UNIVERSIDADE DA CORUÑA

facultade de informática da coruña

Introducción al simulador Webots

Grao en Enxeñaría Informática - 4º curso
ROBÓTICA

Simulador Webots

- Simulador 3D para robots móviles.
- Originalmente diseñado para investigación de algoritmos de control en robots móviles.
- Desde 2018, simulador de código abierto con licencia Apache 2.0.
- Multiplataforma: Windows, MacOS, Linux.
  - Instalación en Linux: preferible instalación desde paquete Debian (con APT) a instalación con paquete Snap.
- https://cyberbotics.com/doc/guide/index

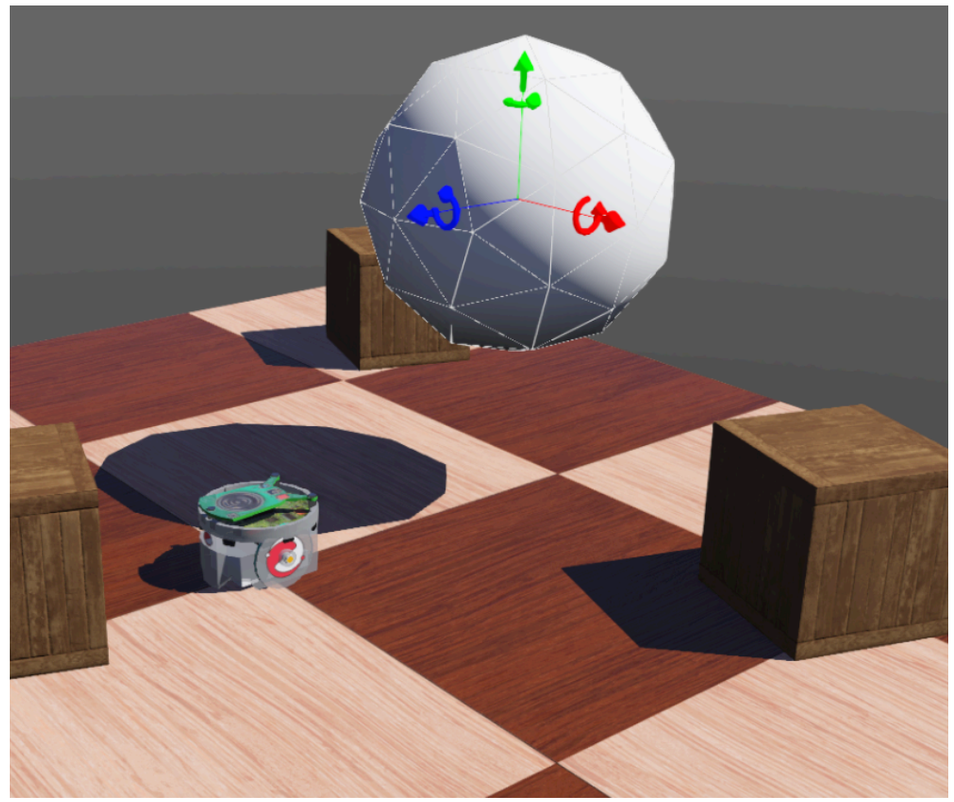
Simulador Webots

- Permite diseñar escenarios virtuales en 3D (worlds) con motor de física (masas, coeficientes de fricción, etc.)

- Permite añadir al entorno **objetos pasivos** y **objetos móviles (robots)**.

- Incluye **modelos de multitud de robots comerciales**, incluyendo sus dispositivos de sensorización y actuación.

- El usuario puede **programar el controlador** de cada robot de forma independiente para conseguir el comportamiento deseado.



Uso del simulador en las prácticas

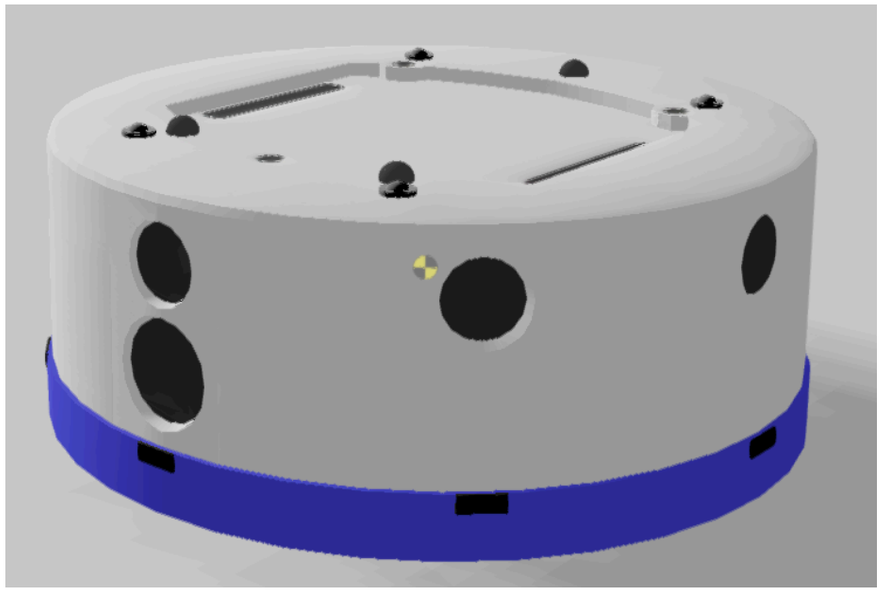
- Se utilizará un modelo de robot con ruedas y múltiples sensores para monitorización del entorno: **K-Team's Khepera IV**.

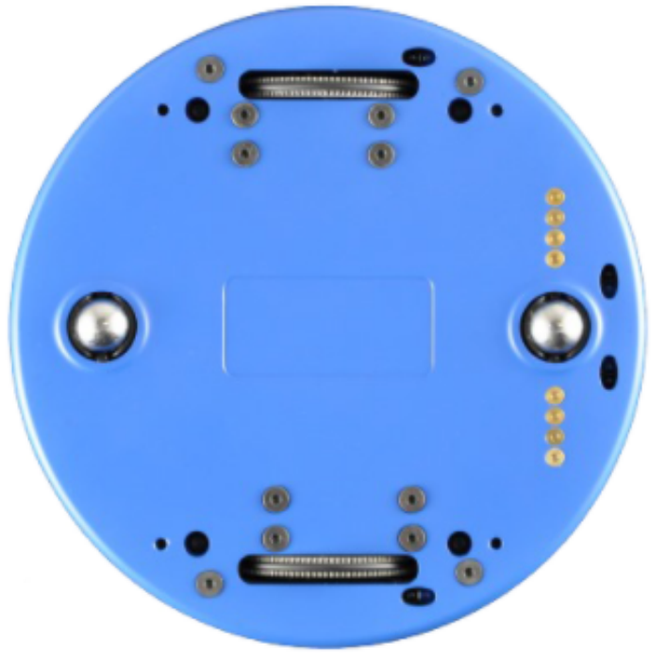
- Se utilizarán escenarios (worlds) adaptados al problema a resolver en la práctica.

- El alumno programará el control del robot (controlador) necesario para cumplir con los objetivos de la práctica.

- Se puede comenzar a experimentar con el escenario de ejemplo del robot Khepera-VI ya disponible.

Robot K-Team's Khepera IV





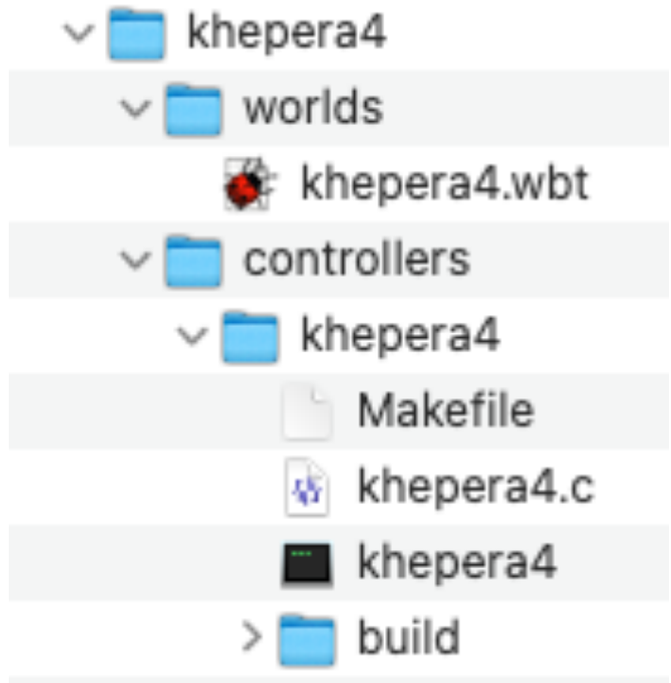
- Referencia del modelo de simulación para Webots:
  - https://www.cyberbotics.com/doc/guide/khepera4#khepera4-wbt
- Página web del producto:
  - https://www.k-team.com/khepera-iv

Robot Khepera IV

- Motores para el movimiento:
  - **2 motores** con encoders incrementales (left wheel motor, right wheel motor).
  - **2 sensores de posición** (left wheel sensor, right wheel sensor).
- Sensores principales:
  - **8 sensores infrarrojos** de proximidad y de luz ambiente, con un rango **desde 0 cm a 25cm**.
    - La medida de infrarrojo varía con el **inverso de la distancia**.
  - 4 sensores infrarrojos de proximidad al suelo para seguimiento de líneas y evitar caídas.
  - **5 sensores de ulltrasonidos** con un rango **desde 25cm a 2m**.
  - **Acelerómetro** y **giroscopio de 3 ejes**.
  - **Cámara color** (752×480 pixels, 30FPS).
- Otros datos:
  - 3 LED RGB programables encima del robot.
  - Tamaño: 140mm de diámetro, 58mm de altura.

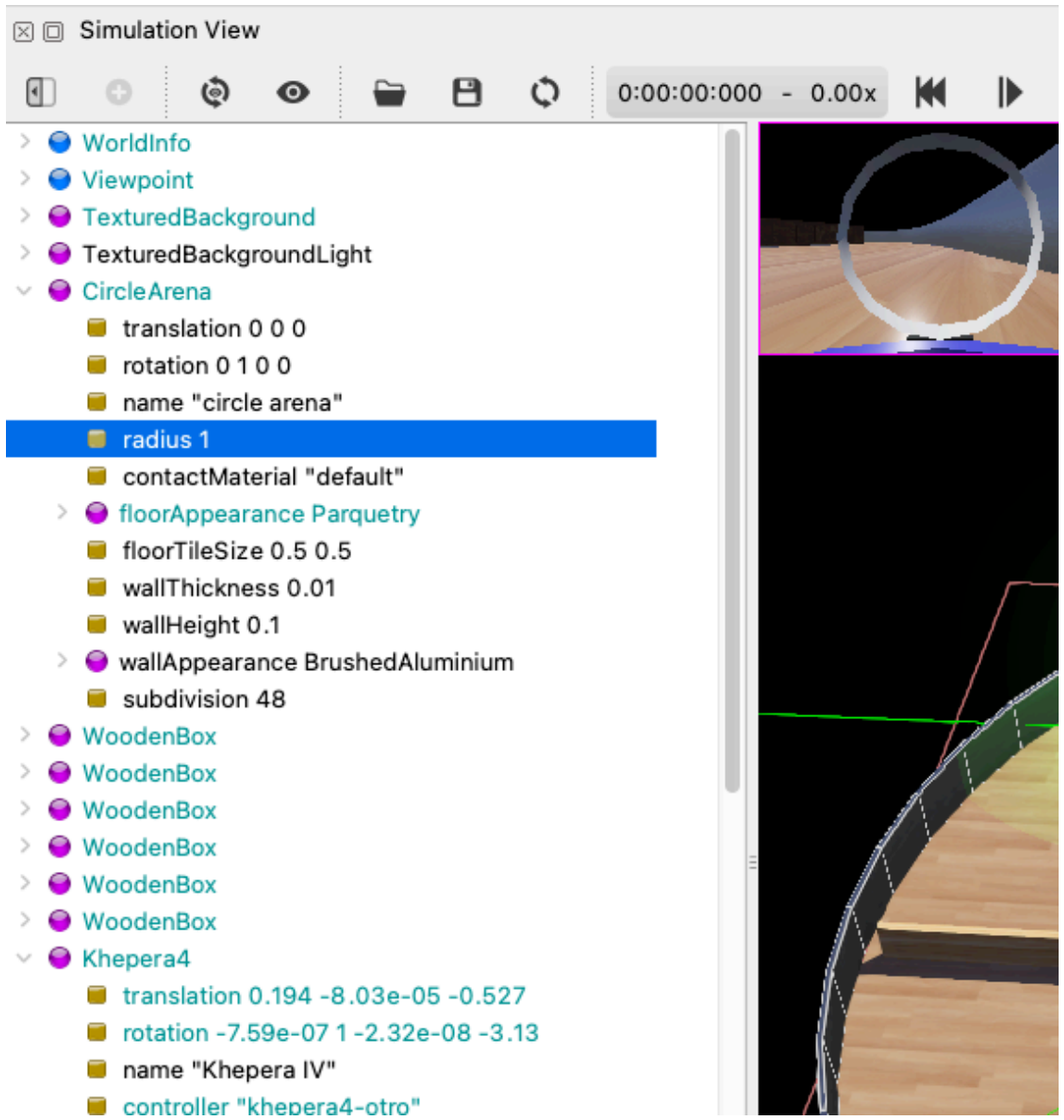
Escenarios (worlds)

- Los escenarios se almacenan en ficheros .wbt.
- Pueden cargarse escenarios de ejemplo disponibles en Webots.
- Si se desea modificar estos escenarios o el controlador del robot es necesario que se guarden en directorios del usuario.
  - El propio Webots nos preguntará al intentar guardar.
- Al compilar los controladores, el resultado se genera en nuestra estructura de directorios de usuario.



Escenarios (worlds)

- Podemos editar las propiedades de los elementos del escenario (estáticos y robots) desde el árbol de objetos.

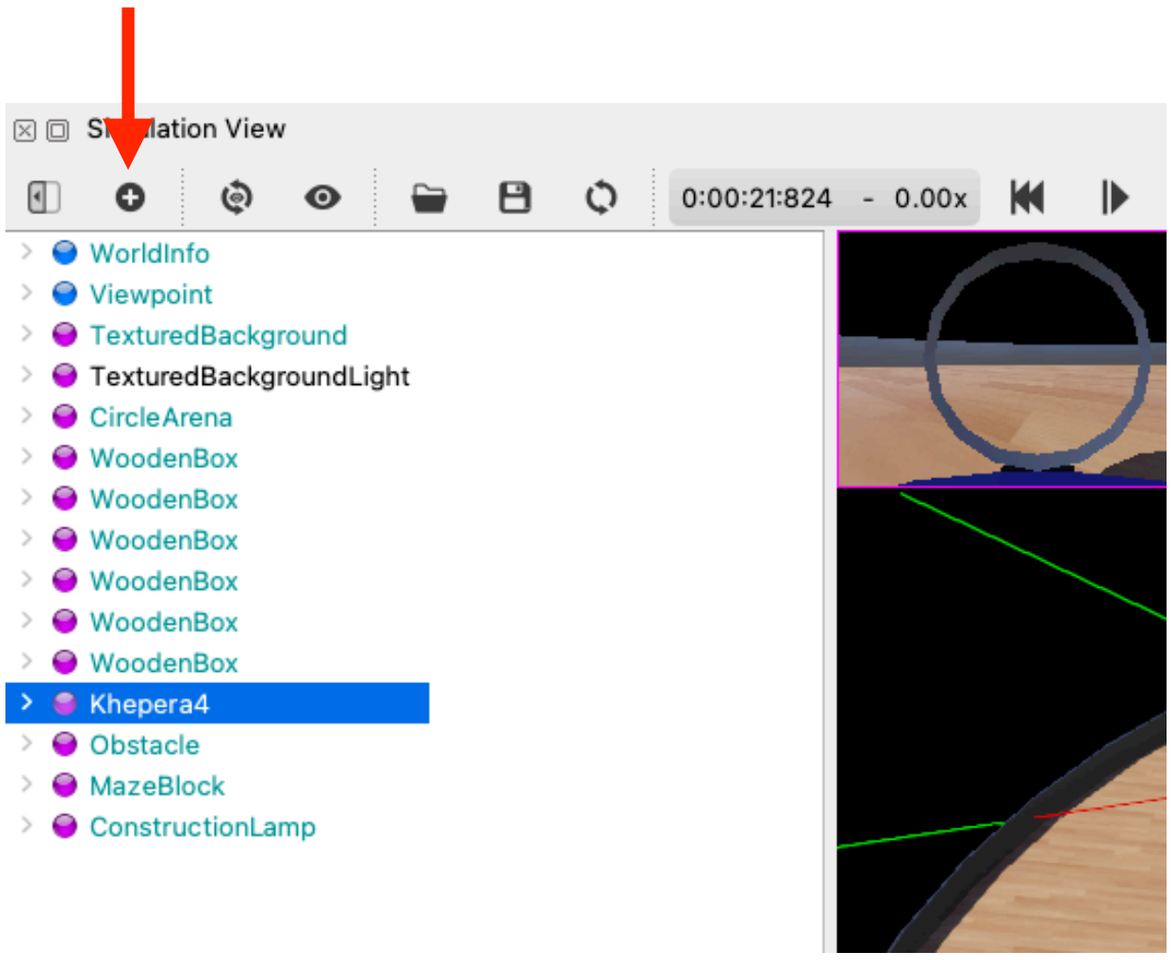


Objeto estático →

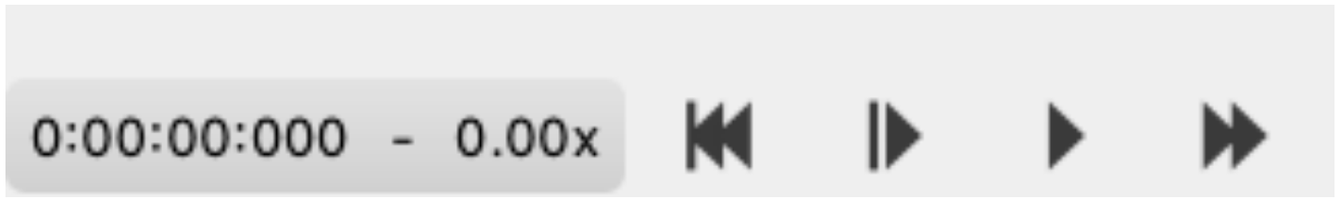
Robot →

Escenarios (worlds)

- Podemos añadir nuevos elementos a un escenario (objetos pasivos o robots).



Ejecución de la simulación



- Desde la barra de menús podemos **iniciar** y **pausar** la simulación (se ejecutará el controlador establecido para el robot). También podemos **ejecutarla paso a paso**.

- También podemos **resetear** la simulación. El escenario se restaurará al estado en el que lo guardamos, en su fichero .wbt).

- **¡Atención!** Si guardamos el escenario una vez iniciada una simulación modificaremos el estado inicial del mismo. Para evitarlo, debemos guardar antes de ejecutar la simulación o después de hacer un “reset”.

Programación de controladores

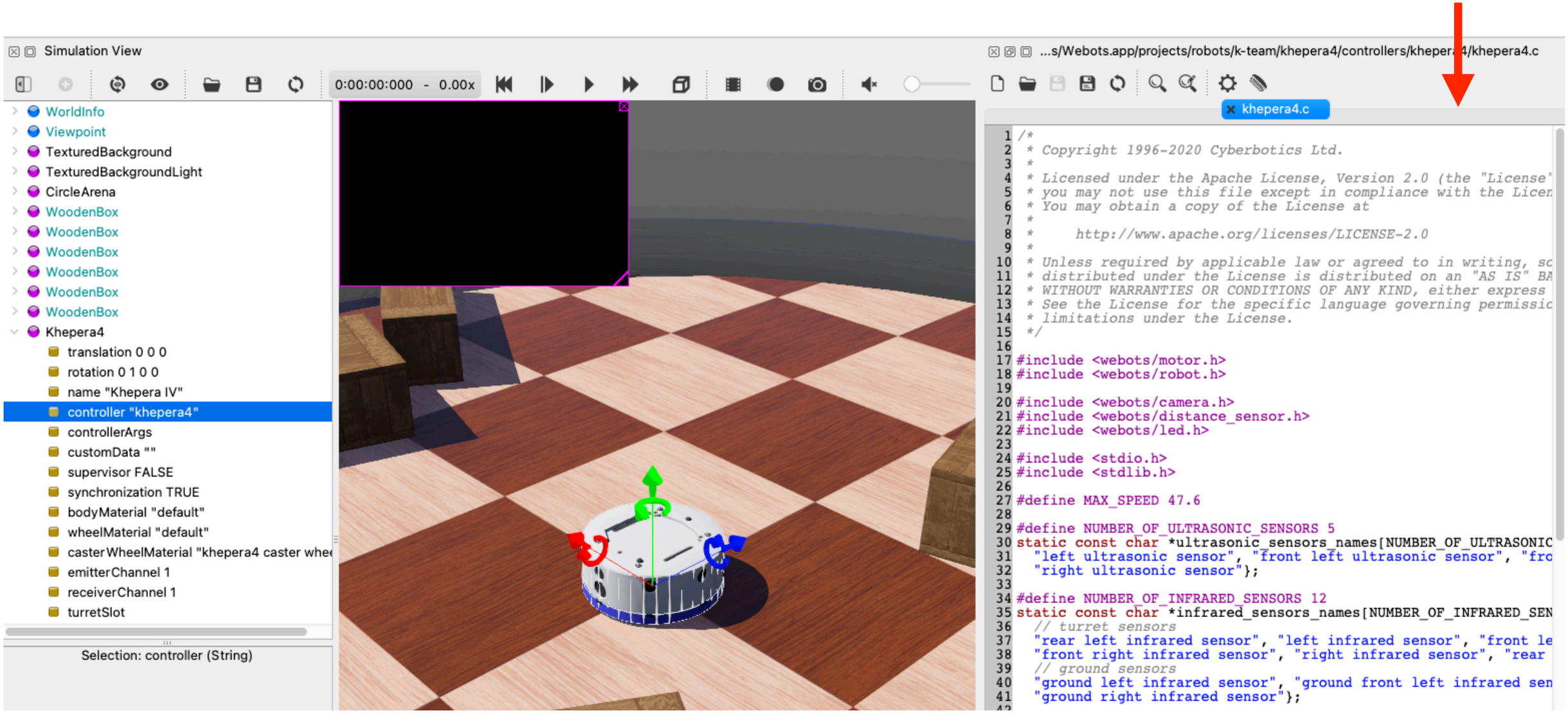
- Los controladores pueden programarse en diferentes lenguajes (C/C++, Python, Java, Matlab)
  - Para las prácticas utilizaremos **Python** o **C**.
  - Los ejemplos de la documentación y el API están documentados para todos los lenguajes soportados.

- Cuando se inicia una simulación se lanza un proceso para cada controlador asociado a un robot (incluso si varios robots comparten el mismo controlador).

- Documentación sobre programación en Webots para Windows/Linux/MacOS:
  - Uso de Python: https://cyberbotics.com/doc/guide/using-python
  - Uso de C: https://cyberbotics.com/doc/guide/using-c

Programación de controladores

- El entorno de Webots proporciona un editor de código fuente junto con opciones para crear, vincular y compilar controladores.



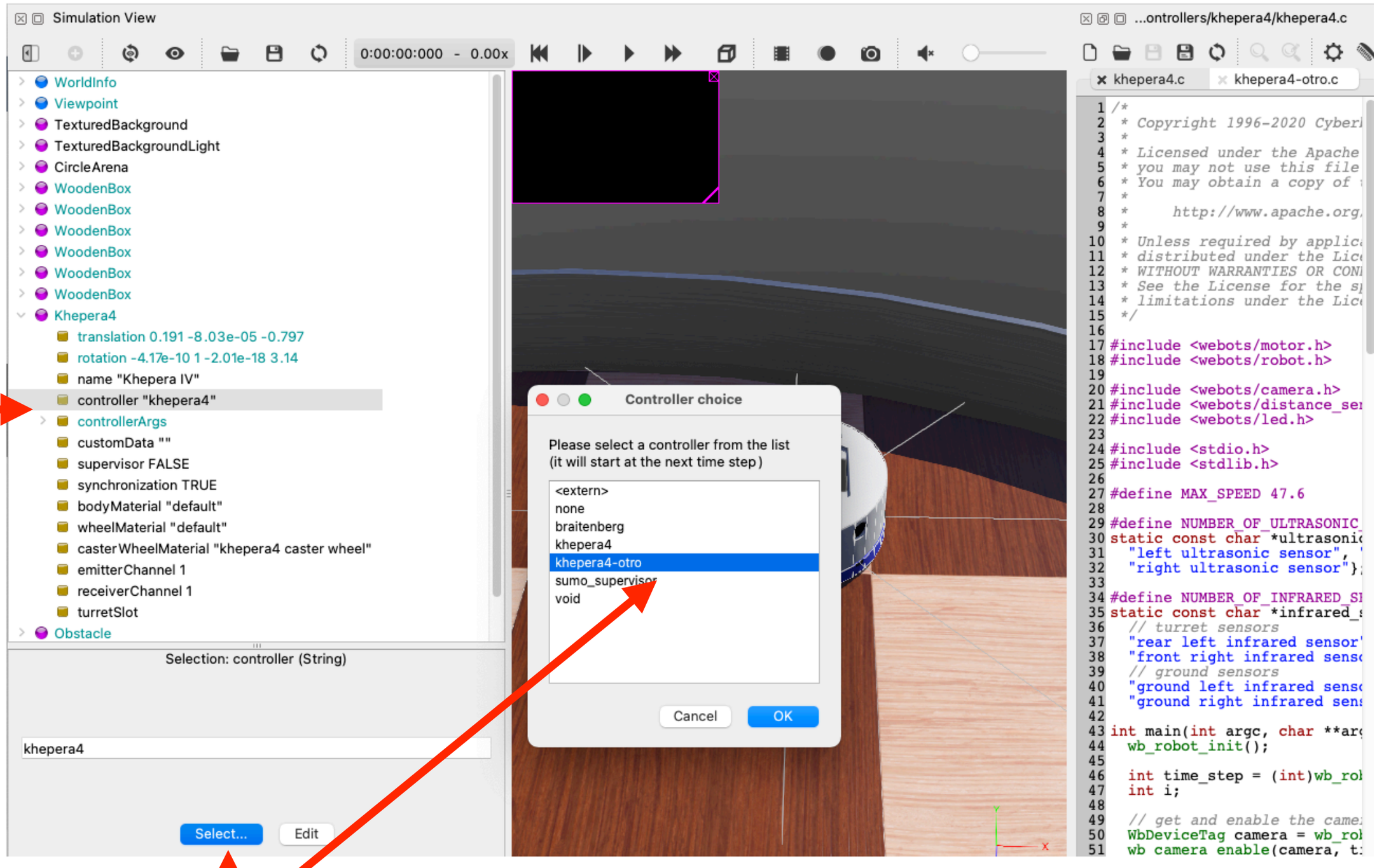
Compilar un controlador

- Si tenemos un controlador en C y no en un lenguaje interpretado, deberemos compilarlo antes de que pueda ser ejecutado.

- Los errores de compilación nos saldrán en la consola de Webots.

- Para cambiar a un nuevo controlador de un robot, primero debemos compilarlo (solo si estamos programando en C/C++).

Seleccionar otro controlador

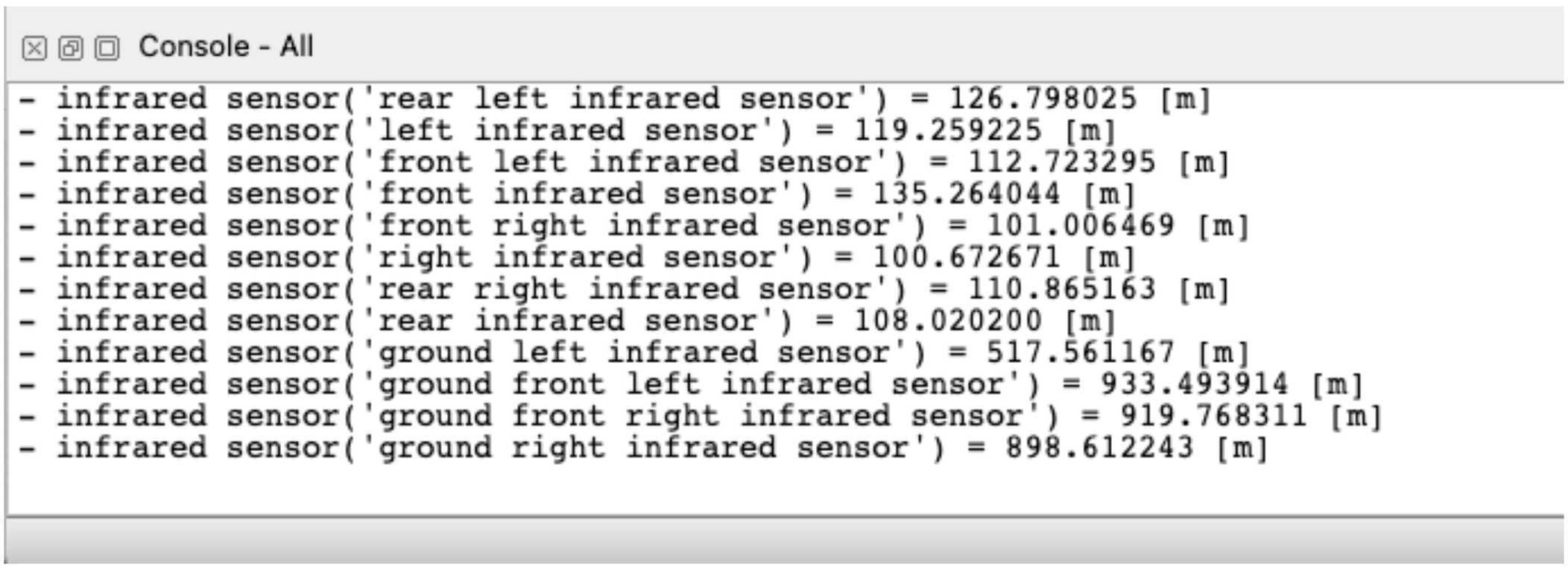


Controlador actual

Seleccionar otro fichero de controlador (C, Python, …)

Programación de controladores

- Podemos generar información de depuración en la consola de Webots utilizando printf() en el código del controlador.
  - **¡Atención!** Puede afectar al comportamiento si se escribe a una frecuencia alta.



Programación de controladores

```c
#include <webots/robot.h>

// Added a new include file
#include <webots/motor.h>

#define TIME_STEP 64

#define MAX_SPEED 6.28

int main(int argc, char **argv) {
  wb_robot_init();

  // get a handler to the motors and set target position to infinity (speed control)
  WbDeviceTag left_motor = wb_robot_get_device("left wheel motor");
  WbDeviceTag right_motor = wb_robot_get_device("right wheel motor");
  wb_motor_set_position(left_motor, INFINITY);
  wb_motor_set_position(right_motor, INFINITY);

  // set up the motor speeds at 10% of the MAX_SPEED.
  wb_motor_set_velocity(left_motor, 0.1 * MAX_SPEED);
  wb_motor_set_velocity(right_motor, 0.1 * MAX_SPEED);

  while (wb_robot_step(TIME_STEP) != -1) {
  }

  wb_robot_cleanup();

  return 0;
}
```

Librerías necesarias

Antes de cualquier otra llamada a la API

Bucle de control principal. Incluye llamada a *wb_robot_step()* para avanzar la simulación.

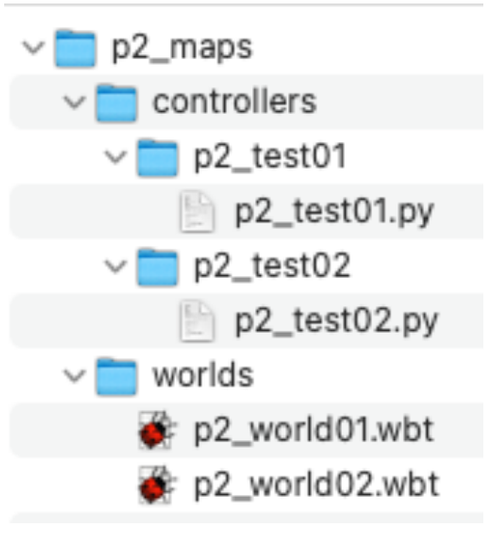
Termina comunicación entre controlador y simulador

Ejemplo básico, disponible a través del menú *"Open Sample World"*:

```
–> samples/tutorials/my_first_simulation.wbt
```

Programación de controladores

- Múltiples ejemplos de mundos y controladores básicos incluidos con Webots:
  - samples/tutorials/my_first_simulation.wbt
  - robots/k-team/khepera-4/khepera-4.wbt
  - …
- A través del menú *"Open Sample World"* para abrir un mundo de ejemplo y su controlador asociado.
- Podremos cambiar el controlador del robot/s por otro.
- Nuestros controladores y mundos deben guardarse con la siguiente estructura de directorios:



Tras abrir un fichero de mundo (.wbt) en Webots, podremos asignar al robot uno de los controladores que tengamos disponibles en una subcarpeta de la carpeta *controllers.*

Programación de controladores

- **¡Importante!** Documentación que se debe revisar para iniciarse en la programación de controladores:
  - https://cyberbotics.com/doc/guide/controller-programming
  - https://cyberbotics.com/doc/guide/tutorials

- **¡Importante!** Documentación de referencia de la API:
  - https://cyberbotics.com/doc/reference/nodes-and-api-functions
  - Ejemplo (funciones de Robot): https://cyberbotics.com/doc/reference/robot

Programación de controladores

- Función `wb_robot_step(time_step)`:
  - Tiene que estar presente en todos los controladores.
  - Esta función sincroniza los datos de los sensores, los actuadores y demás elementos del entorno entre controlador y simulador.
  - Debe ser llamada repetidamente para que la simulación avance paso a paso.
  - El parámetro `time_step` indica el tiempo de simulación que se avanza en la ejecución (en milisegundos) antes de devolver el control. Se refiere a tiempo de simulación no a tiempo real. Debe ser mayor o igual que el tiempo configurado para el mundo.
    - Los mundos de ejemplo de la práctica tienen configurado un step de 16ms. El time_step más bajo en nuestro controlador podrá ser de 16ms o 32ms por ejemplo, para una buena respuesta del robot en base a los datos del sensorización.
  - Repetidas lecturas/escrituras de sensores/actuadores darán el mismo resultado mientras la simulación no avance un nuevo paso (es decir, se avance time_step milisegundos de simulación).
  - Devuelve `-1` cuando Webots finaliza el controlador (se hace un reset a la simulación, se recarga el mundo, se termina webots, etc.)
  - https://cyberbotics.com/doc/guide/controller-programming?tab-language=c#the-step-and-wb_robot_step-functions

Lectura de sensores

Pasos a seguir (código en C):

```
WbDeviceTag sensor = wb_robot_get_device(“my_distance_sensor");
wb_distance_sensor_enable(sensor, TIME_STEP);
. . .
const double value = wb_distance_sensor_get_value(sensor);
```

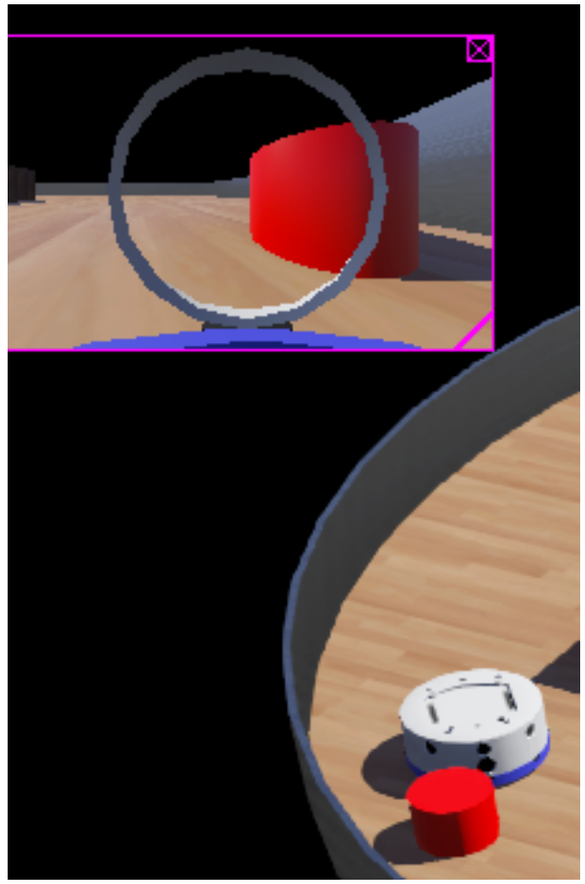
- Si, por ejemplo, se indica un TIME_STEP del doble del *controll step*, los datos del sensor se actualizarán cada dos llamadas a `wb_robot_step()`
- Una actualización muy rápida puede relanzar la simulación.
    - Se debe tener en cuenta el tipo de sensor para determinar una frecuencia de actualización razonable.
- La llamada a `wb_distance_sensor_get_value(sensor)` obtiene el valor del sensor actualizado en la última sincronización.

Actuadores

- Ejemplo: movimiento de los motores de las ruedas.
  - Movimiento por posición o por velocidad.
  - https://cyberbotics.com/doc/reference/motor#velocity-control

Cámara

- Introducción al uso de la **cámara**.
  - https://cyberbotics.com/doc/reference/camera
- ¡Atención! Procesar imágenes puede implicar una alta carga computacional…
  - Posible impacto negativo en el tiempo de respuesta de actuaciones críticas del robot.
  - Prestar atención a la frecuencia de lectura y procesamiento de los datos.
  - Intentar minimizar el tiempo de procesamiento con un código optimizado.

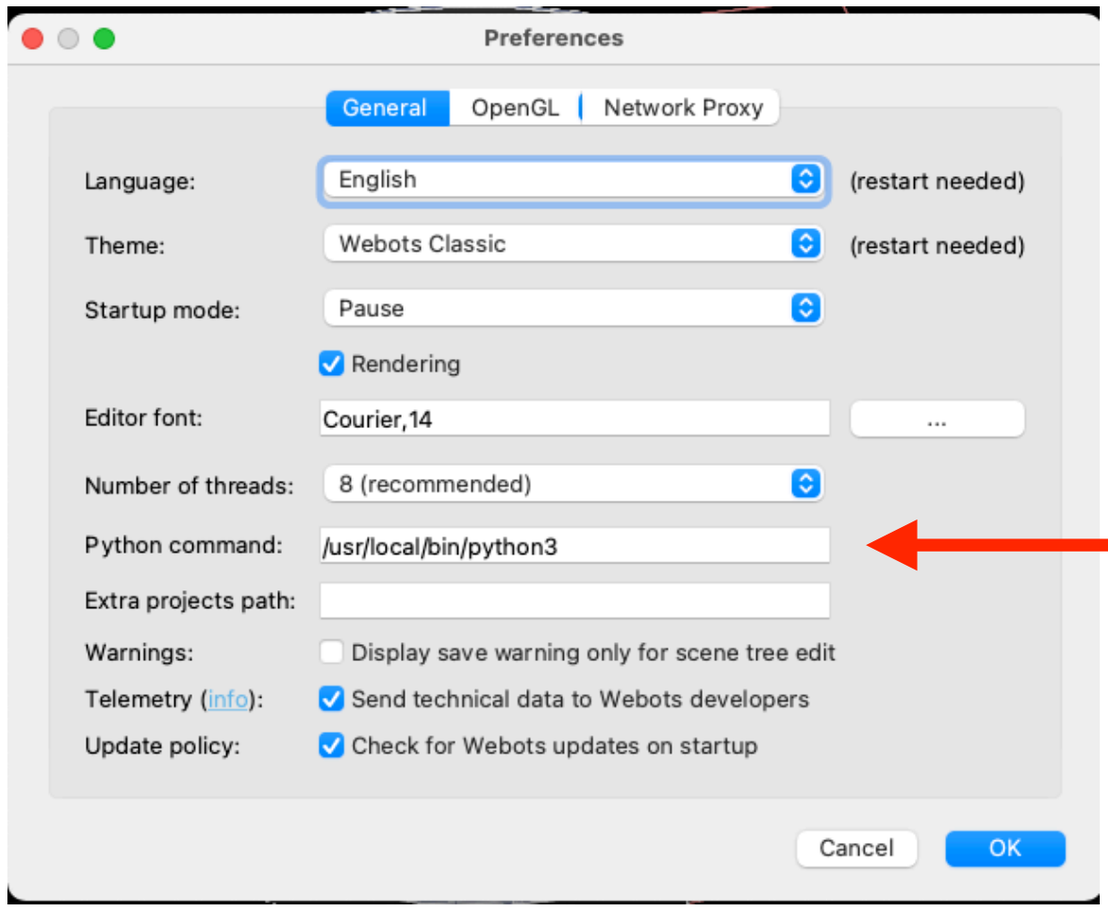


UNIVERSIDADE DA CORUÑA

Programación de controladores en Python

Controladores en Python

- Utilizar una de las versiones soportadas de Python 3 (ver documentación).
- Definir la configuración “Python command” según la instalación local de Python (disponible a través del menú de Preferencias de Webots).
- Crear un controlador en Python y asignarlo al robot.
  - Lenguaje interpretado => no es necesario compilar.
- https://cyberbotics.com/doc/guide/using-python



Código básico de un controlador

```
from controller import Robot, Motor, DistanceSensor

# Obtenemos el time_step por defecto o establecemos uno
#TIME_STEP = int(robot.getBasicTimeStep())
TIME_STEP = 32

# Máxima velocidad para Khepera4
# Podemos obtener el valor con una función getMaxVelocity() de la clase Motor
MAX_SPEED = 47.6

robot = Robot()

sensor = robot.getDevice("sensor_name")
sensor.enable(TIME_STEP)

while robot.step(TIME_STEP) != -1:
    value = sensor.getValue()
    print("Sensor value is: ", value)
```

Controladores externos

- Webots permite la programación de controladores que se programen y ejecuten como programas externos al simulador.
  - Podemos utilizar un IDE externo para programar y depurar (ej. Visual Studio Code o PyCharm para Python)
- **Documentación:** https://cyberbotics.com/doc/guide/running-extern-robot-controllers

UNIVERSIDADE DA CORUÑA

Monitorización gráfica de los sensores

Monitorización de sensores

1. Disponemos de una opción para visualizar gráficamente los valores de los distintos sensores del robot.

2. Pulsar con el botón derecho sobre el objeto y seleccionar la opción “Show Robot Window”

3. Seleccionar y activar los sensores correspondientes (acelerómetro, cámara, sensores de distancia, giróscopo, …).

Monitorización de sensores