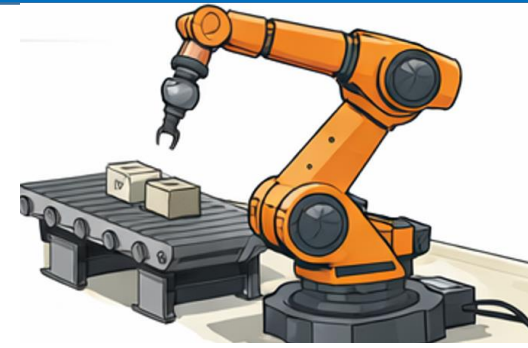


Robotique et vision par ordinateur



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



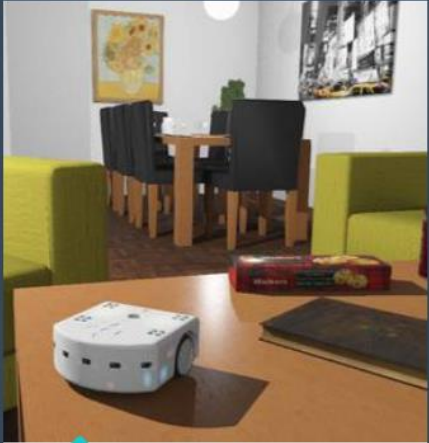
Software to install: Webot simulator

Available for free at : www.cyberbotics.com

← ↻ <https://cyberbotics.com> ⋮ ☆ 📄 ⚙️ ☆ 👤 ⋮ Chat

Cyberbotics Services Webots News Blog Download Documentation ▾ Contact us

Discover bundled demos running out-of-the-box.



Simulate **educational robots** in your classroom to train your students.



Simulate **mobile manipulators** and train your AI model.



Interface simulated robots with your favorite robotics framework such as **ROS**.



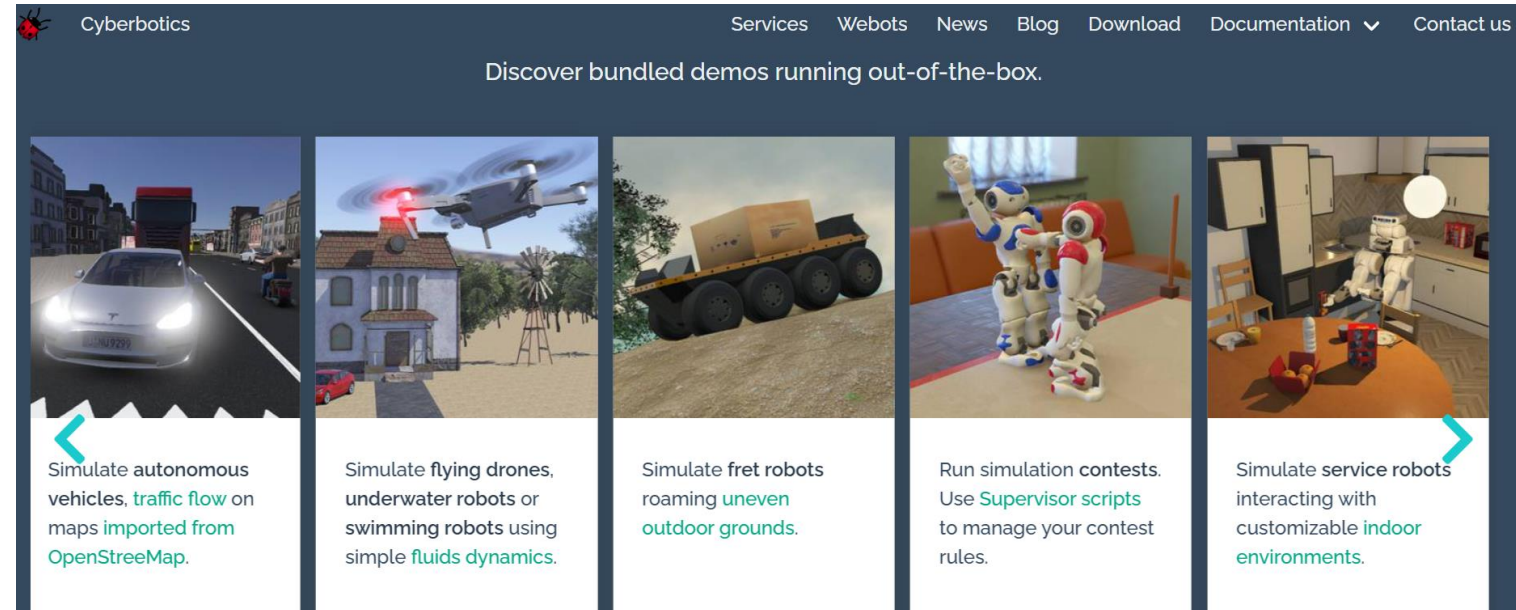
Simulate **assembly lines** to prototype your production plant.



Simulate **complex physics** such as **mechanum wheels** or **tracks** simply from a high level of abstraction.

Why Use Webots?

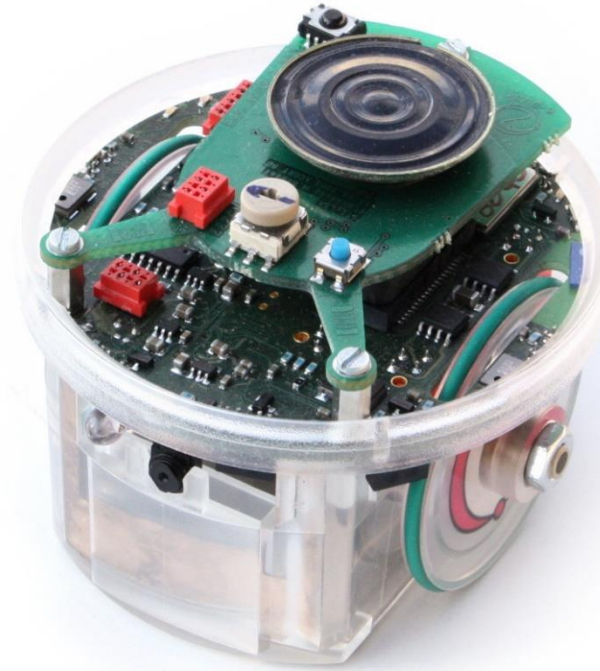
- ❑ Physics-based robot simulator
- ❑ No hardware required
- ❑ Fast testing and visualization
- ❑ Supports Python, C, C++
- ❑ Widely used in education and research



The Simplest Robot: Differential-Drive Robot

Structure:

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

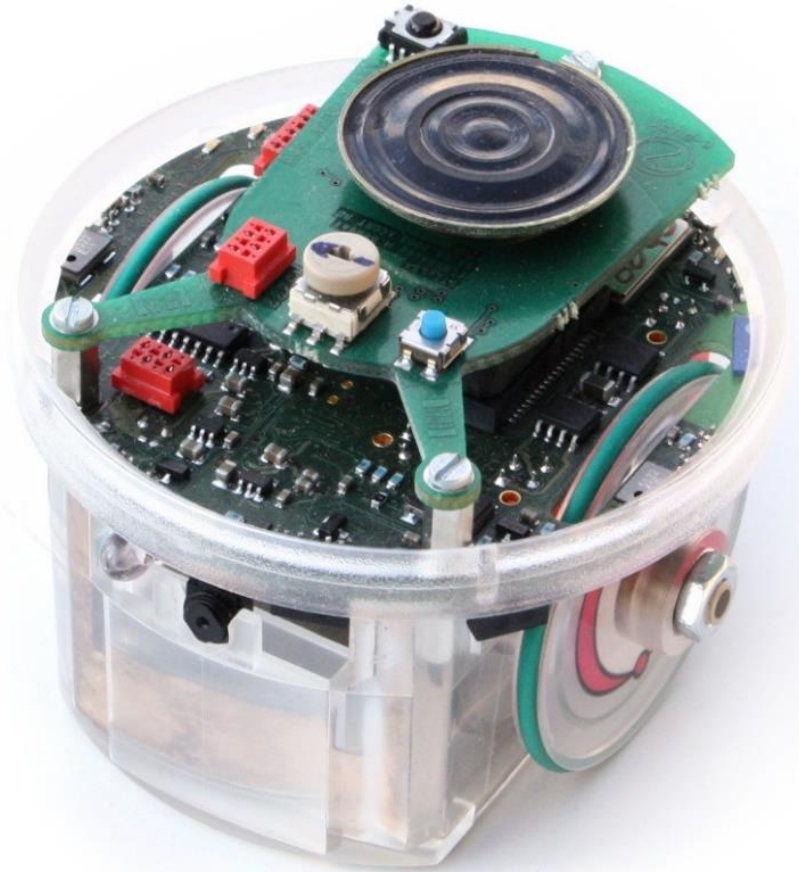


The e-puck is a small (7 cm) differential wheeled mobile robot. It was originally designed for micro-engineering education by Michael Bonani and Francesco Mondada at the ASL laboratory of Prof.

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:

- ω_L : left wheel angular speed
- ω_R : 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
$\omega_L = \omega_R$	Straight line
$\omega_L = -\omega_R$	Rotation in place
$\omega_L \neq \omega_R$	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

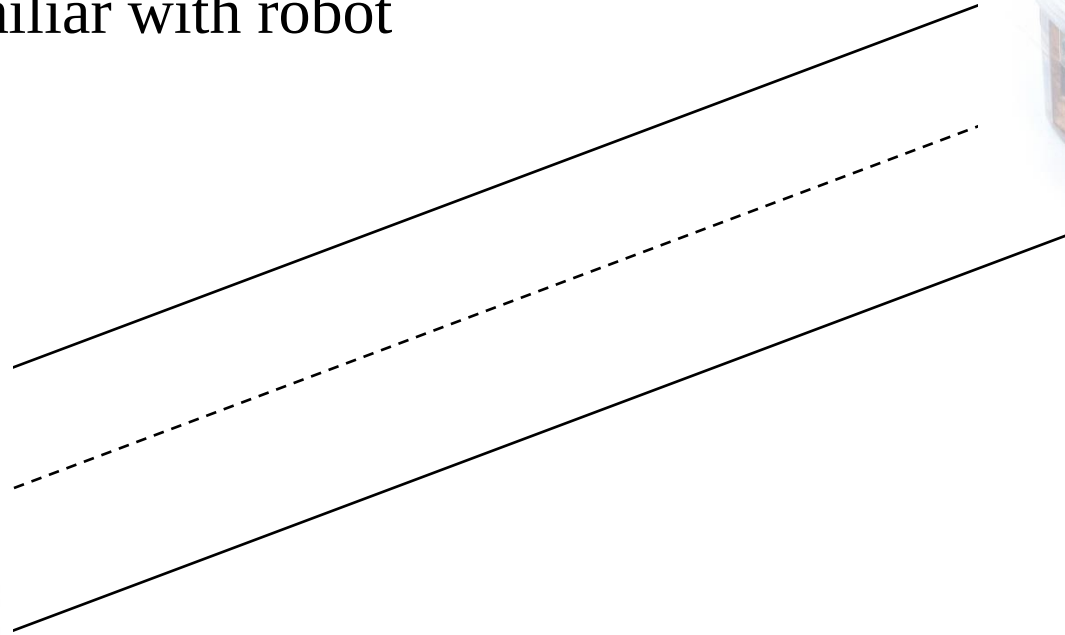
Controller_1 – straight forward move

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



Final position



Initial 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'))
```

- 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

- 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)

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.
- $6.28 \text{ rad/s} \approx 2\pi \text{ rad/s} \approx 1 \text{ revolution per second}$
- This value is robot-dependent and usually specified in the robot's datasheet or model.

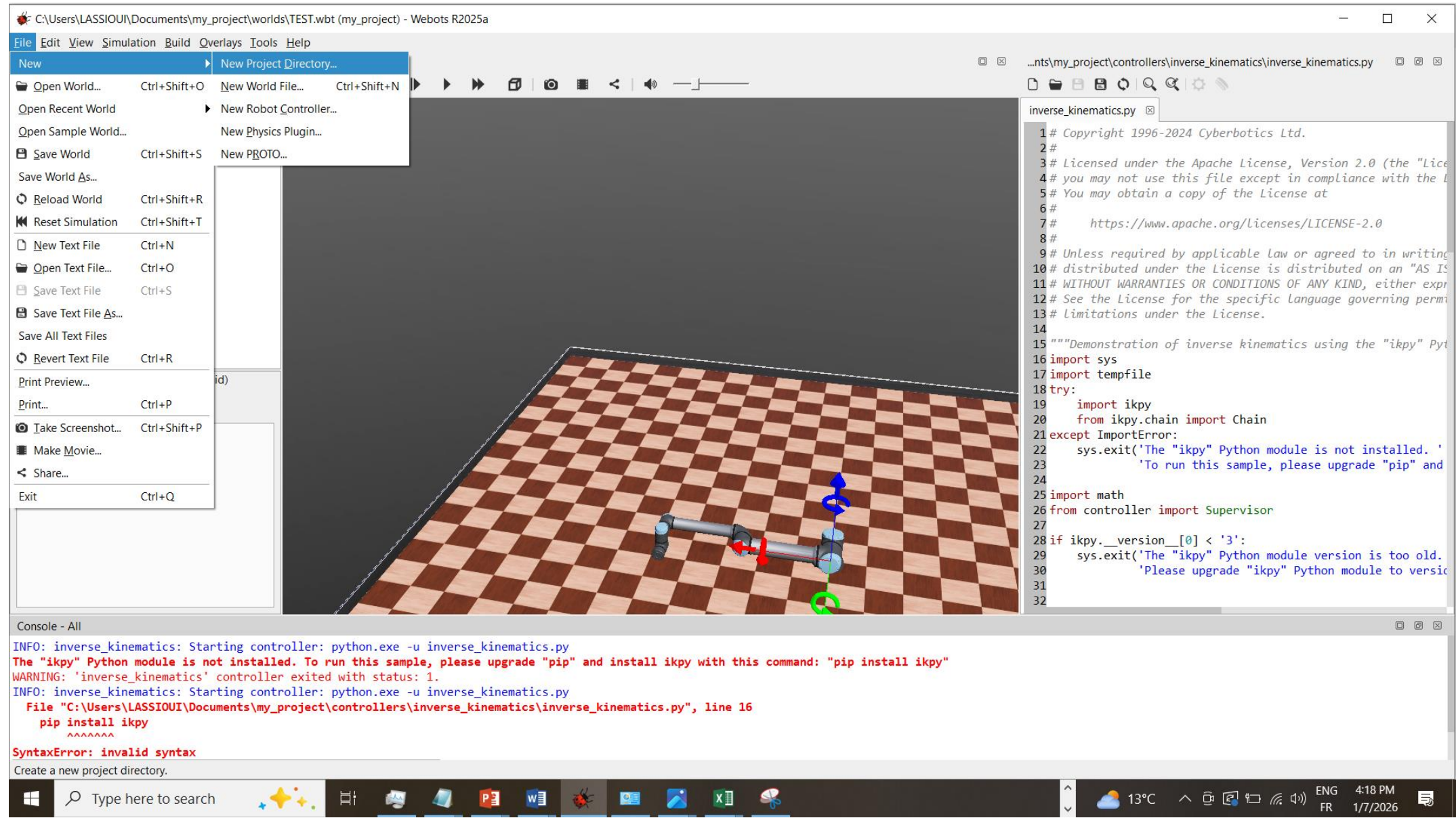
- This is the **real-time control loop**.
- `robot.step(timestep)` :
 - Advances the simulation by one timestep
 - Returns `-1` when the simulation ends
- All sensing, decision-making, and actuation must occur inside this loop.

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

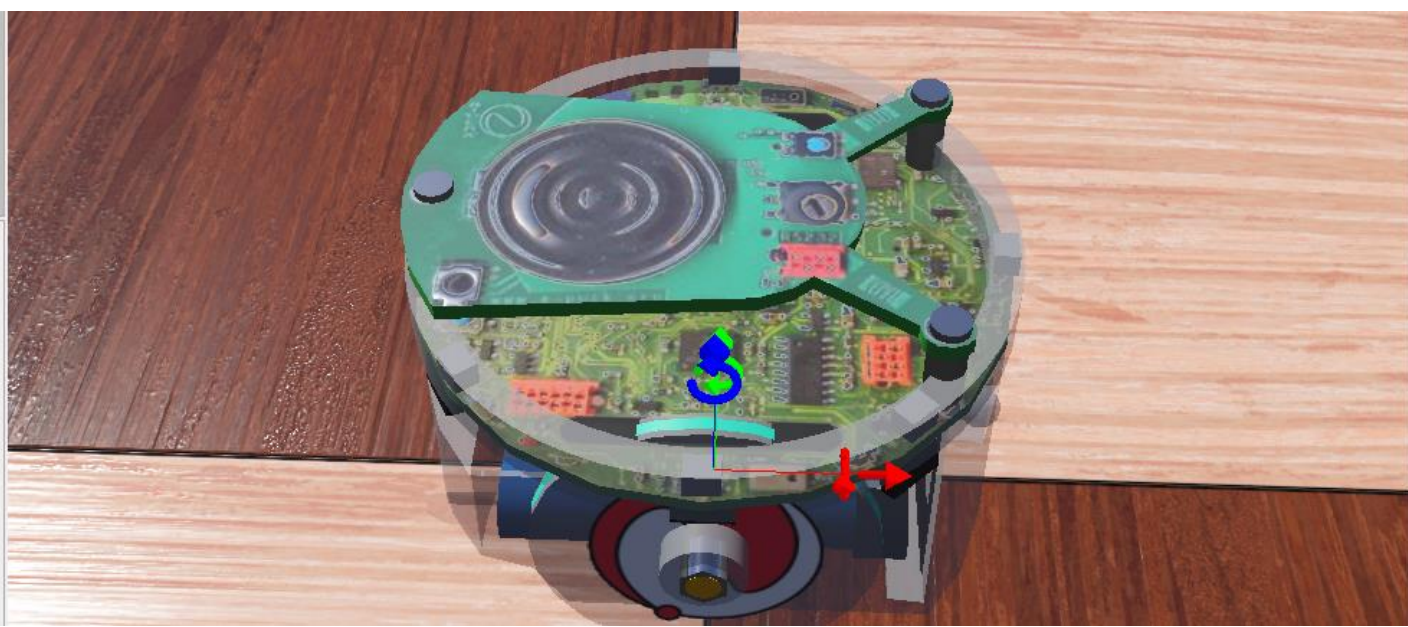
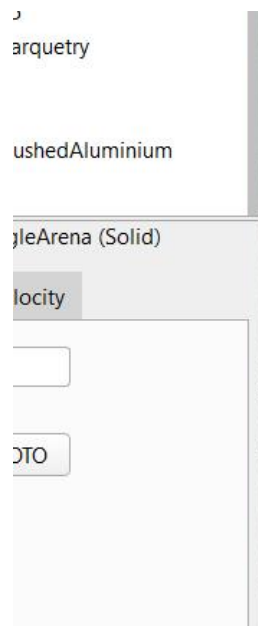
Ready for control

The screenshot displays a software interface with a scene tree on the left and a central workspace. The scene tree lists various objects, with 'RectangleArena "rectangle arena"' selected. Below the tree, a table shows the selected object's properties: translation (0 0 0), rotation (0 0 1 0), name ("rectangle arena"), contactMaterial ("default"), floorSize (1 1), floorTileSize (0.5 0.5), floorAppearance (Parquetry), and wallThickness (0.01). A button labeled 'Print EXTERNPROTO' is visible. In the center, a dialog box titled 'Add a node' is open, showing a tree of robot models. The 'e-puck' model under the 'gctronic' category is selected. To the right of the dialog, a preview of the e-puck robot is shown, along with its description: 'EPFL educational and research mini mobile robot. The e-puck is powered by a dsPIC processor and features a very large number of sensors in its basic configuration. More info here: <http://www.e-puck.org>.' Below the preview, documentation links and a license notice are provided. On the far right, a code editor shows a Python script named 'inverse_kinematics.py' with a copyright notice and a license statement.

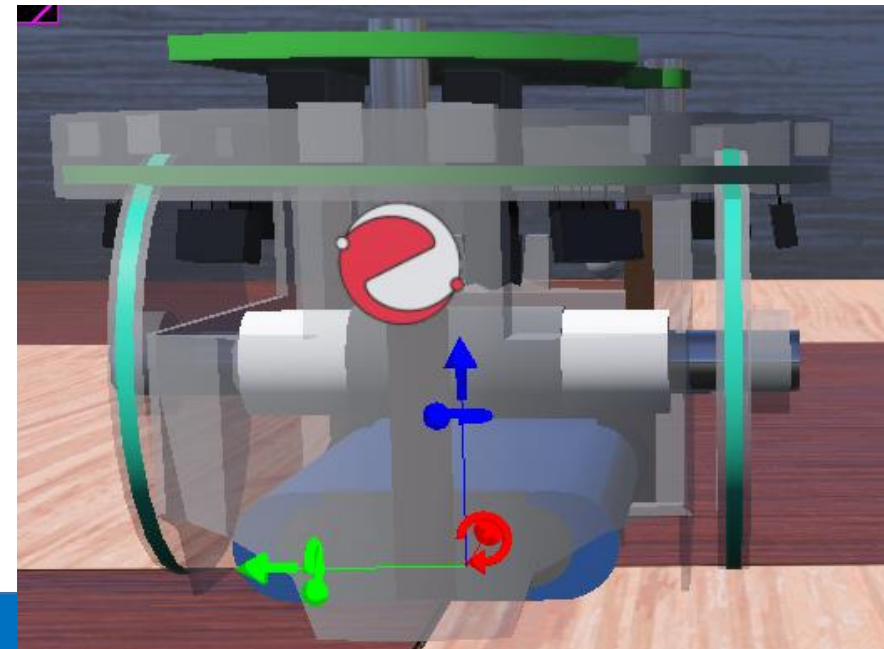
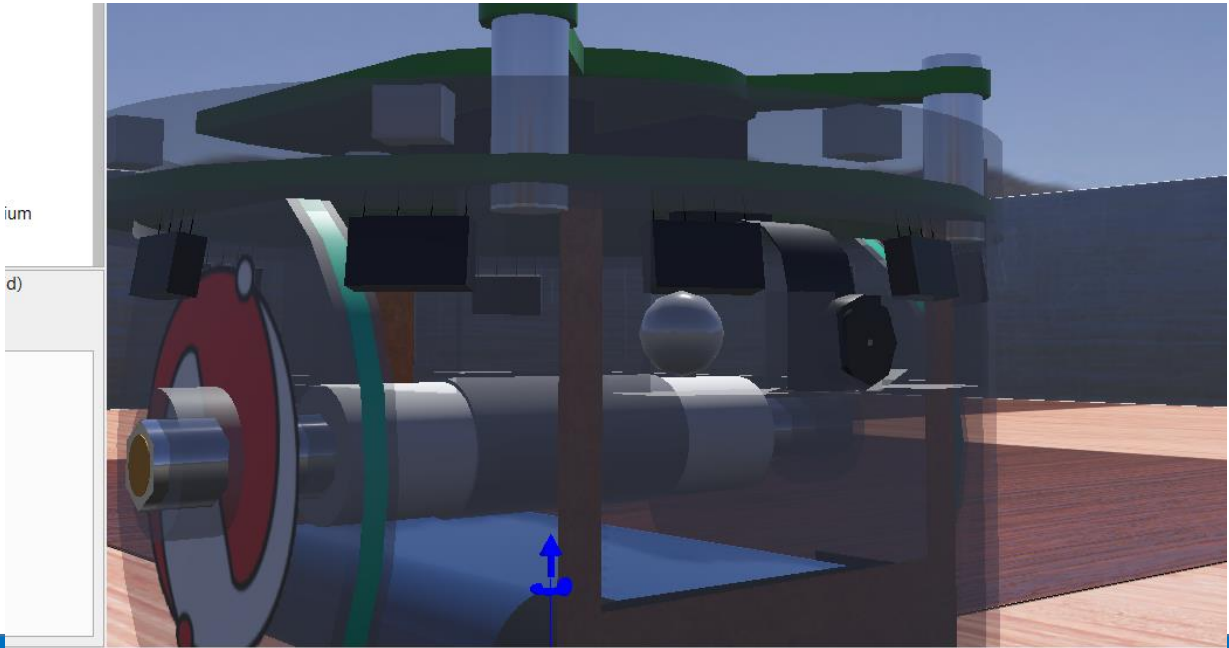
```
1 # Copyright 1996-2024
2 #
3 # Licensed under the
4 # you may not use thi
5 # You may obtain a co
6 #
7 # https://www.apa
8 #
9 # Unless required by
10 # distributed under t
11 # WITHOUT WARRANTIES
12 # See the License for
13 # limitations under t
14
15 """Demonstration of i
16 import sys
17 import tempfile
18 try:
19     import ikpy
20     from ikpy.chain i
21 except ImportError:
22     sys.exit('The "ik
23         'To run
24
25 import math
26 from controller impor
27
28 if ikpy.__version__[0
29     sys.exit('The "ik
30         'Please
31
32
```

*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*

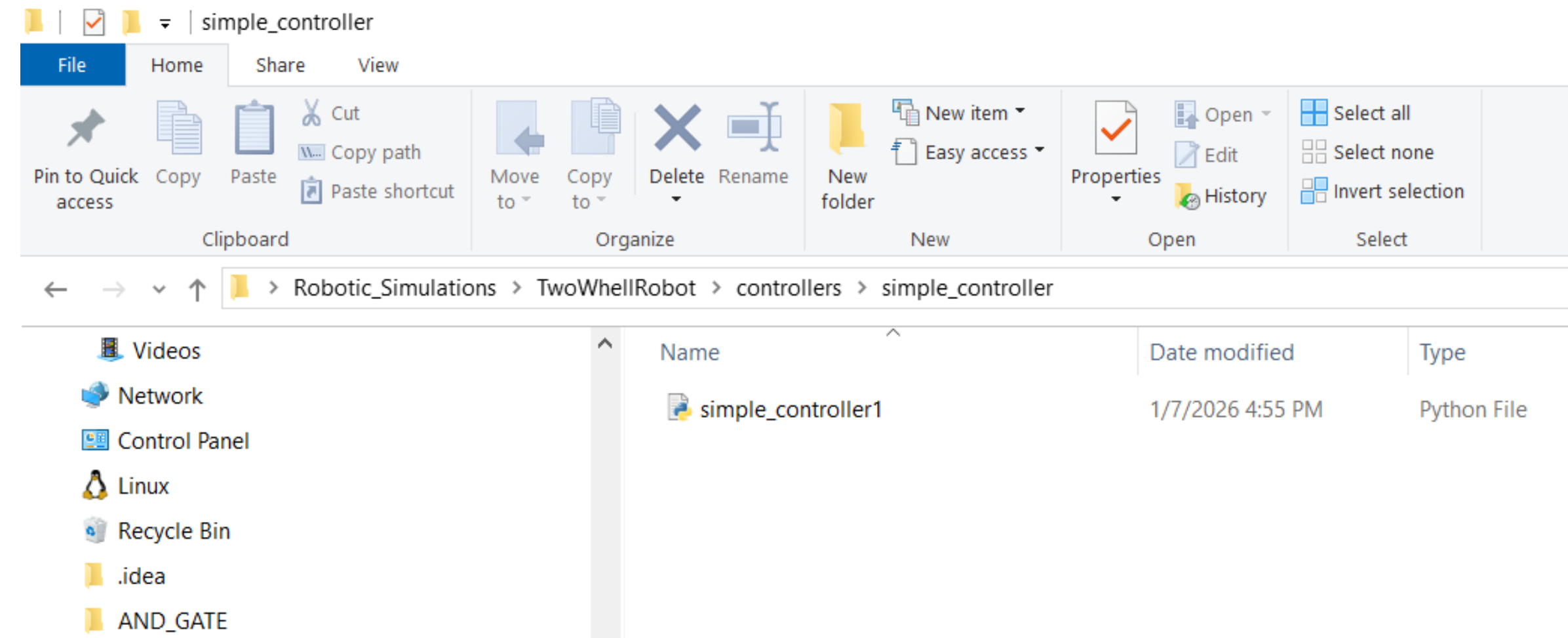


```
10 # distribute
11 # WITHOUT W/
12 # See the Li
13 # limitation
14
15 """Demonstrc
16 import sys
17 import tempf
18 try:
19     import i
20     from ikp
21 except Impor
22     sys.exit
23
24
25 import math
26 from control
27
28 if ikpy.__ve
29     sys.exit
30
31
```



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



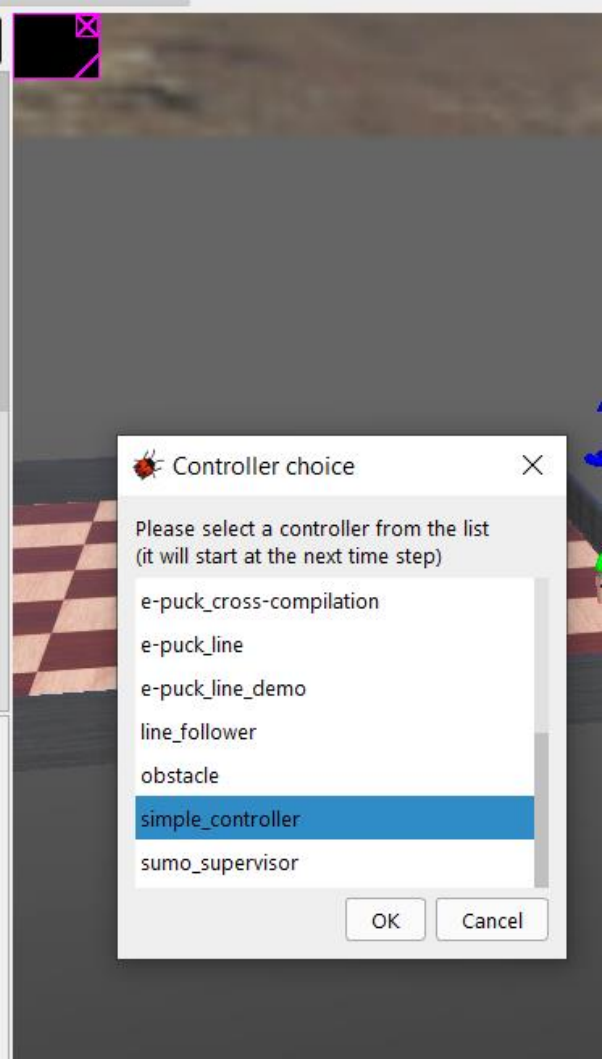
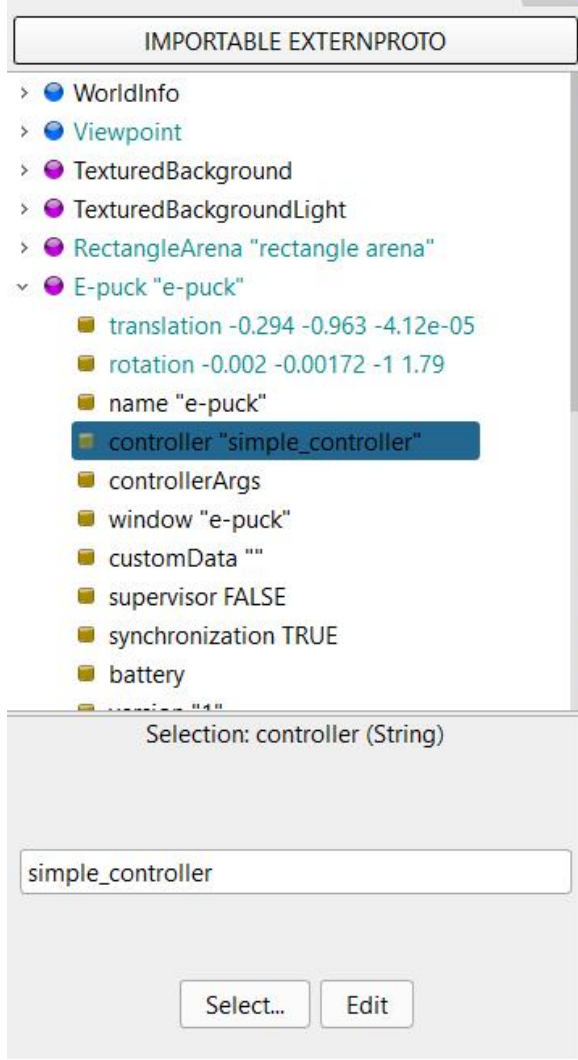
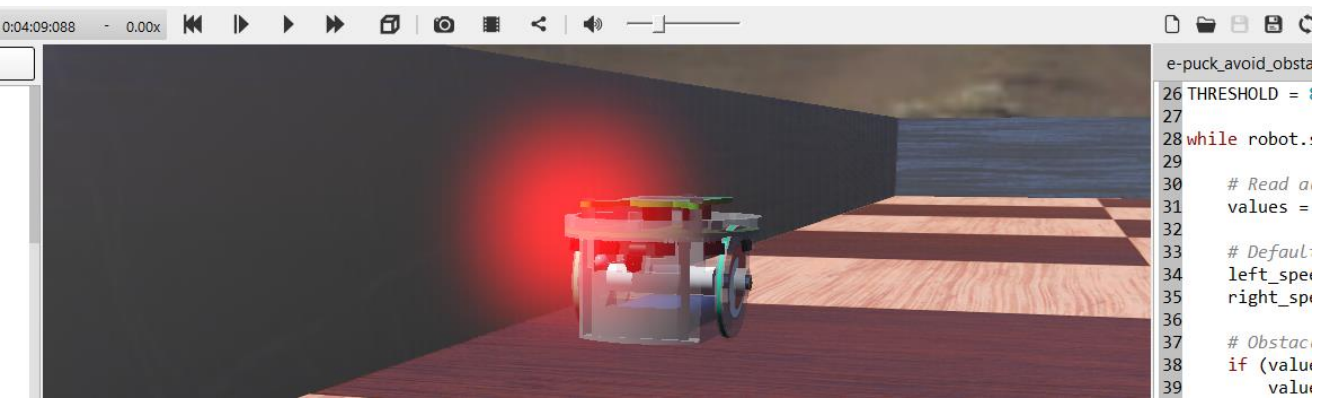
Assign a Controller to the Robot

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

📁 > Robotic_Simulations > First_simulation > controllers					
	Name	Date modified	Type	Size	
	📁 controller_1	1/10/2026 3:13 PM	File folder		
	📁 controller_2	1/10/2026 3:22 PM	File folder		
	📁 controller_3	1/10/2026 3:37 PM	File folder		
	📁 controller_4	1/11/2026 3:13 PM	File folder		

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\LASSIOU\Desktop\Robotic_Simulations\Differential_Robot\controllers\controller_2\controller_2.py - Notepad++

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

```
1 from controller import Robot
2
3 # Create the Robot instance
4 robot = Robot()
5
6 # Get simulation time step
7 timestep = int(robot.getBasicTimeStep())
8
9 # Get motors
10 left_motor = robot.getDevice("left wheel motor")
11 right_motor = robot.getDevice("right wheel motor")
12
13 # Velocity control mode
14 left_motor.setPosition(float('inf'))
15 right_motor.setPosition(float('inf'))
16
17 # Motor parameters
18 MAX_SPEED = 6.28 # rad/s
19
20 # Time parameters (in seconds)
21 FORWARD_TIME = 5.0 # time to go straight
22 TURN_TIME = 1.0 # time to turn 90 degrees (to adjust experimentally)
23
24 # State variables
25 state = "FORWARD"
26 state_start_time = robot.getTime()
27
```

```
27
28 # Main control loop
29 while robot.step(timestep) != -1:
30
31     current_time = robot.getTime()
32     elapsed_time = current_time - state_start_time
33
34     if state == "FORWARD":
35         # Move straight
36         left_motor.setVelocity(0.5 * MAX_SPEED)
37         right_motor.setVelocity(0.5 * MAX_SPEED)
38
39         if elapsed_time >= FORWARD_TIME:
40             state = "TURN"
41             state_start_time = current_time
42
43     elif state == "TURN":
44         # Turn on the spot (left turn)
45         left_motor.setVelocity(-0.3 * MAX_SPEED)
46         right_motor.setVelocity(0.3 * MAX_SPEED)
47
48         if elapsed_time >= TURN_TIME:
49             state = "FORWARD"
50             state_start_time = current_time
51
```

To do
Obstacle avoidance python code

Obstacle avoidance python program

```
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
```

```
28 while robot.step(timestep) != -1:
29     # Read all proximity sensors
30     values = [sensor.getValue() for sensor in ps]
31     # Default motion: forward
32     left_speed = 0.5 * MAX_SPEED
33     right_speed = 0.5 * MAX_SPEED
34     # Obstacle in front (ps0, ps1, ps6, ps7)
35     if (values[0] > THRESHOLD or values[1] > THRESHOLD or
36         values[6] > THRESHOLD or values[7] > THRESHOLD):
37         left_speed = -0.4 * MAX_SPEED
38         right_speed = 0.4 * MAX_SPEED
39     # Obstacle on the right (ps2)
40     elif values[2] > THRESHOLD:
41         left_speed = 0.5 * MAX_SPEED
42         right_speed = 0.2 * MAX_SPEED
43     # Obstacle on the left (ps5)
44     elif values[5] > THRESHOLD:
45         left_speed = 0.2 * MAX_SPEED
46         right_speed = 0.5 * MAX_SPEED
47     # Apply motor commands
48     left_motor.setVelocity(left_speed)
49     right_motor.setVelocity(right_speed)
```