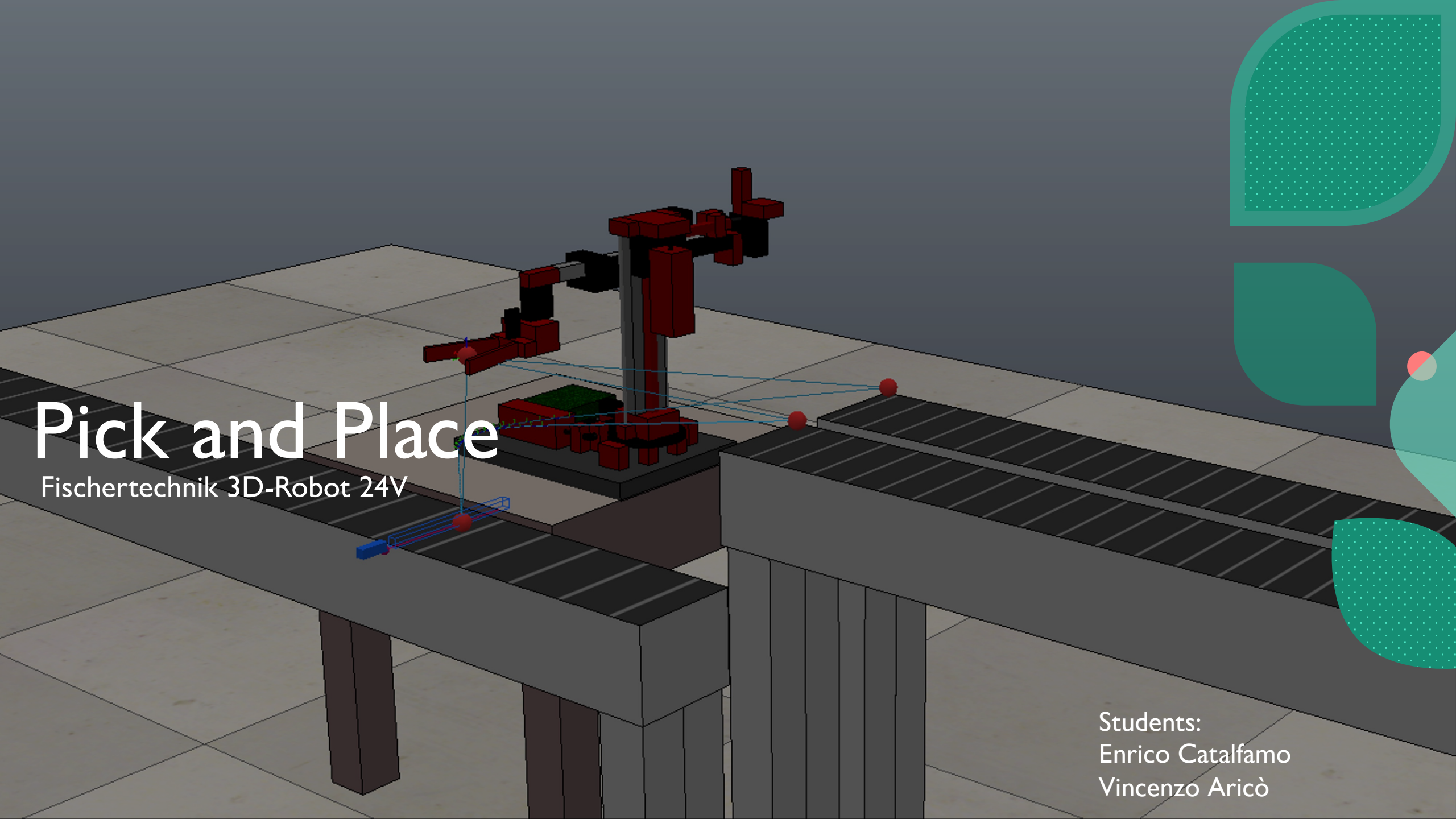


Pick and Place

Fischertechnik 3D-Robot 24V

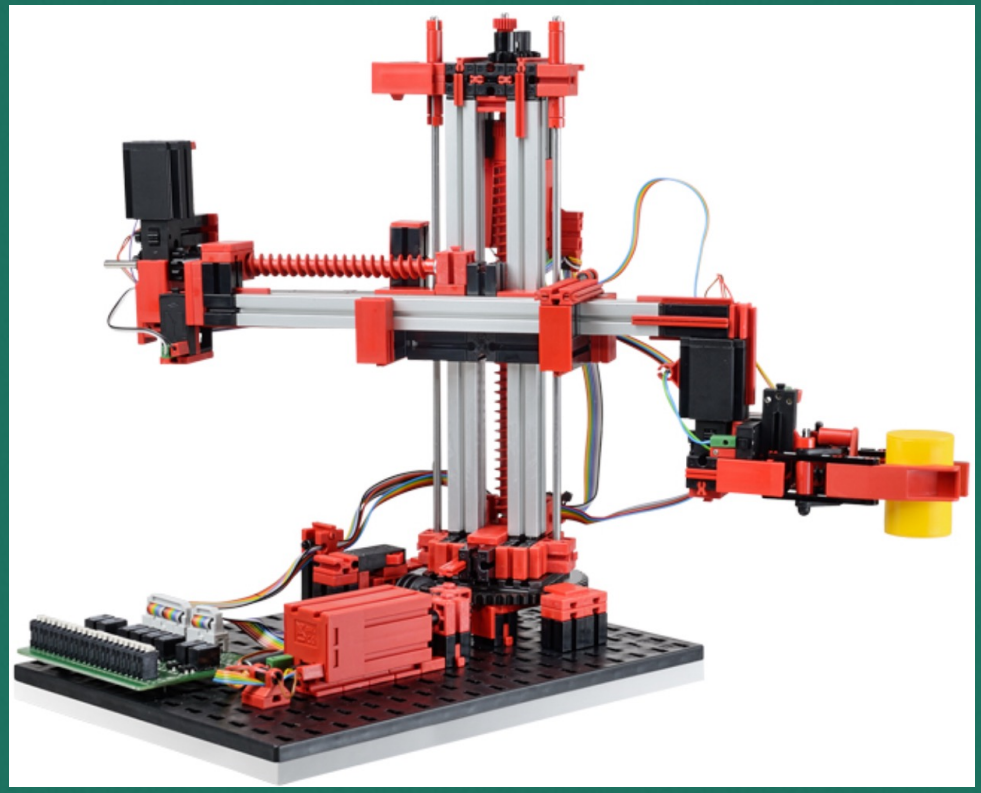
Students:
Enrico Catalfamo
Vincenzo Aricò

The idea and the technologies involved

- **CoppeliaSim Edu** version 4.1.0
- **Robot model**: Fischertechnik 3D-Robot 24V (https://github.com/giorgionocera/fischertechnik-3D-robot)
- **Goal**: reproducing real-case scenario of a **pick and place** system with **proximity sensor** and **object recognition**
- **Main objects** in the scene:
  - 3 conveyor belt
  - 1 proximity sensor (ultrasonic)
  - 1 vision sensor
  - a table with the robot on top

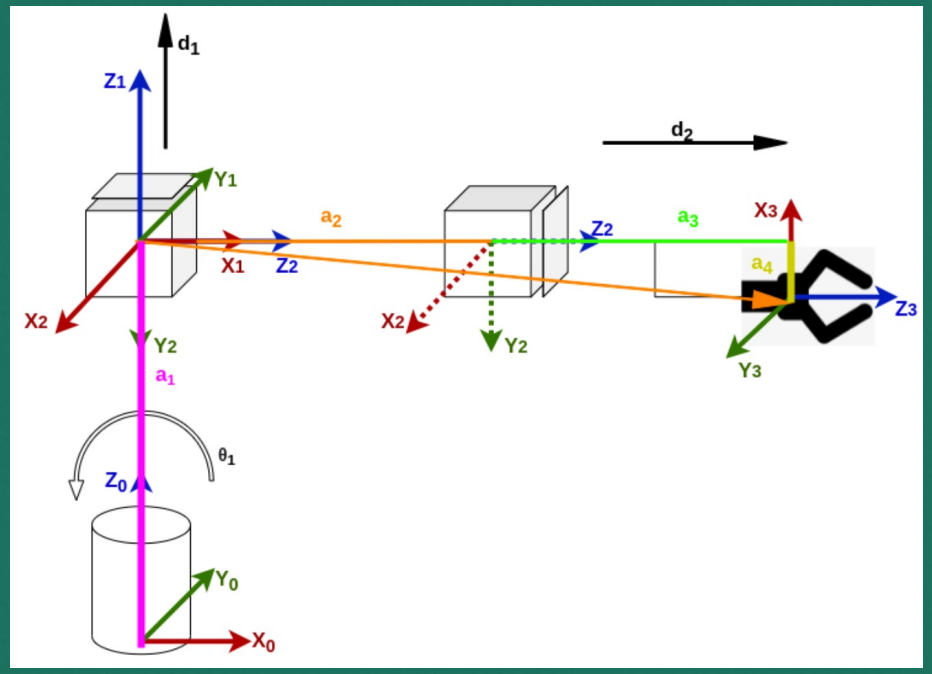
Fischertechnik 3D-Robot 24V

- The 3D-Robot 24V is a robotic arm produced by Fischertechnik. It is a **3-DOF** (3 degrees of freedom) robot and in particular it is composed by:
  - a revolute joint
  - 2 prismatic joints (up/down and frontward/backward)
  - a gripper as end-effector
- **Degrees of freedom**:
  - axis 1: turn 180°
  - axis 2: forward/back 90 mm
  - axis 3: raise/lower 150 mm
- The robot is characterized by a cylindrical geometry: its **workspace** is a **portion of a hollow cylinder**



Direct kinematic Fischertechnik 3D-Robot 24V

- Kinematic analysis of the mechanical structure of a robot concerns the description of the motion with respect to a fixed reference Cartesian frame by **ignoring the forces and moments** that cause motion of the structure. Independently by the weight, the kinematic motion is the same
- **Direct kinematic** describes the **end-effector position and orientation (pose) as a function of the joint variables** of the mechanical structure with respect to a reference frame



Frames according the four DH rules

θ	α	**r** (cm)	**d** (cm)
θ_1	0	0	18.5
$-90°$	$-90°$	0	d_1
$-90°$	0	6.75	$20 + d_2$

Denavit-Hartenberg parameter table

Homogeneous Transformation Matrix Fischertechnik 3D-Robot 24V

$$T_n^0(q) = A_1^0(q1)A_2^1(q2)...A_n^{n-1}(qn)$$

$$A_1^0(q_1) = \begin{bmatrix} c_{\theta_1} & -s_{\theta_1} & 0 & 0 \\ s_{\theta_1} & c_{\theta_1} & 0 & 0 \\ 0 & 0 & 1 & 18.5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_2^1(q_2) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_3^2(q_3) = \begin{bmatrix} r0 & 1 & 0 & 0 \\ -1 & 0 & 0 & -6.75 \\ 0 & -1 & 0 & 20 + d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_3^0(q_3) = \begin{bmatrix} 0 & s_{\theta_1} - c_{\theta_1} & 0 & c_{\theta_1} d_1 \\ 0 & -c_{\theta_1} - s_{\theta_1} & 0 & d_1 s_{\theta_1} \\ 1 & 0 & 0 & d_1 + 13.5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

System functioning description

- A conveyor belt is used to convey the cubes (blue and green) to the pick-up position
- Once the cube reaches the pick-up position, it is detected by the proximity sensor and the conveyor belt stops
- The type of cube is recognized via the vision sensor
- The robot picks the cubes and moves them on a specific conveyor belt based on their color
- During sorting, the main conveyor belt continues to bring items in the pick-up position

Setting the robot in inverse kinematic

- All the robot joints have to be set in **inverse kinematics mode**
- Two dummies have to be added to the scene: **tip** and **target**
- The tip must have the gripper as its parent, and is positioned in its center
- The target is the point in space that the tip must reach. When moving the target in the scene, the robot will try to reach its position
- It's necessary to realize the **IK link** between the two dummies to make sure the tip follows the target
- Finally, you just need to create an **IK group** associated to the tip through CoppeliaSim Calculation Modules

Cuboids generation

- The script is **threaded**, and it is associated to the “Cube_generator” dummy
- When a cube is generated, it is assigned a random color (blue or green)
- In order for the cube to be detected by the vision and proximity sensors and picked by the gripper, the '**visible**' and '**respondable**' properties of the cube must be set
- The generation position is given by the coordinates of the “Cube_generator” dummy, placed on one end of the main conveyor belt
- The generation period is 7 seconds, unless the conveyor belt is stationary

```
function getColor()
    local colors = {{0, 0, 255}, {0, 255, 0}}
    rn = math.random(2)
    return colors[rn]
end

local Proximity_sensorHandle = sim.getObjectHandle('Proximity_sensor')
local Cube_generatorHandle = sim.getObjectHandle('Cube_generator')
local cube = 0
local visibleEdges = 2
local respondableShape = 8
local size = {0.03, 0.03, 0.03}
local mass = 0.1
local specialProperty = sim.boolOr32(sim.objectspecialproperty_renderable, sim.objectspecialproperty_detectable_all)

while true do
    if sim.readProximitySensor(Proximity_sensorHandle)==0 then
        local shape = sim.createPureShape(cube, visibleEdges+respondableShape, size, mass, NULL)
        sim.setObjectPosition(shape, -1, {sim.getObjectPosition(Cube_generatorHandle, -1)[1],
                                          sim.getObjectPosition(Cube_generatorHandle, -1)[2],
                                          sim.getObjectPosition(Cube_generatorHandle, -1)[3]})

        sim.setObjectSpecialProperty(shape, specialProperty)
        sim.setShapeColor(shape, nil, sim.colorcomponent_ambient_diffuse, getColor())
        sim.wait(7)
    end
end
```

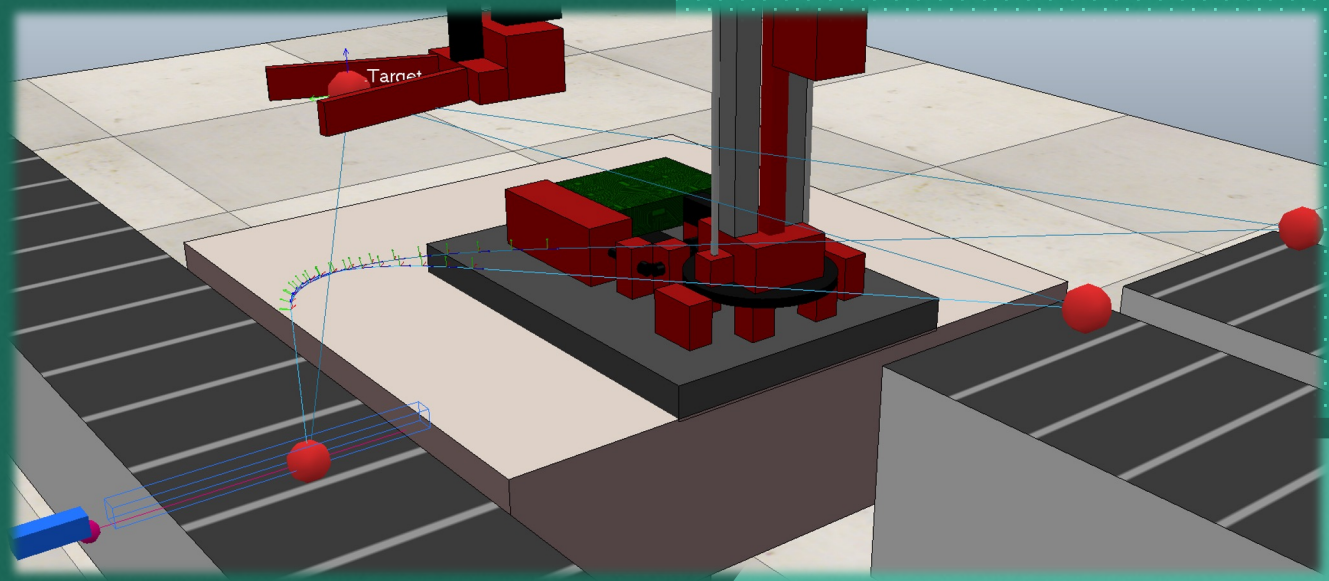
Robot motion code

- The sorting path changes based on the color of the cube
- Inside the while loop the *getCubeColor()* function is called which modifies the **cubeColor** variable, setting it based on the color of the cube detected by the vision sensor
- The cubeColor variable will be -1 if there is no cube at the pick-up position
- The *followPath()* function moves the target along the selected path. Since the tip follows the target, the robot will follow the path

```
function sysCall_threadmain()
    -- Put some initialization code here


    -- Put your main loop here, e.g.:
    --
    -- while sim.getSimulationState()~=sim.simulation_advancing_abouttos
    --     local p=sim.getObjectPosition(objHandle,-1)
    --     p[1]=p[1]+0.001
    --     sim.setObjectPosition(objHandle,-1,p)
    --     sim.switchThread() -- resume in next simulation step
    -- end
    targetHandle = sim.getObjectHandle('Target')
    End_green_pathHandle = sim.getObjectHandle('End_green')
    End_blue_pathHandle = sim.getObjectHandle('End_blue')
    Start_pathHandle = sim.getObjectHandle('Start')
    Start_green_pathHandle = sim.getObjectHandle('Start_green')
    Start_blue_pathHandle = sim.getObjectHandle('Start_blue')

    Vision_sensorHandle = sim.getObjectHandle('Vision_sensor')
    Proximity_sensorHandle = sim.getObjectHandle('Proximity_sensor')
    gripperHandle = sim.getObjectHandle('Force_sensor')

    cubeColor = -1 -- blue = 0, green = 1

    while true do
        getCubeColor()

        if cubeColor==0 then
            sim.followPath(targetHandle, Start_pathHandle, 1, 0, 0.1, 0)
            obj = grabCube() -- chiudi gripper
            sim.followPath(targetHandle, Start_blue_pathHandle, 1, 0, 0.
            dropCube(obj) -- apri gripper
            sim.followPath(targetHandle, End_blue_pathHandle, 1, 0, 0.1,
        elseif cubeColor==1 then
            sim.followPath(targetHandle, Start_pathHandle, 1, 0, 0.1, 0)
            obj = grabCube() -- chiudi gripper
            sim.followPath(targetHandle, Start_green_pathHandle, 1, 0, 0
            dropCube(obj) -- apri gripper
            sim.followPath(targetHandle, End_green_pathHandle, 1, 0, 0.1
        end
    end
end
```

Paths creation

- A **Path object** is created through which it is possible to define the points in space that the robot must pass through
- Specifically, there are **two main Paths**: one for the pick and place of the blue cubes and one for the pick and place of the green cubes
- The two paths include returning to the robot's rest position
- The trajectory planning based on the path points is carried out by CoppeliaSim using the interpolation parameters



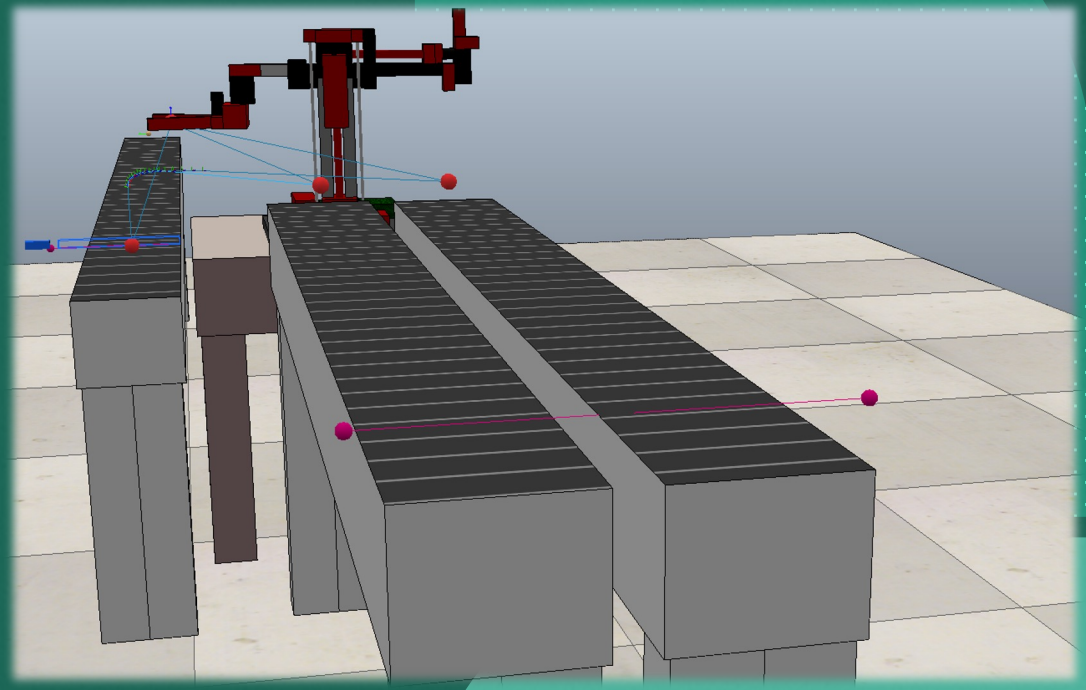
Grabbing and dropping cubes

- Since the robot model does not have a working gripper, to pick the cubes we use a **force sensor** placed in the same position as the tip (that becomes its parent)
- Exploiting the **parenting relationship**, it is possible to 'hook' the cube to the force sensor and therefore to the gripper, without it being necessary for the gripper to close
- Similarly it is possible to drop the cube by removing the parenting relationship with the force sensor

```
68 function grabCube()
69     _, _, _, detectedObjectHandle, _ = sim.readProximitySensor(Proximity_sensorHandle)
70     -- print(sim.getObjectName(detectedObjectHandle))
71     sim.setObjectParent(detectedObjectHandle, gripperHandle, true)
72     return detectedObjectHandle
73 end
74
75
76 function dropCube(obj)
77     sim.setObjectParent(obj, -1, true)
78 end
```

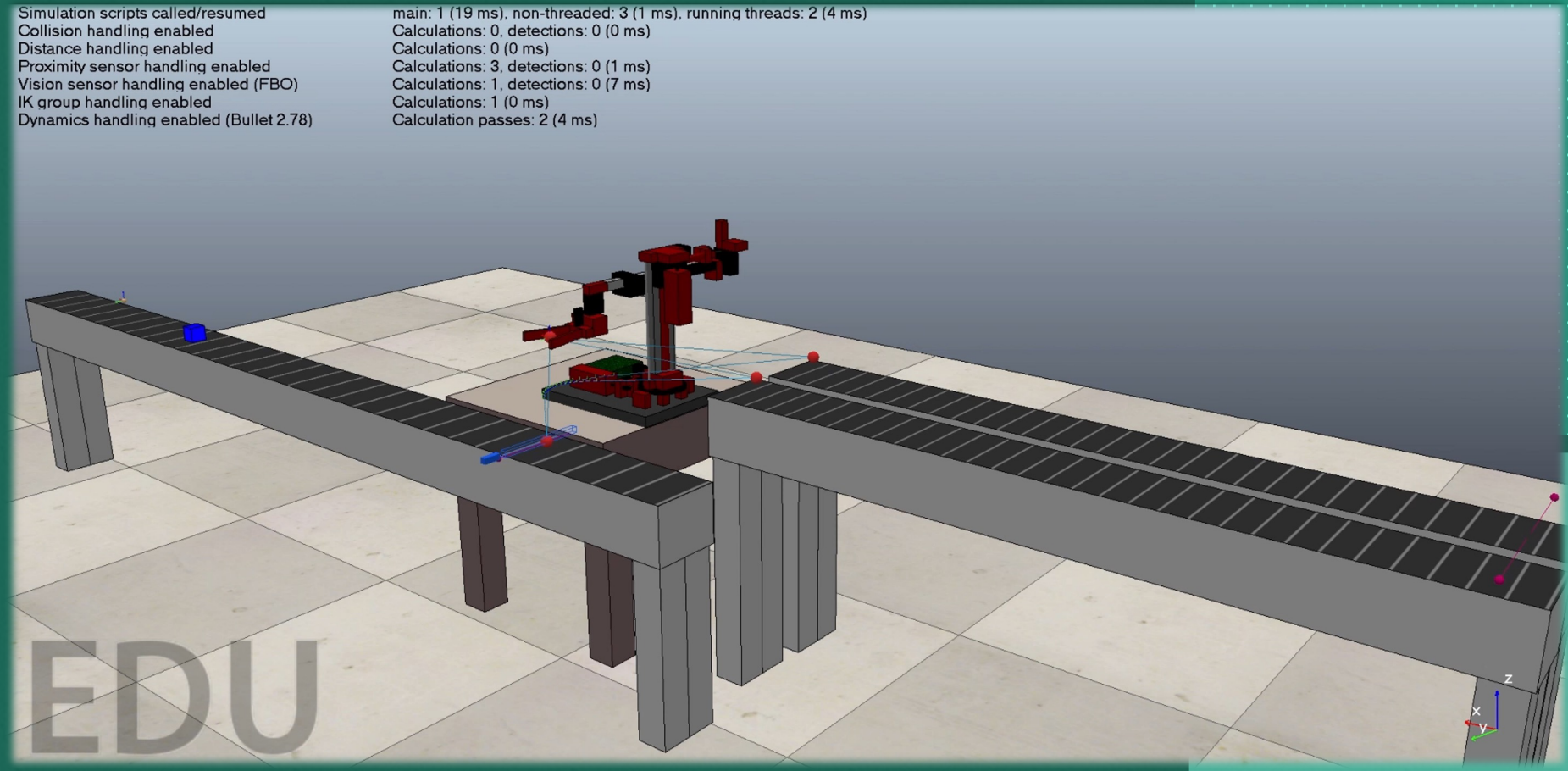
Cuboids deletion

- Once the cubes are placed in the arrival conveyor belts, it is necessary to introduce a **cube elimination mechanism** to prevent them from accumulating unnecessarily in the scene, causing a waste of memory
- For this reason, **proximity sensors** have been inserted at the end of the conveyor belts
- When a proximity sensor detects a cube, its handle is taken to delete it from the scene.



```
result, distance, detectedPoint, detectedObjectHandle, detectedSurfaceNormalVector = sim.readProximitySensor(Remove_blueCubeHandle)
if result > 0 then
    sim.removeObject(detectedObjectHandle)
end
```

Simulation video



Graphs

- Two graphs are plotted as a function of simulation time during the simulation:
  - **detection state of the proximity sensor (red)**, which will be at 1 when a cube is detected at the pick-up position, otherwise the state will be at 0.
  - **value of green and blue detected by the vision sensor compared to the maximum value (255)**, so when the value of green is maximum, the cube will be predicted as green, vice versa when the value of blue is maximum.