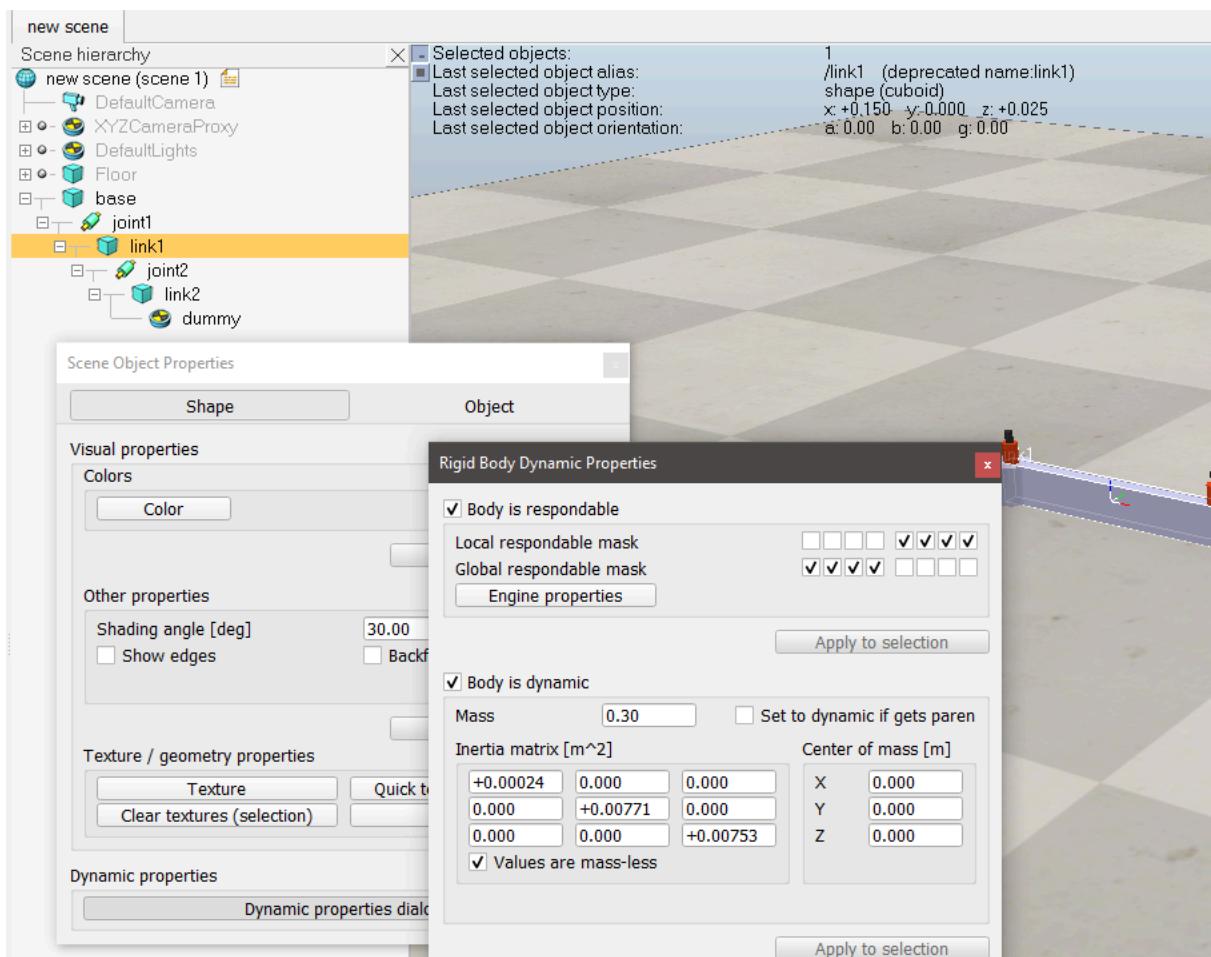


Paso 8: Cambiar propiedades dinámicas

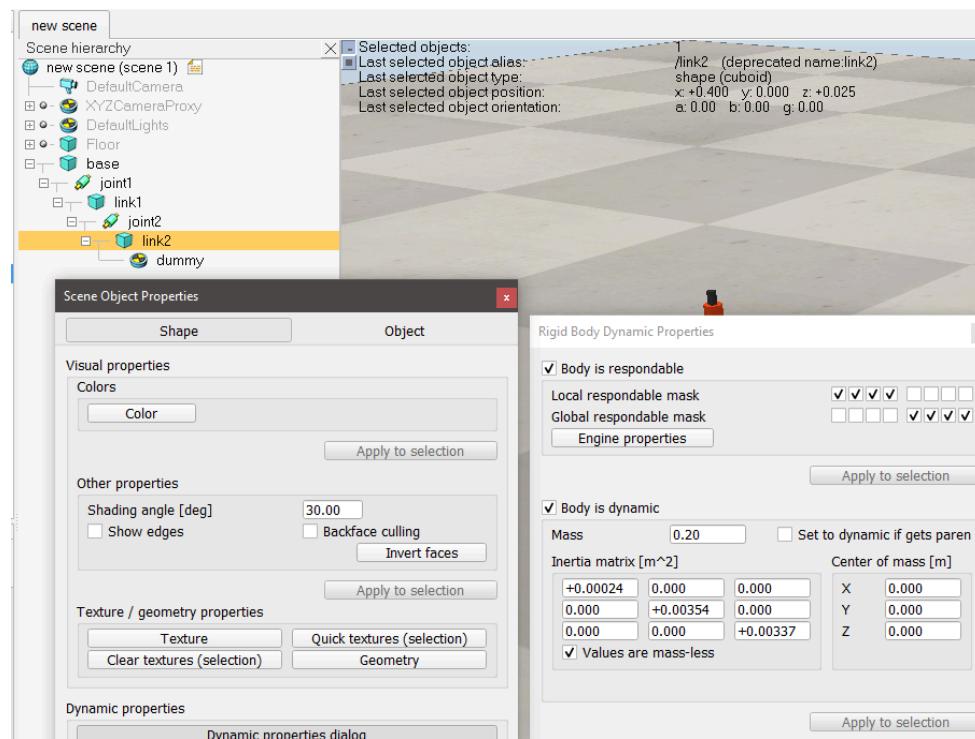
Base:



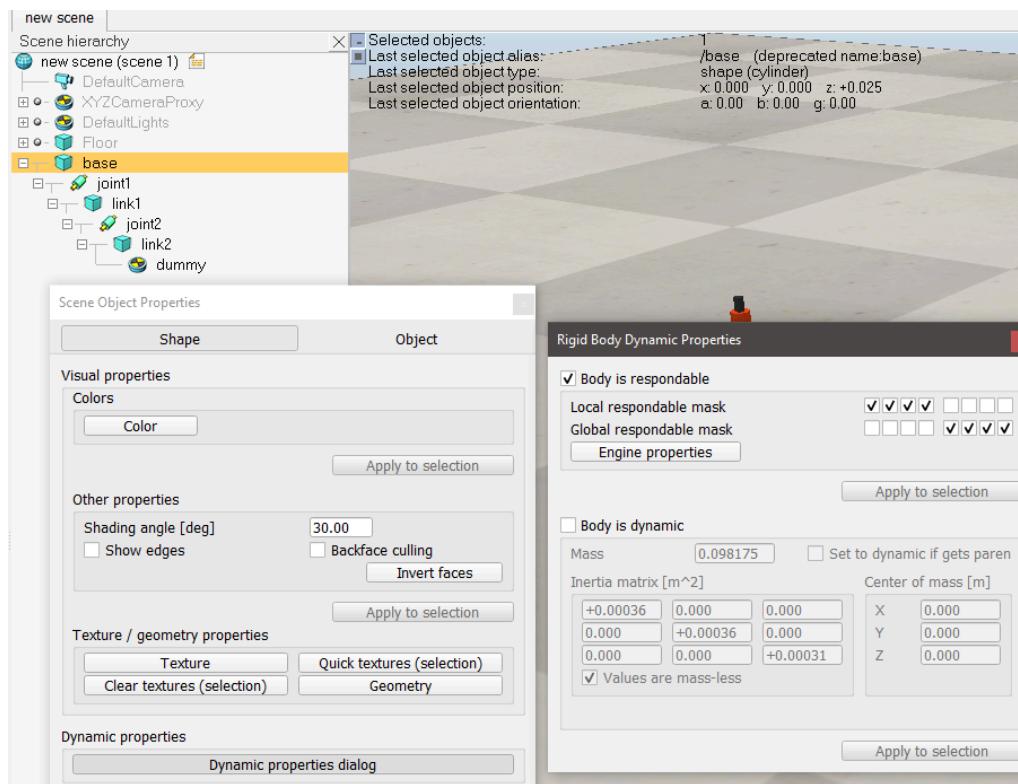
link1: las propiedades opuestas a la base:



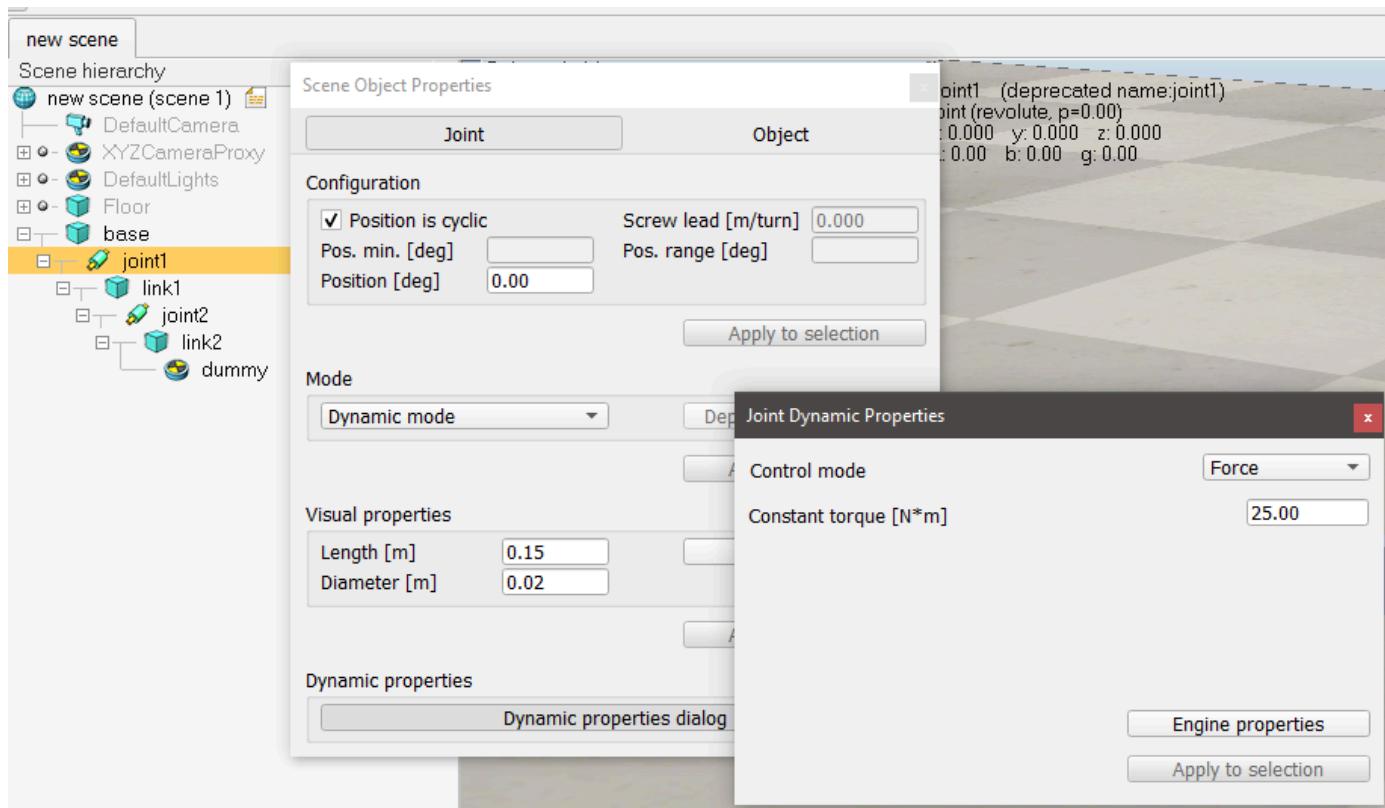
link2 como la base:



También quitar de la base el Body is dynamic para que quede anclado o sólido al piso.



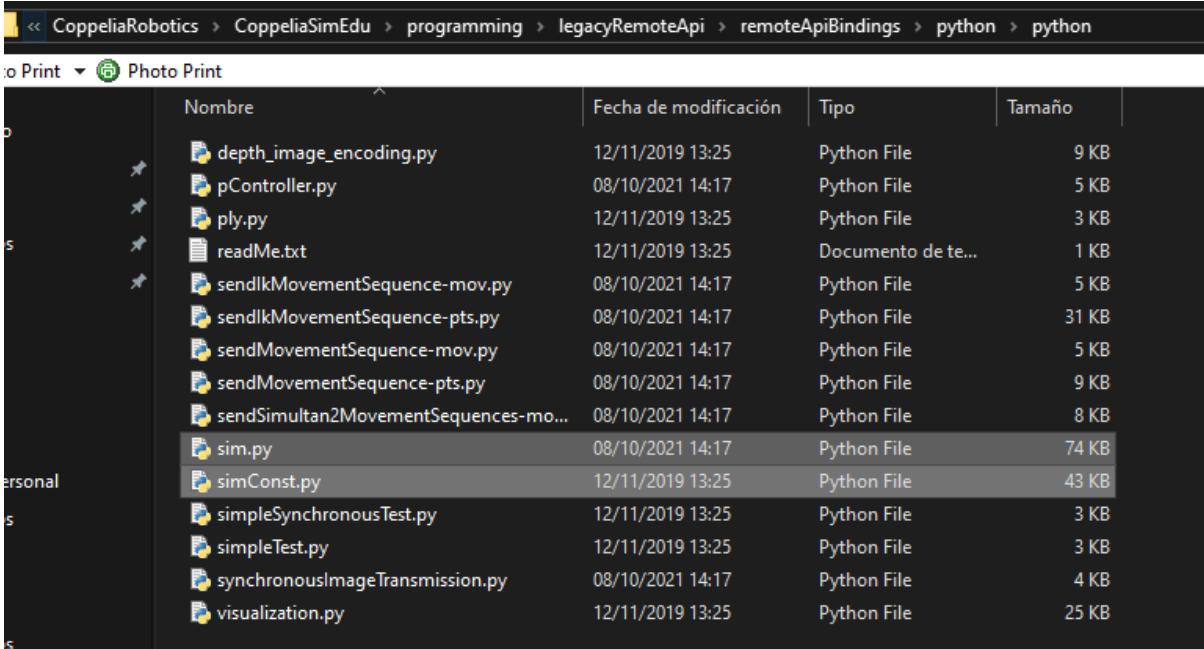
Paso 9: Cambiar propiedades dinámicas de los joint 1 y 2.



2. Robótica con Python - Primeros pasos en CoppeliaSim - Uso de Remote API.

- Enlace vídeo: https://youtu.be/bzyuHDmhq84?si=90VqV4PTVdIeY_6w
- Cuaderno de Jupyter: <https://github.com/cmoralesd/robopy>

Paso 10: Descargar el cuaderno 4 del github y mover archivos python

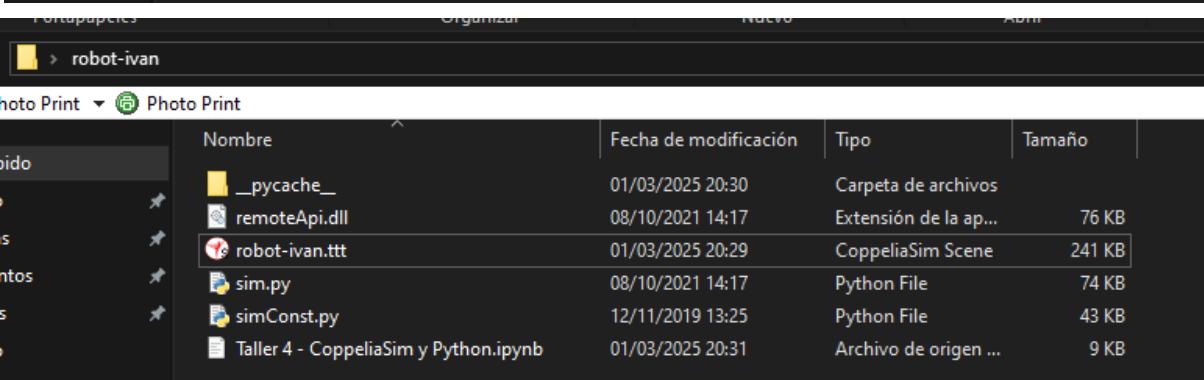


The screenshot shows a file explorer window with the following directory path:

```
« CoppeliaRobotics > CoppeliaSimEdu > programming > legacyRemoteApi > remoteApiBindings > python > python
```

The table lists several Python files:

Nombre	Fecha de modificación	Tipo	Tamaño
depth_image_encoding.py	12/11/2019 13:25	Python File	9 KB
pController.py	08/10/2021 14:17	Python File	5 KB
ply.py	12/11/2019 13:25	Python File	3 KB
readMe.txt	12/11/2019 13:25	Documento de te...	1 KB
sendIkMovementSequence-mov.py	08/10/2021 14:17	Python File	5 KB
sendIkMovementSequence-pts.py	08/10/2021 14:17	Python File	31 KB
sendMovementSequence-mov.py	08/10/2021 14:17	Python File	5 KB
sendMovementSequence-pts.py	08/10/2021 14:17	Python File	9 KB
sendSimultaneousMovementSequences-mo...	08/10/2021 14:17	Python File	8 KB
sim.py	08/10/2021 14:17	Python File	74 KB
simConst.py	12/11/2019 13:25	Python File	43 KB
simpleSynchronousTest.py	12/11/2019 13:25	Python File	3 KB
simpleTest.py	12/11/2019 13:25	Python File	3 KB
synchronousImageTransmission.py	08/10/2021 14:17	Python File	4 KB
visualization.py	12/11/2019 13:25	Python File	25 KB



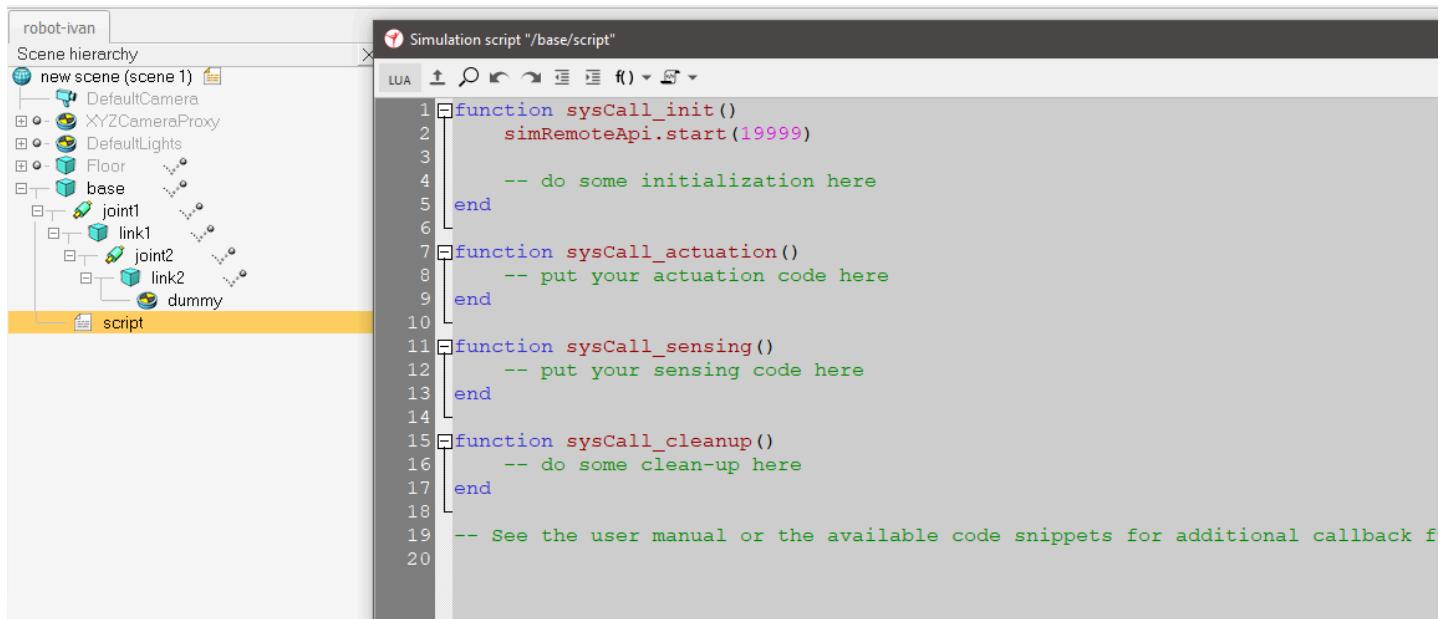
The screenshot shows a file explorer window with the following directory path:

```
« Portapapeles > robot-ivan
```

The table lists the contents of the 'robot-ivan' folder:

Nombre	Fecha de modificación	Tipo	Tamaño
__pycache__	01/03/2025 20:30	Carpeta de archivos	
remoteApi.dll	08/10/2021 14:17	Extensión de la ap...	76 KB
robot-ivan.ttt	01/03/2025 20:29	CoppeliaSim Scene	241 KB
sim.py	08/10/2021 14:17	Python File	74 KB
simConst.py	12/11/2019 13:25	Python File	43 KB
Taller 4 - CoppeliaSim y Python.ipynb	01/03/2025 20:31	Archivo de origen ...	9 KB

Paso 11: Abrir puerto para conectar con python



The screenshot shows a Jupyter Notebook interface. The 'EXPLORER' sidebar lists files: '_pycache_', 'remoteApi.dll', 'robot-ivan.ttt', 'sim.py', 'simConst.py', and 'Taller 4 - CoppeliaSim y Python.ipynb'. The notebook cell [6] contains the following Python code:

```
import sim
import numpy as np
```

The cell output shows a checkmark and '0.0s'. The next cell [8] contains the following Python code:

```
def connect(port):
    # Establece la conexión a VREP
    # port debe coincidir con el puerto de conexión en VREP
    # retorna el número de cliente o -1 si no puede establecer conexión
    sim.simxFinish(-1) # just in case, close all opened connections
    clientID=sim.simxStart('127.0.0.1',port,True,True,2000,5) # Conectarse
    if clientID == 0: print("conectado a", port)
    else: print("no se pudo conectar")
    return clientID
```

The cell output shows a checkmark and '0.0s'. The next cell [9] contains the following Python code:

```
# Conectarse al servidor de VREP
# *** ejecutar cada vez que se reinicia la simulación ***
clientID = connect(19999)
```

The cell output shows a checkmark and '0.0s'. The final output shows the message '... conectado a 19999'.

Paso 12: Obtener los handlers

2. Obtener los manejadores (handlers)

Un manejador (handler) es un número identificador que asigna VREP para cada uno de los elementos de la es-

```
[19] # Obtenemos el manejador para el dummy
      returnCode,handle=sim.simxGetObjectHandle(clientID,'dummy',sim.simx_opmode_blocking)
      dummy = handle
      print(dummy)
```

[19] ✓ 0.0s

... 21

```
[20] # A partir de su manejador podemos accionar sobre el objeto,
      # por ejemplo, obtener su posición
      returnCode,pos=sim.simxGetObjectPosition(clientID, dummy, -1, sim.simx_opmode_blocking)
      print(pos)
```

[20] ✓ 0.0s

... [0.4999988079071045, 0.00011141360300825909, 7.17289054819048e-08]

▷ ▾

```
[21] # Obtenemos los manejadores para cada una de las articulaciones del robot
      ret,joint1=sim.simxGetObjectHandle(clientID,'joint1',sim.simx_opmode_blocking)
      ret,joint2=sim.simxGetObjectHandle(clientID,'joint2',sim.simx_opmode_blocking)
      print(joint1, joint2)
```

[21] ✓ 0.0s

... 17 19

Paso 13: Posición de las articulaciones

3. ¿Cuál es la posición de las articulaciones?

Utilizando los manejadores, podemos obtener información de los elementos.

```
[22] # leemos la posición de joint1, en radianes.
      returnCode, pos1 = sim.simxGetJointPosition(clientID, joint1, sim.simx_opmode_blocking)
      print(pos1)
```

[22] ✓ 0.0s

... -0.00042923836736008525

▷ ▾

```
[23] returnCode, pos2 = sim.simxGetJointPosition(clientID, joint2, sim.simx_opmode_blocking)
      print(pos2)
```

[23] ✓ 0.0s

... 0.0016175099881365895

Paso 14: Mover el robot con código

4. ... y movemos el robot

Utilizando los manejadores, podemos enviar parámetros a los elementos.

```
[135] # enviamos la posición de joint1, en radianes.  
q1 = 0 * np.pi/180  
returnCode = sim.simxSetJointTargetPosition(clientID, joint1, q1, sim.simx_opmode_oneshot)  
print(returnCode)
```

[135]

... 0

```
[132] q2 = 40 * np.pi/180  
returnCode = sim.simxSetJointTargetPosition(clientID, joint2, q2, sim.simx_opmode_oneshot)  
print(returnCode)
```

[132]

... 0

```
[122] # Ponemos todo el código junto  
# conectamos  
clientID = connect(19999)  
# obtenemos los manejadores  
returnCode,handle=sim.simxGetObjectHandle(clientID,'dummy',sim.simx_opmode_blocking)  
dummy = handle  
ret,joint1=sim.simxGetObjectHandle(clientID,'joint1',sim.simx_opmode_blocking)  
ret,joint2=sim.simxGetObjectHandle(clientID,'joint2',sim.simx_opmode_blocking)  
#movemos  
q1 = -90 * np.pi/180  
returnCode = sim.simxSetJointTargetPosition(clientID, joint1, q1, sim.simx_opmode_oneshot)  
q2 = 0 * np.pi/180  
returnCode = sim.simxSetJointTargetPosition(clientID, joint2, q2, sim.simx_opmode_oneshot)
```

[122]

Paso 15: Fallos y revisiones

Como me daba fallos al moverlas las articulaciones he probado con estos códigos nuevos:

Código para saber si esta todo correcto

```
returnCode, handle = sim.simxGetObjectHandle(clientID, 'dummy', sim.simx_opmode_blocking)
if returnCode != sim.simx_return_ok:
    print("Error al obtener el handle de 'dummy'")
    exit()

ret, joint1 = sim.simxGetObjectHandle(clientID, 'joint1', sim.simx_opmode_blocking)
if ret != sim.simx_return_ok:
    print("Error al obtener el handle de 'joint1'")
    exit()

ret, joint2 = sim.simxGetObjectHandle(clientID, 'joint2', sim.simx_opmode_blocking)
if ret != sim.simx_return_ok:
    print("Error al obtener el handle de 'joint2'")
    exit()
```

También se ha añadido un código completo con para que se ejecute de manera automática:

Código completo con cambios para que se mueva y vuelva al principio

```
import time
import numpy as np

# Verificar conexión con CoppeliaSim
if clientID != -1:
    print("Conexión establecida correctamente.")
else:
    print("Error en la conexión con CoppeliaSim.")
    exit()

# Enviar los comandos de movimiento con opmode_blocking
q1 = 0 * np.pi / 180 # Ángulo para joint1 (cambiar a un valor simple)
q2 = 0 * np.pi / 180 # Ángulo para joint2

# Mover joint1
returnCode = sim.simxSetJointTargetPosition(clientID, joint1, q1, sim.simx_opmode_blocking)
if returnCode != sim.simx_return_ok:
    print("Error al mover joint1")
else:
    print("Movimiento de joint1 solicitado")

# Mover joint2
returnCode = sim.simxSetJointTargetPosition(clientID, joint2, q2, sim.simx_opmode_blocking)
if returnCode != sim.simx_return_ok:
    print("Error al mover joint2")
else:
    print("Movimiento de joint2 solicitado")

# Esperar un segundo para que los movimientos se apliquen
time.sleep(1)

# Verificar las posiciones actuales de las articulaciones
ret, jointPos1 = sim.simxGetJointPosition(clientID, joint1, sim.simx_opmode_blocking)
ret, jointPos2 = sim.simxGetJointPosition(clientID, joint2, sim.simx_opmode_blocking)

# Mostrar las posiciones de las articulaciones
print(f"Posición actual de joint1: {jointPos1}")
print(f"Posición actual de joint2: {jointPos2}")
```

28] Conexión establecida correctamente.

Movimiento de joint1 solicitado

Movimiento de joint2 solicitado

Posición actual de joint1: -7.974923832421155e-09

Posición actual de joint2: 0.02335146628320217