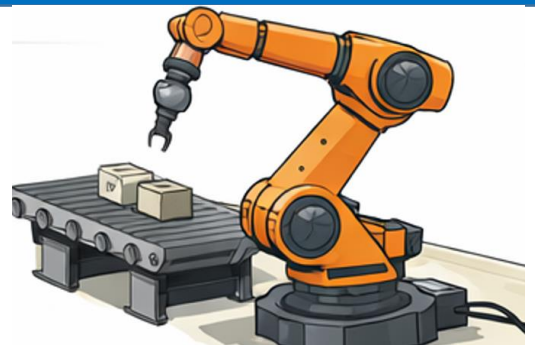
Robotique et vision par ordinateur



Master en ingénierie des systèmes embarqués (MUS ISE)

Chapitre 2: Two Wheeled Robot

Année universitaire 2025/2026

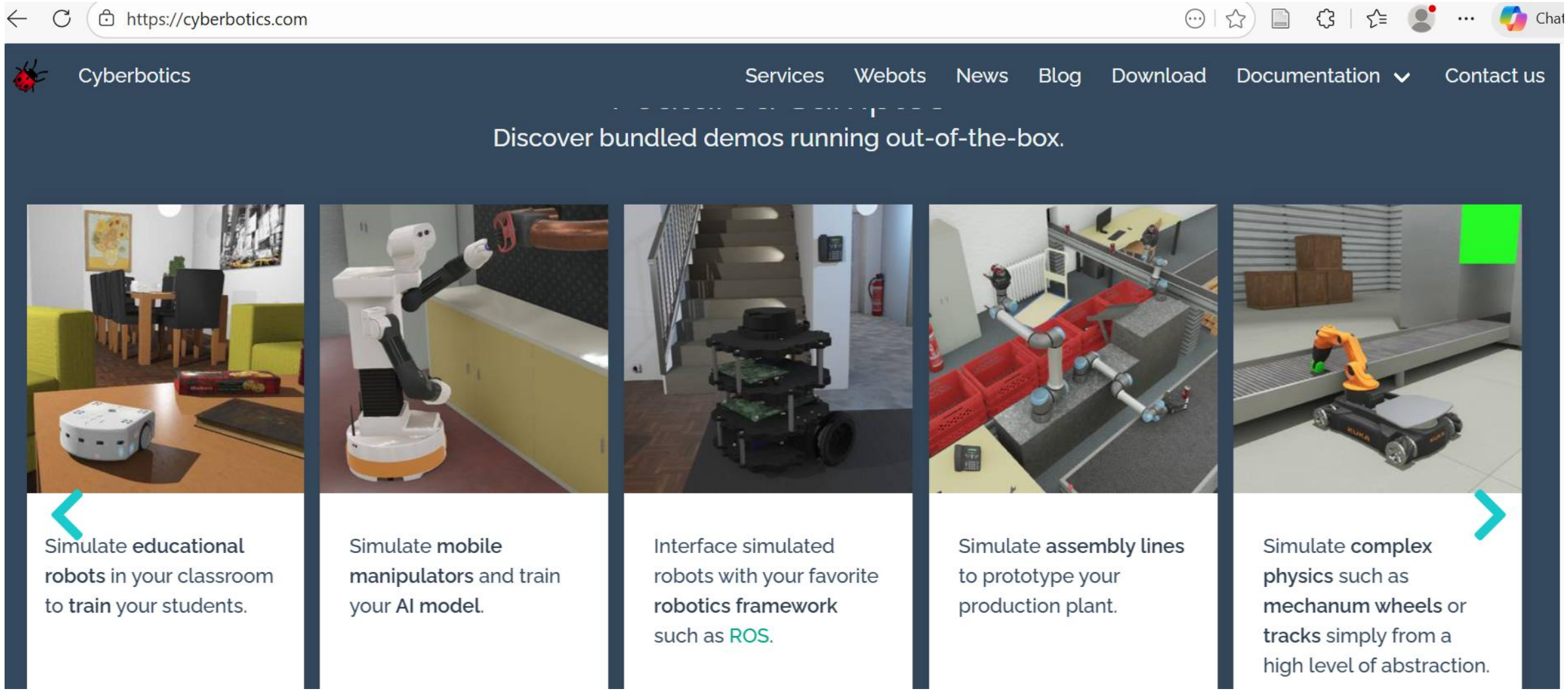
Dr. Ing. LASSIOUI Abdellah

Chapters Objectives

The main objectives of this chapters are:

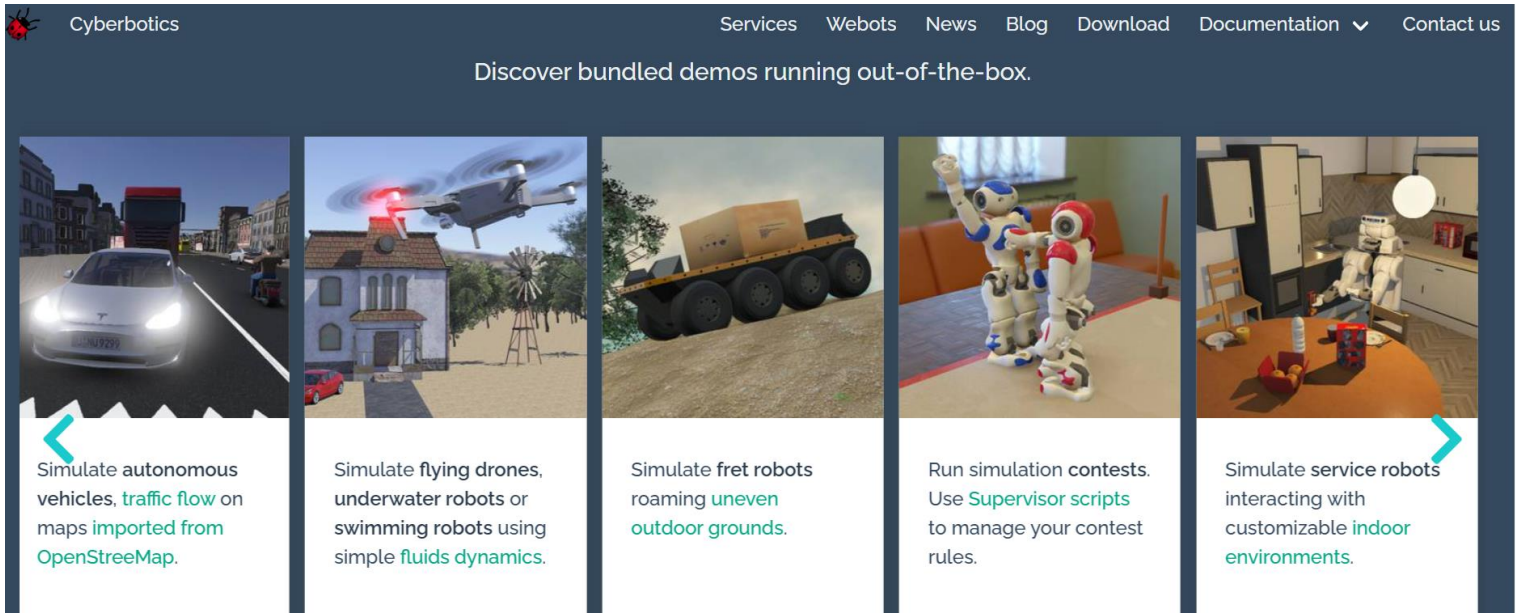
- ❑ Understand the structure of a simple mobile robot
- ❑ Analyze its mechanical and kinematic model
- ❑ Control its motion using Webots and Python
- ❑ Link mathematical models to simulated behavior

Software to install: Webot simulator
Available for free at : www.cyberbotics.com



Why Use Webots?

- Physics-based robot simulator
- No hardware required
- Fast testing and visualization
- Supports Python, C, C++
- Widely used in education and research



The Simplest Robot: Differential-Drive Robot

Structure:

- ❑ Two driven wheels
- ❑ One passive caster wheel
- ❑ Rigid chassis
- ❑ **Degrees of freedom:**
- ❑ Position: x,y
- ❑ Orientation: θ

The e-puck is a small (7 cm) differential wheeled mobile robot. It was originally designed for micro-engineering education by Michael Bonani and Francesco Mondada at the ASL laboratory of Prof.

The e-puck is open hardware and its onboard software is open-source, and is built and sold by several companies.

Technical details

- Diameter: 70 mm
- Height: 50 mm
- Weight: 200 g
- Max speed: 13 cm/s
- Autonomy: 2 hours moving
- dsPIC 30 CPU @ 30 MHz (15 MIPS)
- 8 KB RAM
- 144 KB Flash
- 2 step motors
- 8 infrared proximity and light (TCRT1000)
- color camera, 640x480
- 8 LEDs in ring + one body LED + one front LED
- 3D accelerometers
- 3 microphones
- 1 loudspeaker

Differential-Drive Kinematics

Let:

- ωL: left wheel angular speed
- ωR: right wheel angular speed

Linear velocity

$$v = \frac{R}{2}(\omega_R + \omega_L)$$

Angular velocity

$$\omega = \frac{R}{L}(\omega_R - \omega_L)$$

Motion cases

Wheel speeds	Robot motion
wL= wR	Straight line
wL= - wR	Rotation in place
wL # wR	Circular motion

Controller Concept

Controller = program that:

- ❑ Reads sensors
- ❑ Computes commands
- ❑ Drives actuators

In Webots:

- ❑ Controller runs in discrete time
- ❑ Motors are velocity-controlled

Programming Flow in Webots

- ❑ Create robot in Webots
- ❑ Assign a controller
- ❑ Read motor devices
- ❑ Set wheel velocities
- ❑ Run simulation

Controller_1 – straight forward move

The objective of this first program is to move the robot from an initial position to a final position
The objective is to show how to perform a simple moving task in order to get familiar with robot simulation

Initial position

Final position

Webots Control Code (Python) (part 1)

```
from controller import Robot

# Create the Robot instance
robot = Robot()

# Get simulation time step
timestep = int(robot.getBasicTimeStep())

# Get motors
left_motor = robot.getDevice("left wheel motor")
right_motor = robot.getDevice("right wheel motor")

# Set motors to velocity control mode
left_motor.setPosition(float('inf'))
right_motor.setPosition(float('inf'))
```

- Imports the **Robot class** from the Webots controller API.
- This class provides access to:
  - Simulation timing
  - Actuators (motors)
  - Sensors
  - Communication with the simulated robot
- Without this import, the controller cannot interact with the Webots simulation.

- Instantiates the Robot object.
- This object represents the **current robot instance** in the Webots world.
- All interactions with motors and sensors are performed through this object.

- Retrieves the **basic simulation time step** defined in the Webots world file (in milliseconds).
- The controller loop must run **synchronously** with this timestep.
- Converting to `int` is required because `robot.step()` expects an integer.

- Retrieves the motor devices by their **exact names** as defined in the Webots robot model.
- Each motor is an instance of the `Motor` class.
- These objects allow control of:
  - Position
  - Velocity
  - Torque (depending on configuration)

- By default, Webots motors operate in **position control mode**.
- Setting the position to `∞` disables position control and enables **velocity control**.
- This is mandatory when implementing:
  - Differential drive
  - Continuous motion
  - Speed-based control laws

Webots Control Code (Python) (part 2)

```
# Initial speeds
left_speed = 0.0
right_speed = 0.0

# Maximum motor speed (depends on robot)
MAX_SPEED = 6.28  # rad/s

# Main control loop
while robot.step(timestep) != -1:

    # -------- Motion selection --------

    # 1. Move forward
    left_speed = 0.5 * MAX_SPEED
    right_speed = 0.5 * MAX_SPEED

 # Send commands to motors
    left_motor.setVelocity(left_speed)
    right_motor.setVelocity(right_speed)
```

- Defines variables that will store wheel angular velocities.
- Values are expressed in **radians per second (rad/s)**.
- Initialization prevents undefined behavior at startup.

- Represents the **maximum allowable angular velocity** of the motor.
- `6.28 rad/s ≈ 2π rad/s ≈ 1 revolution per second`
- This value is robot-dependent and usually specified in the robot's datasheet or model.

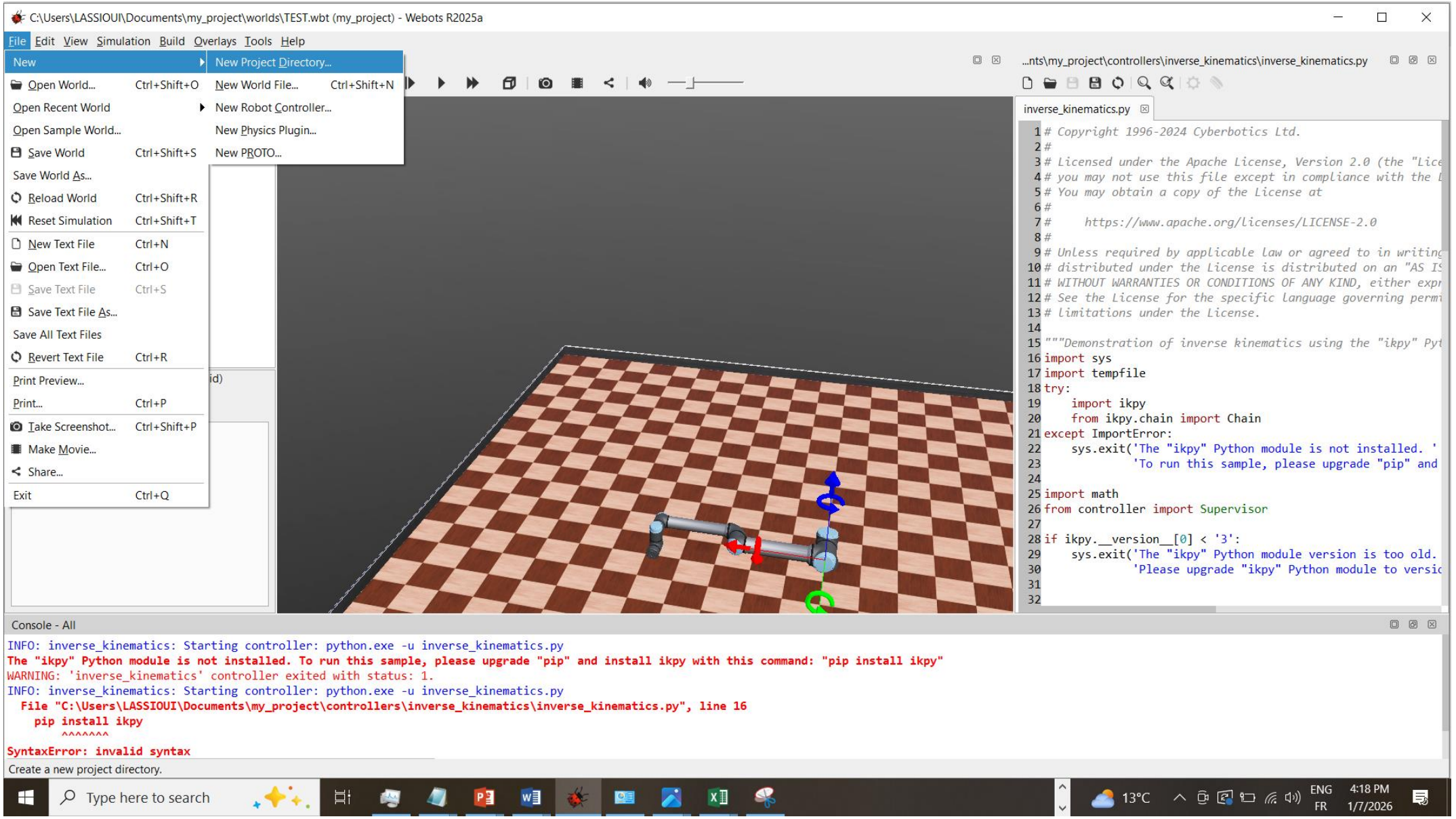
- This is the **real-time control loop**.
- `robot.step(timestep)` :
  - Advances the simulation by one timestep
  - Returns `-1` when the simulation ends
- All sensing, decision-making, and actuation must occur inside this loop.

This is equivalent to a **discrete-time control system**.

- Sets both wheel speeds to **50% of the maximum speed**.
- Since both wheels rotate at the same speed:
  - The robot moves **straight forward**
- This corresponds to a **basic open-loop velocity command**.

- Applies the computed velocities to the motors.
- The motor controller inside Webots:
  - Converts angular velocity to torque
  - Updates wheel rotation in the physics engine
- These commands are executed at every timestep.

Aller vers file → New → New Project Directory



Add an e-puck robot

*In the **Scene Tree***

*Click **Add → Robot***

Navigate to:

robots → gctronic → e-puck

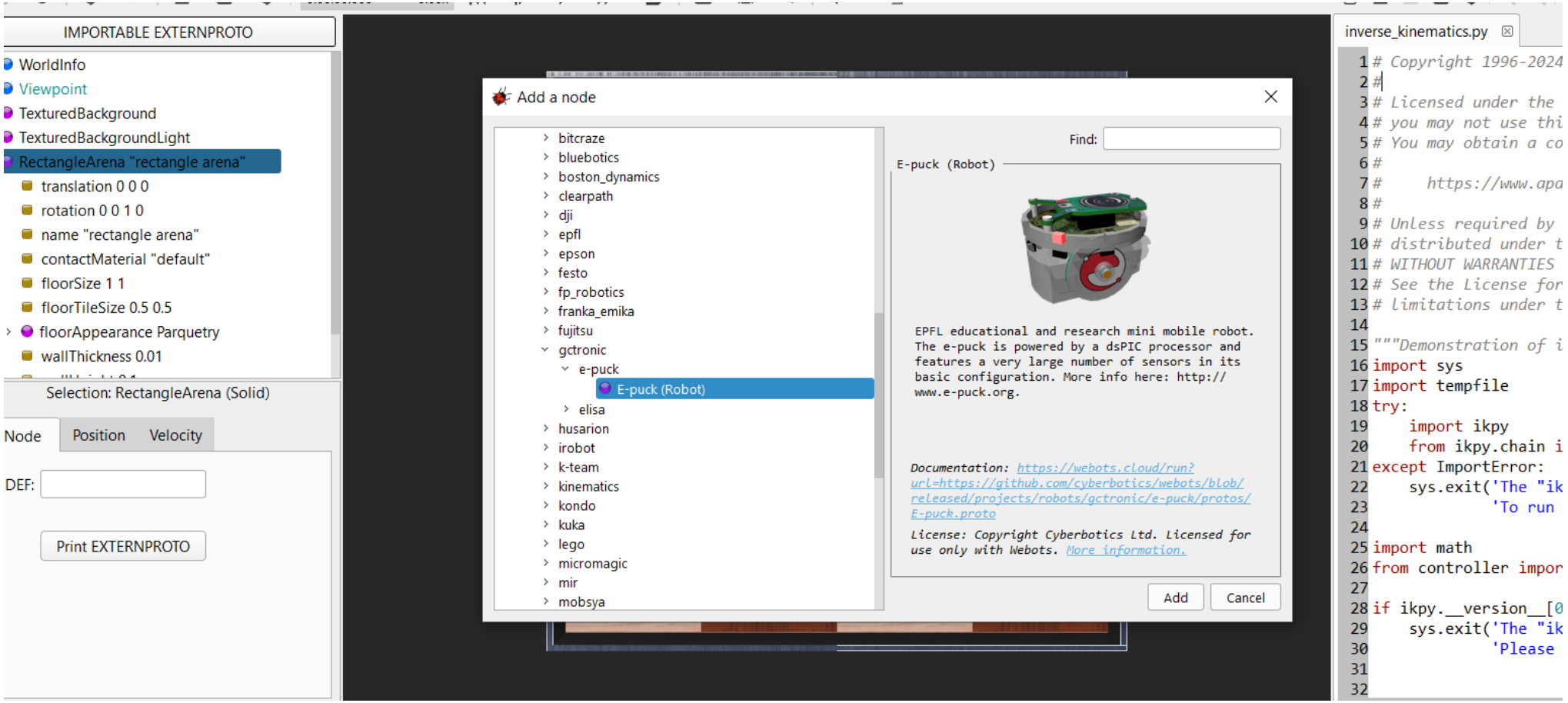
*Insert **E-puck***

You now have:

Differential-drive robot

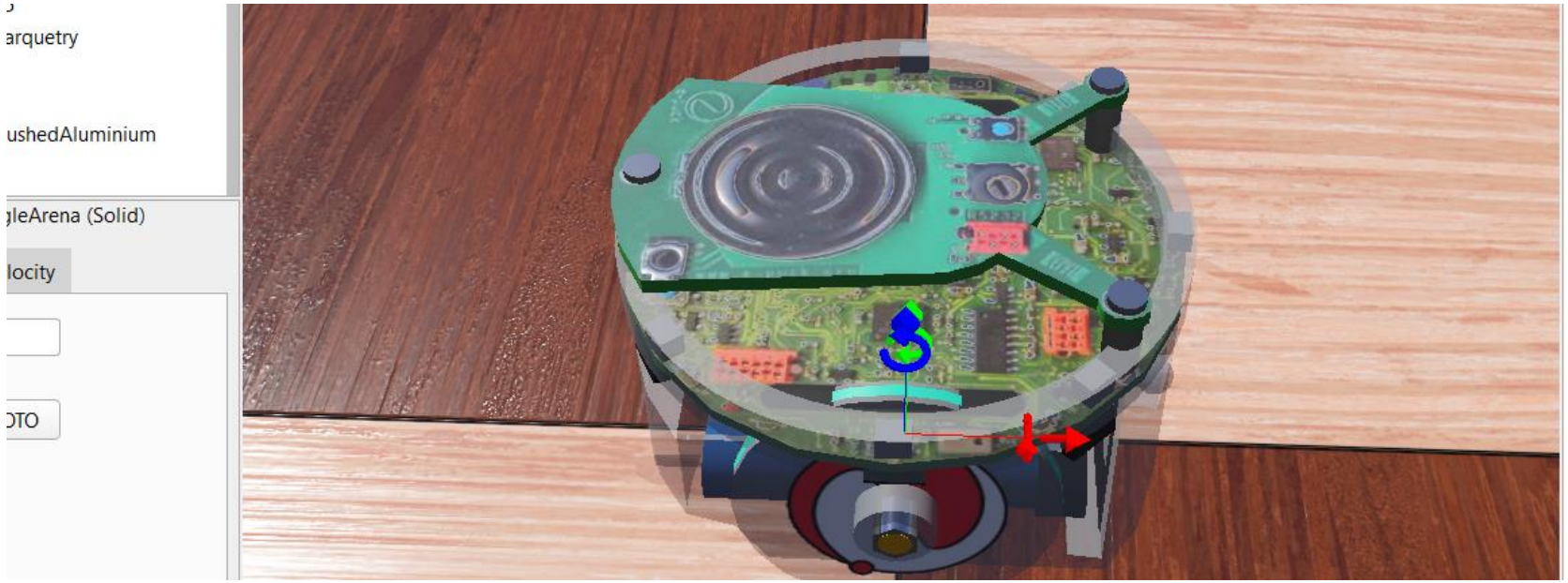
Two wheel motors

Ready for control

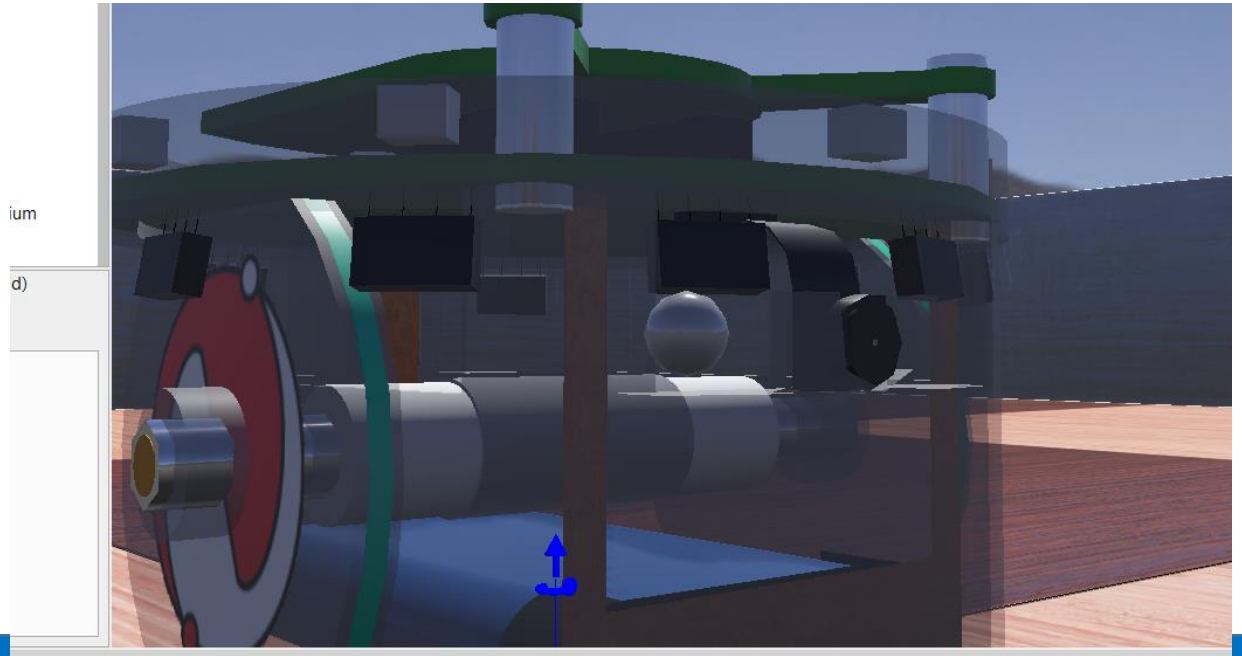


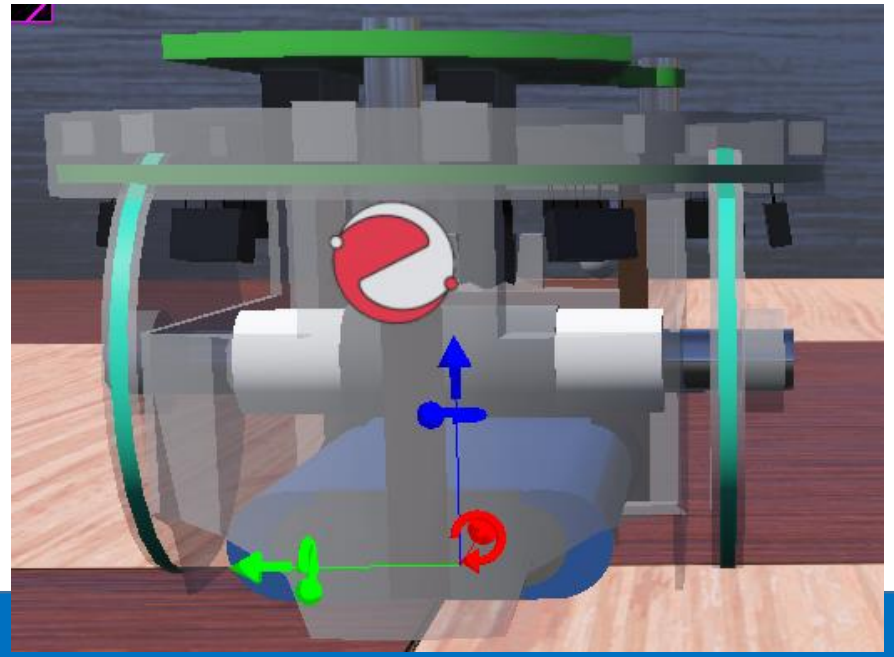
Add an e-puck robot
*In the **Scene Tree***
*Click **Add → Robot***
Navigate to:
robots → gctronic → e-puck
*Insert **E-puck***

You now have:
Differential-drive robot
Two wheel motors
Ready for control



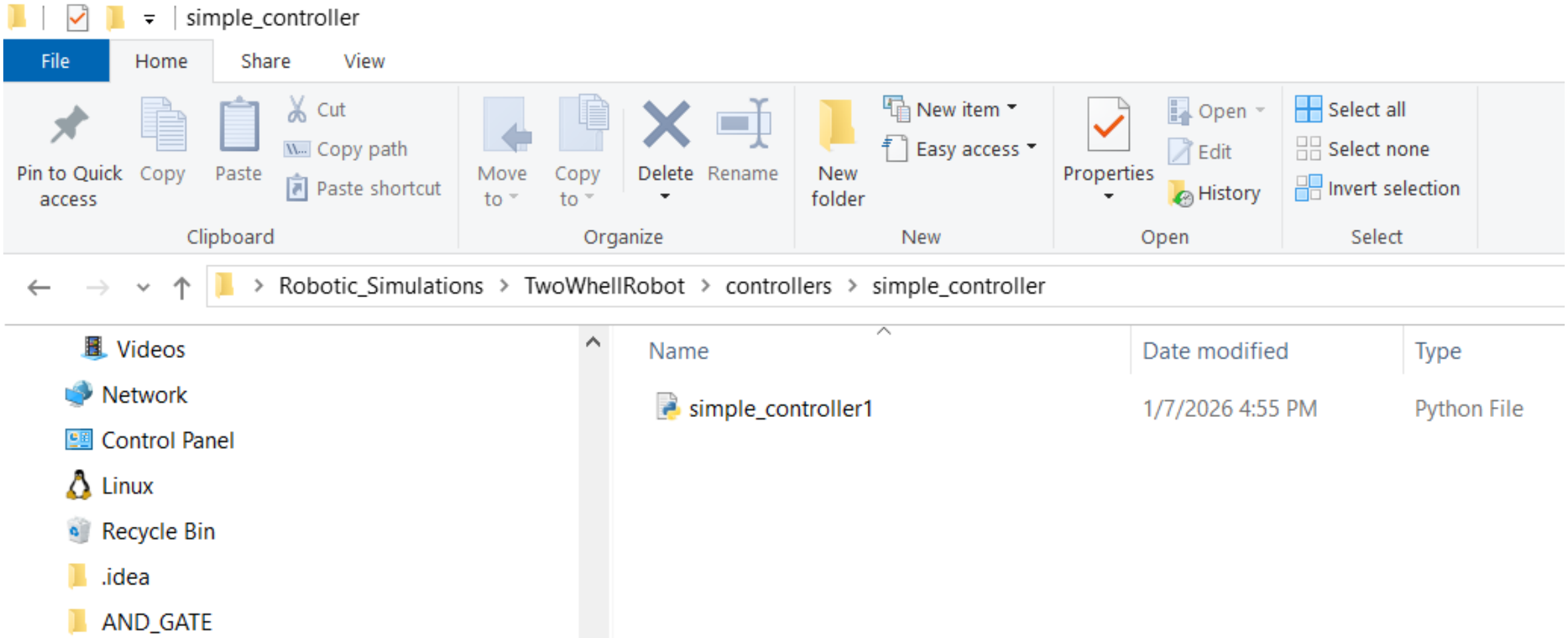

```
# distribute
# WITHOUT WA
# See the Li
# limitation

"""Demonstra
import sys
import tempf
try:
    import i
    from ikp
except Impor
    sys.exit


import math
from control

if ikpy.__ve
    sys.exit
```



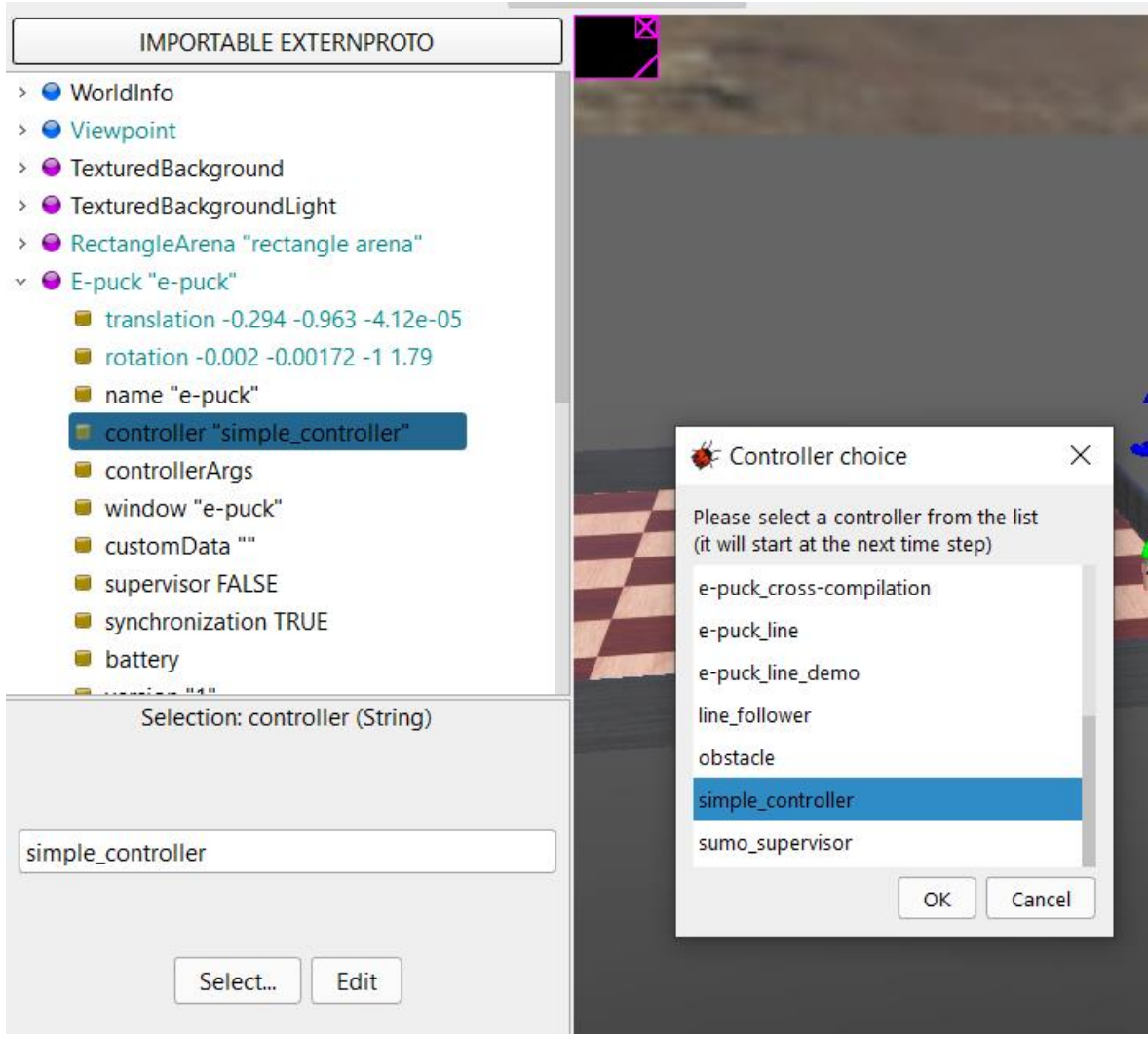


Assign a Controller to the Robot

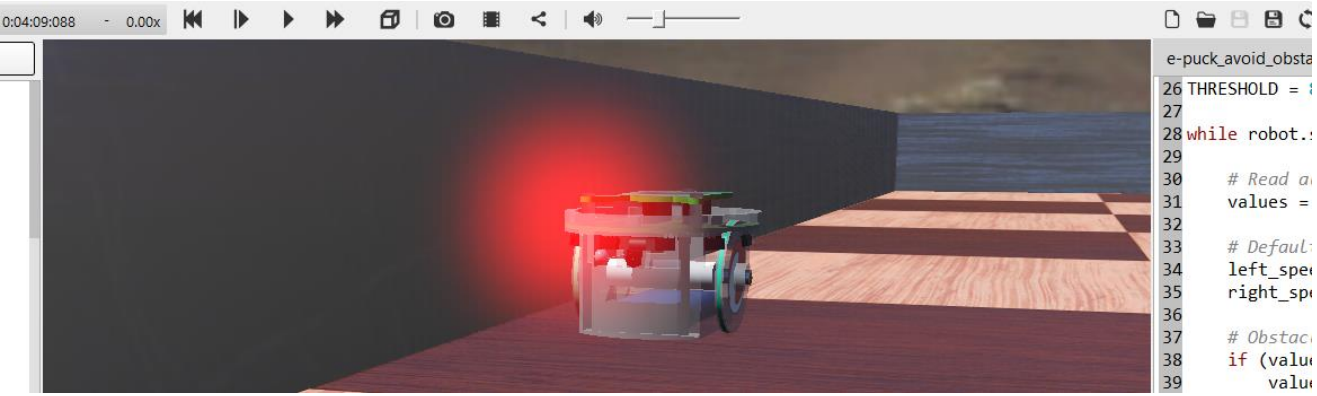
Aller dans le dossier de travail et créer un dossier appelé simple_controller puis un fichier simple_controller.py



Assign a Controller to the Robot
Click the robot in the **Scene Tree**
In the **Field Editor**, find: Controller
Click on select then choose simple_controller.py



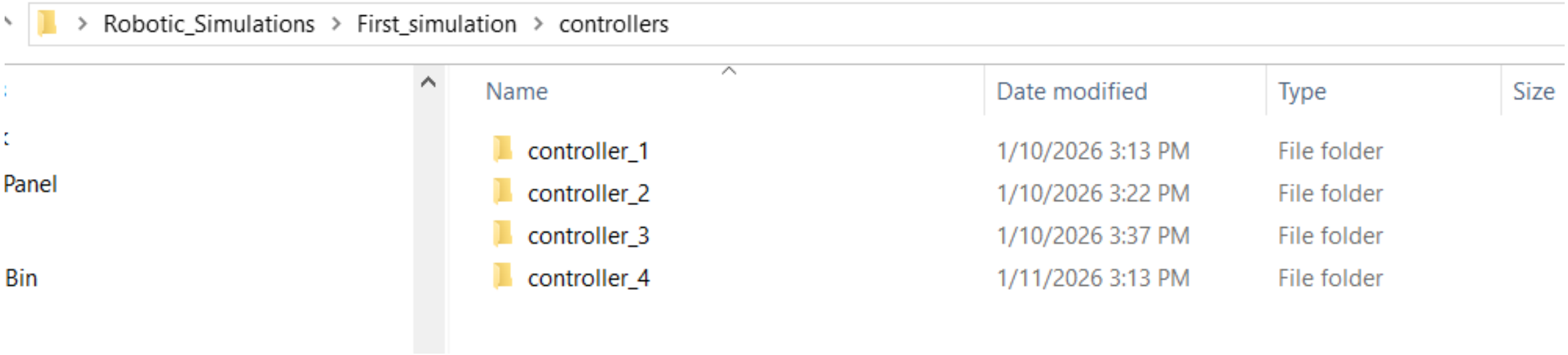
Click on the run button to lunch the simulation



Assign a Controller to the Robot

Go to the controllers folder in the working directory and add two other controllers for performing the following tasks:

- ❑ Controller_2. py: The robot turns around itself
- ❑ Controller_3.py: the robot performs a circular motion
- ❑ controller_4.py: the robot performs a rectangular motion



Solution for the circular and turn around controllers

```
from controller import Robot

# Create the Robot instance
robot = Robot()

# Get simulation time step
timestep = int(robot.getBasicTimeStep())

# Get motors
left_motor = robot.getDevice("left wheel motor")
right_motor = robot.getDevice("right wheel motor")

# Set motors to velocity control mode
left_motor.setPosition(float('inf'))
right_motor.setPosition(float('inf'))
```

```
# Initial speeds
left_speed = 0.0
right_speed = 0.0

# Maximum motor speed (depends on robot)
MAX_SPEED = 6.28  # rad/s

# Main control loop
while robot.step(timestep) != -1:

    # -------- Motion selection --------

    # 2. Rotate in place (uncomment to test)
    left_speed = 0.5 * MAX_SPEED
    right_speed = -0.5 * MAX_SPEED

    # 3. Circular motion (uncomment to test)
    left_speed = 0.3 * MAX_SPEED
    right_speed = 0.6 * MAX_SPEED

    # Send commands to motors
    left_motor.setVelocity(left_speed)
    right_motor.setVelocity(right_speed)
```

Solution for the rectangular motion

C:\Users\LASSIOUI\Desktop\Robotic_Simulations\Differential_Robot\controllers\controller_2\controller_2.py - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

System_inst.vhd | CPU_1_inst.vhd | CPU_1.vhd | CPU_1_inst.vhd | CPU2_SLAVE_inst.vhd | controller_1.py | controller_2.py

```python
from controller import Robot

# Create the Robot instance
robot = Robot()

# Get simulation time step
timestep = int(robot.getBasicTimeStep())

# Get motors
left_motor = robot.getDevice("left wheel motor")
right_motor = robot.getDevice("right wheel motor")

# Velocity control mode
left_motor.setPosition(float('inf'))
right_motor.setPosition(float('inf'))

# Motor parameters
MAX_SPEED = 6.28  # rad/s

# Time parameters (in seconds)
FORWARD_TIME = 5.0   # time to go straight
TURN_TIME = 1.0      # time to turn 90 degrees (to adjust experimentally)

# State variables
state = "FORWARD"
state_start_time = robot.getTime()
```

```python
# Main control loop
while robot.step(timestep) != -1:

    current_time = robot.getTime()
    elapsed_time = current_time - state_start_time

    if state == "FORWARD":
        # Move straight
        left_motor.setVelocity(0.5 * MAX_SPEED)
        right_motor.setVelocity(0.5 * MAX_SPEED)

        if elapsed_time >= FORWARD_TIME:
            state = "TURN"
            state_start_time = current_time

    elif state == "TURN":
        # Turn on the spot (left turn)
        left_motor.setVelocity(-0.3 * MAX_SPEED)
        right_motor.setVelocity(0.3 * MAX_SPEED)

        if elapsed_time >= TURN_TIME:
            state = "FORWARD"
            state_start_time = current_time
```

To do

Obstacle avoidance python code

Obstacle avoidance python program

```python
from controller import Robot

# Create robot instance
robot = Robot()
timestep = int(robot.getBasicTimeStep())

# Motors
left_motor = robot.getDevice("left wheel motor")
right_motor = robot.getDevice("right wheel motor")

left_motor.setPosition(float('inf'))
right_motor.setPosition(float('inf'))

MAX_SPEED = 6.28

# e-puck distance sensors
ps_names = ["ps0", "ps1", "ps2", "ps3", "ps4", "ps5", "ps6", "ps7"]
ps = []

for name in ps_names:
    sensor = robot.getDevice(name)
    sensor.enable(timestep)
    ps.append(sensor)

# Obstacle detection threshold
THRESHOLD = 80.0
```

```python
while robot.step(timestep) != -1:
    # Read all proximity sensors
    values = [sensor.getValue() for sensor in ps]
    # Default motion: forward
    left_speed = 0.5 * MAX_SPEED
    right_speed = 0.5 * MAX_SPEED
    # Obstacle in front (ps0, ps1, ps6, ps7)
    if (values[0] > THRESHOLD or values[1] > THRESHOLD or
        values[6] > THRESHOLD or values[7] > THRESHOLD):
        left_speed = -0.4 * MAX_SPEED
        right_speed = 0.4 * MAX_SPEED
    # Obstacle on the right (ps2)
    elif values[2] > THRESHOLD:
        left_speed = 0.5 * MAX_SPEED
        right_speed = 0.2 * MAX_SPEED
    # Obstacle on the left (ps5)
    elif values[5] > THRESHOLD:
        left_speed = 0.2 * MAX_SPEED
        right_speed = 0.5 * MAX_SPEED
    # Apply motor commands
    left_motor.setVelocity(left_speed)
    right_motor.setVelocity(right_speed)
```