**Webots Coding Tips**

The e-puck is a small robot having differential wheels, 10 [LEDs](https://cyberbotics.com/doc/reference/led), and several sensors including 8 [DistanceSensors](https://cyberbotics.com/doc/reference/distancesensor) and a [Camera](https://cyberbotics.com/doc/reference/camera).

1. **Controller Program**

A **controller** is a program that defines the behavior of a robot. The controller field of a Robot node specifies which controller is currently associated to the robot[[1]](#footnote-1), see Figure 1.

A screenshot of a computer

Description automatically generated

**Figure 1.**

**1.a How to save your webots world file and compile your controller**

When a Webots world is modified with the intention of being saved, it is fundamental that the simulation is first paused and reloaded to its initial state, i.e., the virtual time counter on the main toolbar should show 0:00:00:000, see Figure 2. Otherwise at each save, the position of each 3D object can accumulate errors. Therefore, any modification of the world should be performed in that order: **pause, reset, modify and save the simulation**

A screenshot of a computer

Description automatically generated

**Figure 2.**

You can compile your controller code by selecting the Build / Build menu item or clicking on the gear icon above the code area. Compilation errors are displayed in red in the console. If there are any, fix them and retry to compile. When Webots proposes to reset or reload the world, choose Reset and run the simulation. Don’t forget to save the modified source code (File / Save Text File or click on the”floppy disk” icon above the controller name, see Figure 2).

**1.b Headers of your controller**

For the robot configuration, your controller should include the header declaration:

#include <webots/robot.h>

To command the robot motors, include the motor header:

#include <webots/motor.h>

Likewise, a dedicated header should be used for any other used device in your project. For instance, “distances sensors” require the following:

#include <webots/distance\_sensor.h>

After the include statements, add a macro that defines the duration of each physics/simulation step. This macro will be used as argument to the wb\_robot\_step function, and it will also be used to enable the devices. This duration is specified in **milliseconds** and it must be a multiple of the value in the basicTimeStep field of the [WorldInfo](https://cyberbotics.com/doc/reference/worldinfo) node. The following duration is often used:

#define TIME\_STEP 64

Further defined symbol constants might be added as per the need of your controller. For instance, to control the speed of the robot motors, we often use the following which gives the maximum angular speed of the e-puck robot (2\*PI radian/second, i.e., one revolution per second):

#define MAX\_SPEED 6.28

**1.c Robot Devices**

In most applications, the controller processes the robot sensor readings to send the relevant commands to the robot motors (actuators). To access any device (sensor and actuator) of the robot, use

WbDeviceTag Tag = wb\_robot\_get\_device(deviceName);

Where *Tag* is the device variable name in your program and *deviceName* is the name of the device in the webots world file configuration.

A robot device is referenced by a WbDeviceTag. The WbDeviceTag *Tag* is retrieved by the wb\_robot\_get\_device function; the *Tag* is then used as first argument in every function call concerning this device.

To read a sensor value, use the right method from the built-in sensor API[[2]](#footnote-2) (see [LEDs](https://cyberbotics.com/doc/reference/led), [DistanceSensors](https://cyberbotics.com/doc/reference/distancesensor) , [Camera](https://cyberbotics.com/doc/reference/camera)). For instance, for [DistanceSensors](https://cyberbotics.com/doc/reference/distancesensor), you we can use the function:

**double** **wb\_distance\_sensor\_get\_value**(**WbDeviceTag** tag);

For motors, consult [motor.h](https://cyberbotics.com/doc/reference/motor) for the full list of the built-in functions. Two functions are particularly important:

* **Control per position:**

**void** **wb\_motor\_set\_position**(**WbDeviceTag** tag, **double** position);

The function specifies the desired/target position in radians from the current position

* **Control per velocity:**

This is obtained with two function calls: first the wb\_motor\_set\_position function must be called with INFINITY[[3]](#footnote-3) as a position parameter, then the desired velocity, which may be **positive or negative**, must be specified by calling the wb\_motor\_set\_velocity function:

**wb\_motor\_set\_position**(motor, INFINITY);

**wb\_motor\_set\_velocity**(motor, 6.28); // 1 rotation per second

**1.d Template main() code of the controller**

The following gives a typical structure of the controller program, pay attention to the comments:

// Header files, you may need to add more…

#include <webots/robot.h>

#include <webots/motor.h>

// you may need to add additional macro/symbolic constants here

#define TIME\_STEP 64

#define MAX\_SPEED 6.28

//The arguments of the main function can be specified by the \* "controllerArgs" field of the Robot node

int main(**int** argc, **char** \*\*argv) {

wb\_robot\_init(); // necessary to initialize robot devices

/\* declare here **WbDeviceTag** devicevariables/Tags for storing robot devices : \*/

WbDeviceTag my\_sensor = wb\_robot\_get\_device("s0");

WbDeviceTag left\_motor = wb\_robot\_get\_device("left wheel motor");

WbDeviceTag right\_motor = wb\_robot\_get\_device("right wheel motor");

// Set the target position of the motors, example control per velocity

**wb\_motor\_set\_position**(left\_motor, INFINITY);

**wb\_motor\_set\_position**(right\_motor, INFINITY);

**wb\_motor\_set\_velocity**(left\_motor, MAX\_SPEED);

**wb\_motor\_set\_velocity**(right\_motor, MAX\_SPEED);

// Perform simulation steps of TIME\_STEP milliseconds until receiving an exit event

**while** (wb\_robot\_step(TIME\_STEP) != -1){

// Read sensors outputs

// Process sensor reading/behaviour

// send commands to the actuators/motors

}

// The following is necessary to cleanup webots resources

**wb\_robot\_cleanup();**

return 0;

}

**1.e Examples**

The following programs can be used with your assigned webots worlds file available in Moodle

**Example 1:** One-off Control per position

Please note that in the webots e-puck model, the left/right motor is named left wheel motor / right wheel motor; these names are used in the below code.

#include <webots/robot.h>

#include <webots/motor.h>

#define TIME\_STEP 64

int main(int argc, char \*\*argv) {

wb\_robot\_init();

// get the motor devices

WbDeviceTag left\_motor = wb\_robot\_get\_device("left wheel motor");

WbDeviceTag right\_motor = wb\_robot\_get\_device("right wheel motor");

// set the target position of the motors

wb\_motor\_set\_position(left\_motor, 10.0);

wb\_motor\_set\_position(right\_motor, 10.0);

while (wb\_robot\_step(TIME\_STEP) != -1);

wb\_robot\_cleanup();

return 0;

}

On a successful execution, your robot should move forwards. The robot will move using its maximum speed for a while and then stop once the wheels have rotated of 10 radians. The while loop has an empty body (;), no sensor is read and processed.

### Example 2: Control per velocity

#include <webots/robot.h>

#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

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;

}

The robot will now move and never stop.

1. **Sensors Reading and Actuator Commands**

**2.a How to Steer a Differential Wheeled Robot**

An e-puck is a **differential wheeled robot,** i.e.a [mobile robot](https://en.wikipedia.org/wiki/Mobile_robot) whose movement is based on two separately driven [wheels](https://en.wikipedia.org/wiki/Wheel) placed on either side of the robot body. It can thus change its direction by varying the relative rate of rotation of its wheels and hence does not require an additional steering motion[1].

The diagram below illustrates the principle of operation of differential drive [2]:



**Figure 3.**

* When the two wheels are spinning in the same direction and at the same speed, the robot will move straight, either forward or backward depending on the direction of the wheel motion (sign of the speed).
* When the two wheels are spinning in the same direction, but with different speeds, the robot will **turn away from the faster motor**. For example, if **the right wheel is spinning faster** than the left, the motor will turn **left.**
* If the two wheels are spinning with the same speed, but in opposite directions, the robot will rotate in place, spinning around the midpoint between the two wheels.

**2.b Obstacle Avoidance using Proximity Sensors**

Figure 4 gives the top view of the e-puck model. The red lines represent the directions of the infrared distance sensors located around the turret of the robot. The string labels correspond to the distance sensor names.

A circular green circuit board with red arrows pointing to it

Description automatically generated

**Figure 4.** Top view of the e-puck model. The green arrow indicates the front of the robot.

In order to avoid obstacles, these 8 infra-red distance sensors should be read in order to actuate the two wheels. These nodes are referenced by their name fields (from ps0 to ps7).

The values returned by the distance sensors are scaled between 0 and 4096 (piecewise linearly to the distance). While 4096 means that a big amount of light is measured (an obstacle is close) and 0 means that no light is measured (no obstacle).

For a very simple collision avoidance behavior, as the robot moves forwards and detects an obstacle by the front distance sensors, the robot should be turned towards the obstacle-free direction.

The resulting program consists of the following parts.

// initialize devices

int i;

WbDeviceTag ps[8];

char ps\_names[8][4] = {"ps0", "ps1", "ps2", "ps3", "ps4", "ps5", "ps6", "ps7"

};

for (i = 0; i < 8; i++) {

ps[i] = wb\_robot\_get\_device(ps\_names[i]);

wb\_distance\_sensor\_enable(ps[i], TIME\_STEP);

}

A sensor such as the [DistanceSensor](https://cyberbotics.com/doc/reference/distancesensor) has to be enabled before use: wb\_distance\_sensor\_enable. The second argument of the enable function defines at which rate the sensor will be refreshed.

After initialisation of the devices, initialise the motors:

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);

wb\_motor\_set\_velocity(left\_motor, 0.0);

wb\_motor\_set\_velocity(right\_motor, 0.0);

In the main loop, read the distance sensor values as follows:

// read sensors outputs

double ps\_values[8];

for (i = 0; i < 8 ; i++)

ps\_values[i] = wb\_distance\_sensor\_get\_value(ps[i]);

In the main loop, detect if a collision occurs (i.e**., the value returned by a distance sensor is bigger than a threshold value often retrieved through experimentation**) as follows:

// detect obstacles

bool right\_obstacle =

ps\_values[0] > 80.0 ||

ps\_values[1] > 80.0 ||

ps\_values[2] > 80.0;

bool left\_obstacle =

ps\_values[5] > 80.0 ||

ps\_values[6] > 80.0 ||

ps\_values[7] > 80.0;

Finally, use the information about the obstacle to actuate the wheels as follows:

#define MAX\_SPEED 6.28

...

// initialize motor speeds at 50% of MAX\_SPEED.

double left\_speed = 0.5 \* MAX\_SPEED;

double right\_speed = 0.5 \* MAX\_SPEED;

// modify speeds according to obstacles

if (left\_obstacle) {

// turn right

left\_speed = 0.5 \* MAX\_SPEED;

right\_speed = 0.10 \* MAX\_SPEED;

}

else if (right\_obstacle) {

// turn left

left\_speed = 0.10 \* MAX\_SPEED;

right\_speed = 0.5 \* MAX\_SPEED;

}

// write actuators inputs

wb\_motor\_set\_velocity(left\_motor, left\_speed);

wb\_motor\_set\_velocity(right\_motor, right\_speed);

The complete controller.c code is given below, compile and run it.

#include <webots/robot.h>

#include <webots/distance\_sensor.h>

#include <webots/motor.h>

// time in [ms] of a simulation step

#define TIME\_STEP 64

#define MAX\_SPEED 6.28

// entry point of the controller

int main(int argc, char \*\*argv) {

// initialize the Webots API

wb\_robot\_init();

// internal variables

int i;

WbDeviceTag ps[8];

char ps\_names[8][4] = {

"ps0", "ps1", "ps2", "ps3",

"ps4", "ps5", "ps6", "ps7"

};

// initialize devices

for (i = 0; i < 8 ; i++) {

ps[i] = wb\_robot\_get\_device(ps\_names[i]);

wb\_distance\_sensor\_enable(ps[i], TIME\_STEP);

}

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);

wb\_motor\_set\_velocity(left\_motor, 0.0);

wb\_motor\_set\_velocity(right\_motor, 0.0);

// feedback loop: step simulation until an exit event is received

while (wb\_robot\_step(TIME\_STEP) != -1) {

// read sensors outputs

double ps\_values[8];

for (i = 0; i < 8 ; i++)

ps\_values[i] = wb\_distance\_sensor\_get\_value(ps[i]);

// detect obstacles

bool right\_obstacle =

ps\_values[0] > 80.0 ||

ps\_values[1] > 80.0 ||

ps\_values[2] > 80.0;

bool left\_obstacle =

ps\_values[5] > 80.0 ||

ps\_values[6] > 80.0 ||

ps\_values[7] > 80.0;

// initialize motor speeds at 50% of MAX\_SPEED.

double left\_speed = 0.5 \* MAX\_SPEED;

double right\_speed = 0.5 \* MAX\_SPEED;

// modify speeds according to obstacles

if (left\_obstacle) {

// turn right

left\_speed = 0.5 \* MAX\_SPEED;

right\_speed = 0.10 \* MAX\_SPEED;

}

else if (right\_obstacle) {

// turn left

left\_speed = 0.10 \* MAX\_SPEED;

right\_speed = 0.5 \* MAX\_SPEED;

}

// write actuators inputs

wb\_motor\_set\_velocity(left\_motor, left\_speed);

wb\_motor\_set\_velocity(right\_motor, right\_speed);

}

// cleanup the Webots API

wb\_robot\_cleanup();

return 0; //EXIT\_SUCCESS

}

**2.c Lane Follower Principle**

Assume that a mobile robot has to drive itself forward, following a black line (or lane!) on a white surface, see Figure 5. The robot has two infrared sensors, one at each side of its front, pointing downwards to the surface. The output of the sensor is a high voltage when it is above the black line, and a low voltage when it is over the white surface.

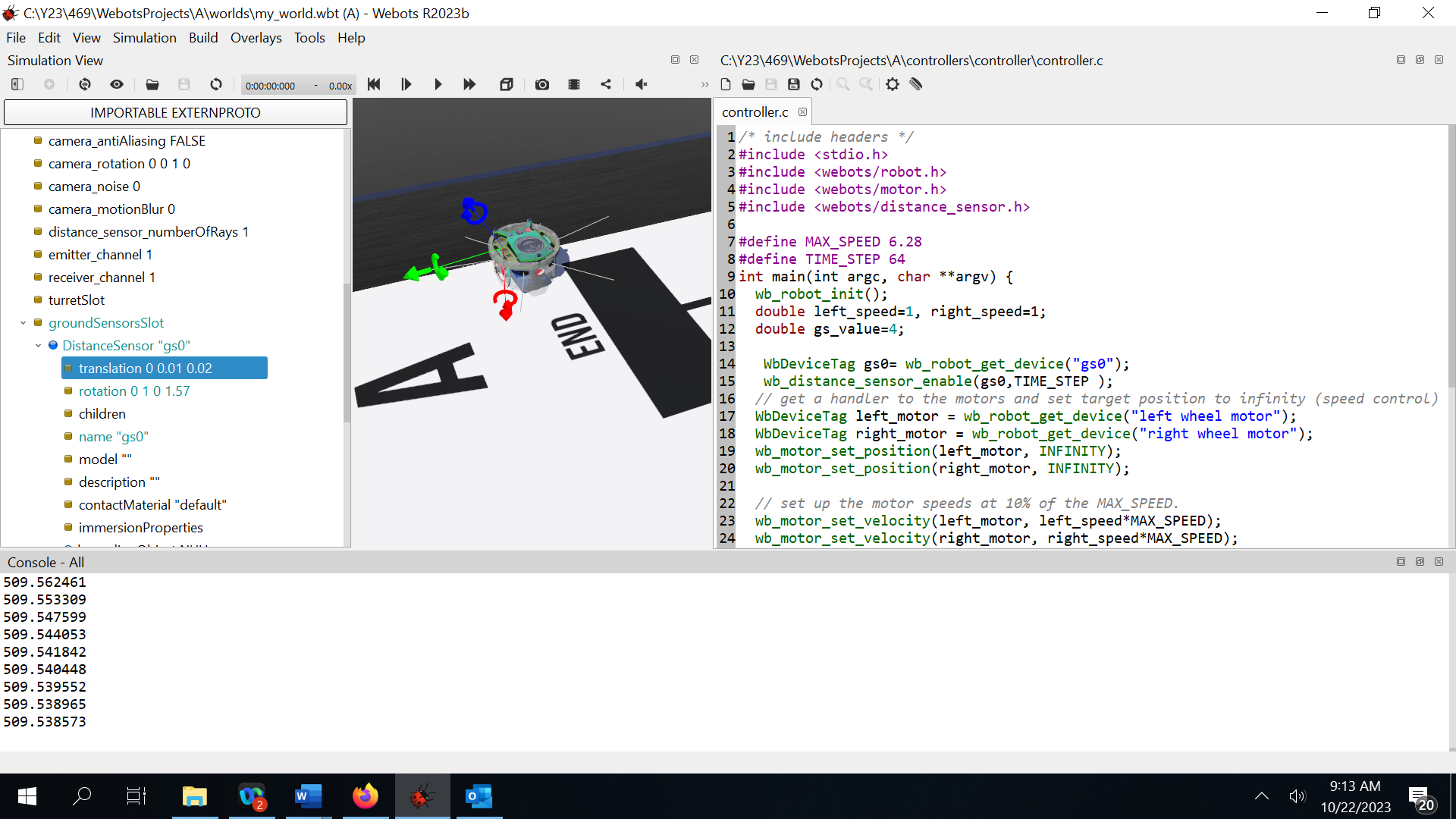
**The robot should steer gently to the left(right) when it starts drifting to the right(left)[[4]](#footnote-4).** This can be detected when one of two sensors returns high and the other returns low. The robot should steer sharply to its left when in state 2 of Figure 5, and to its right when in state 3 instead. Otherwise, it keeps on moving straight. At the start, the robot is positioned ideally to follow the line with both sensors returning low values.



**Figure 5**

**Example:**

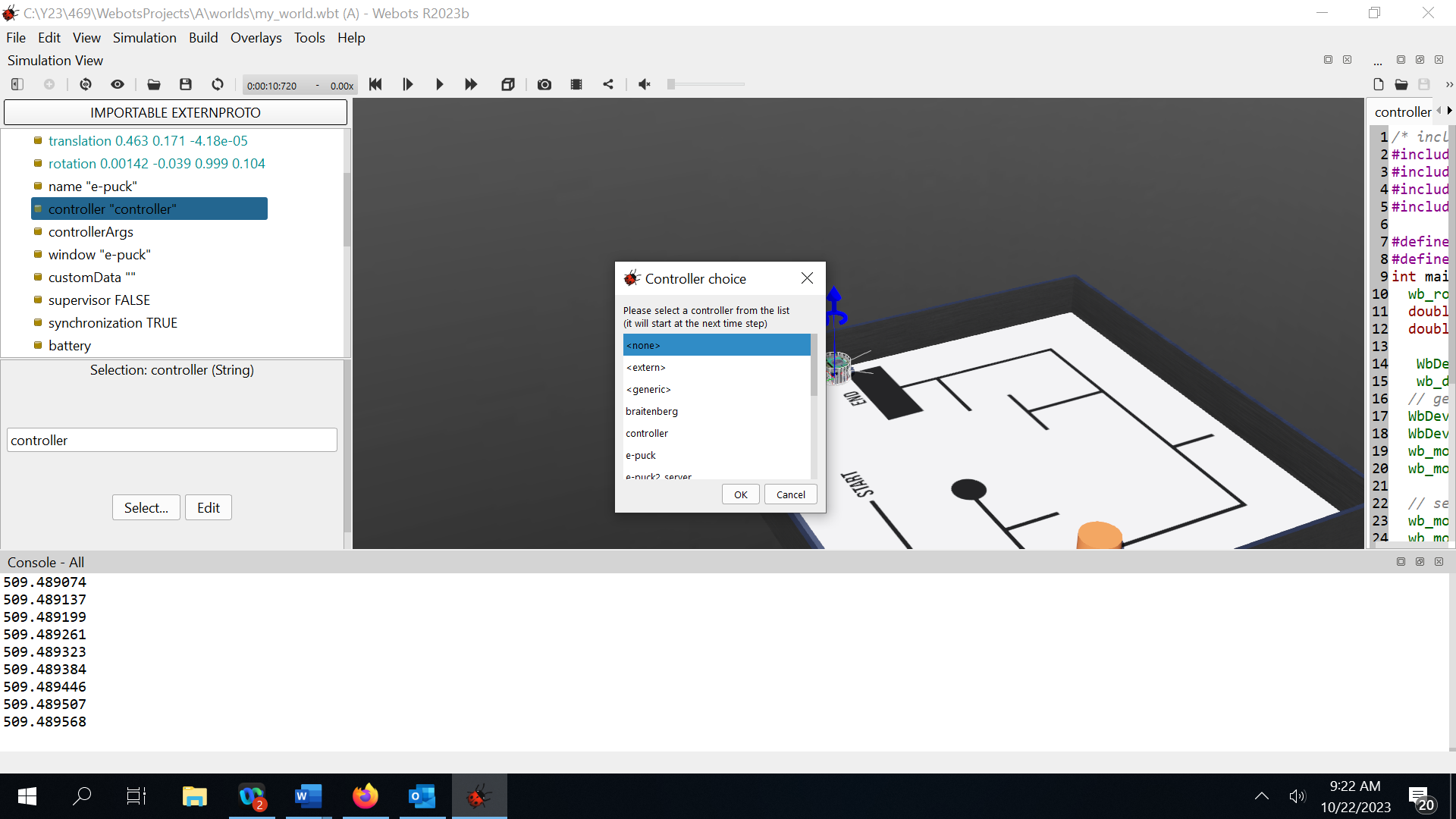
See the attached webots project where the ground sensor is named gs0 and located at z=0.02 (click on translation to alter z, **you might have to place all your IR sensors at this height in your implementation to the project**). The sensor reading gives the value 1000 on a black surface and ~500 on a white surface



**Figure 6**

**Miscellaneous**

1. To set the starting position of your robot, press the key “Shift” and click on the robot to move it to your selected starting position, save then the world as shown in Figure 2.
2. Webots comes with a number of controller examples you can study and learn from. They can be selected by clicking first on the “controller” field under “e-puck node” under the scene tree, then click on “Select” from the below section as shown in Figure 7



**Figure 7**

**References**

[1] <https://en.wikipedia.org/wiki/Differential_wheeled_robot>

[2] <https://42bots.com/tutorials/differential-steering-with-continuous-rotation-servos-and-arduino/>

[3] <https://cyberbotics.com/doc/guide/tutorial-1-your-first-simulation-in-webots>

[4] <https://cyberbotics.com/doc/guide/tutorial-4-more-about-controllers>

1. Expand the robot node in the “scene tree” in your webots project and select “controller”. The “Select” button under the “scene tree” allows you to specify the controller program. The latter can also be edited by right clicking on the robot and selecting “edit controller” [↑](#footnote-ref-1)
2. The **controller API** is the programming interface that gives you access to the simulated sensors and actuators of the robot. For example, including the webots/distance\_sensor.h file allows to use the wb\_distance\_sensor\_\* functions and with these functions you can query the values of the [DistanceSensor](https://cyberbotics.com/doc/reference/distancesensor) nodes. The documentation on the API functions can be found in [Reference Manual](https://cyberbotics.com/doc/reference/nodes-and-api-functions) together with the description of each node. [↑](#footnote-ref-2)
3. INFINITY is a C macro corresponding to the IEEE 754 floating point standard. [↑](#footnote-ref-3)
4. This is done by controlling the speeds of the individual motors, see section 2.b [↑](#footnote-ref-4)