Lab 6: Introduction to Webots and Sensor Modeling

This laboratory requires the following software:

- Webots simulator
- C development tools (gcc, make, etc.)

The laboratory duration is approximately 3 hours. Although this laboratory is not graded, we encourage you to take your own personal notes as the final exam and the course project might leverage results acquired during this laboratory session. For any questions, please contact us at **sis-ta@groupes.epfl.ch**

1.1 Information

In the following text you will find several exercises and questions.

- The notation **S** means that the question can be solved using only additional simulation.
- The notation **Q** means that the question can be answered theoretically, without any simulation.
- The notation **I** means that the problem has to be solved by implementing a piece of code and performing a simulation.
- The notation **B** means that the question is optional and should be answered if you have enough time at your disposal.

1.2 Getting Started (Short reminder)

To start with this lab, you will first need to install Webots and follow the Webots mini tutorials. Download *lab06.zip* available on moodle in your personal directory. Now, extract the lab archive.

2 Introduction

2.1 The e-puck

The e-puck is a miniature mobile robot developed to perform "desktop" experiments for educational purposes.
Figure 1 shows a close-up of the e-puck robot. More information about the e-puck project is available at http://www.e-puck.org/.

The e-puck's most distinguishing characteristic is its small size (7 cm in diameter). Other basic features are: significant processing power (dsPIC 30F6014 from Microchip running at 30 MHz), programmable using the standard gcc compilation tools, energetic autonomy of about 3-4 hours of intensive use (with no additional modules), an extension bus, a belt of eight light and proximity sensors, a 3-axis accelerometer, three microphones, a speaker, a color camera with a resolution of 640x480 pixels, 8 red LEDs placed around the robot and a Bluetooth interface to communicate with a host computer. The wheels are controlled by two miniature stepper motors, and can rotate in both directions.

Figure 1: Close-up of an e-puck robot.

The simple geometrical shape along with the positioning of the motors allows the e-puck to negotiate any kind of obstacle or corner.

Modularity is another characteristic of the e-puck robot. Each robot can be extended by adding a variety of modules.

2.2 Webots

Webots is a fast prototyping and simulation software developed by Cyberbotics Ltd., a spin-off company of EPFL. Webots allows the experimenter to design, program, and simulate virtual robots which act and sense in a 3D environment.

Webots can either simulate the physics of the world and the robots (nonlinear friction, slipping, mass distribution, etc.) or simply kinematic laws. The choice of the level of simulation is a trade-off between simulation speed and simulation realism. However, all sensors and actuators are affected by a realistic amount of noise so that the transfer from simulation to the real robot is usually quite smooth.

Many types of robots can be simulated with Webots, including wheeled, legged, flying, and swimming robots. Some interesting examples can be found in the *Webots Guided Tour* (menu: *Help->Webots Guided Tour).*

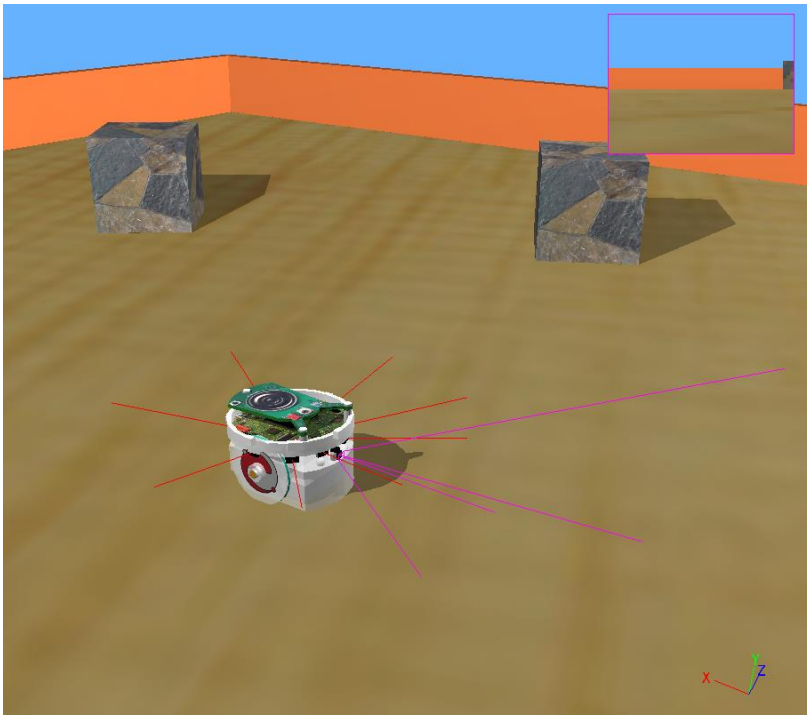

Figure 2: Webots simulation of the e-puck robot.

A simulated model of the e-puck robot is provided with Webots (see Figure 2). The current version of the e-puck model simulates the differential-drive system with physics (including friction, collision detection, etc.), the proximity sensors, the leds, the accelerometer, the light sensors, and the camera. In the future, this model will be further refined to simulate more of the e-puck's functionality. During this laboratory we will perform experiments exclusively with simulated e-puck models.

There are two Webots help sources available online as a PDF and HTML: Webots User Guide and Webots Reference Manual. You can access them under *Documentation* at the following link: http://www.cyberbotics.com/ or directly from the *Help* menu of Webots.

3 Webots mini-tutorial

3.1 Starting Webots

1. Launch Webots. On Windows or MacOS you can launch Webots as any other software. On Ubuntu, you need to enter the following command in a terminal:
   ```
   webots --mode=pause &
   ```
2. From the menu, select *File->Open World*, and choose the *e-puck.wbt* file from the *lab_06/worlds* directory structure that was just created.
3. At this point the e-puck model should appear in the Webots main window.
4. By clicking on the arrow next to 'DEF E_PUCK Robot' in the Scene Tree, positioning yourself on the *controller* field and clicking Edit, the source code of the controller that the e-puck runs will be displayed in the Text Editor. See Figure 3 for more information.

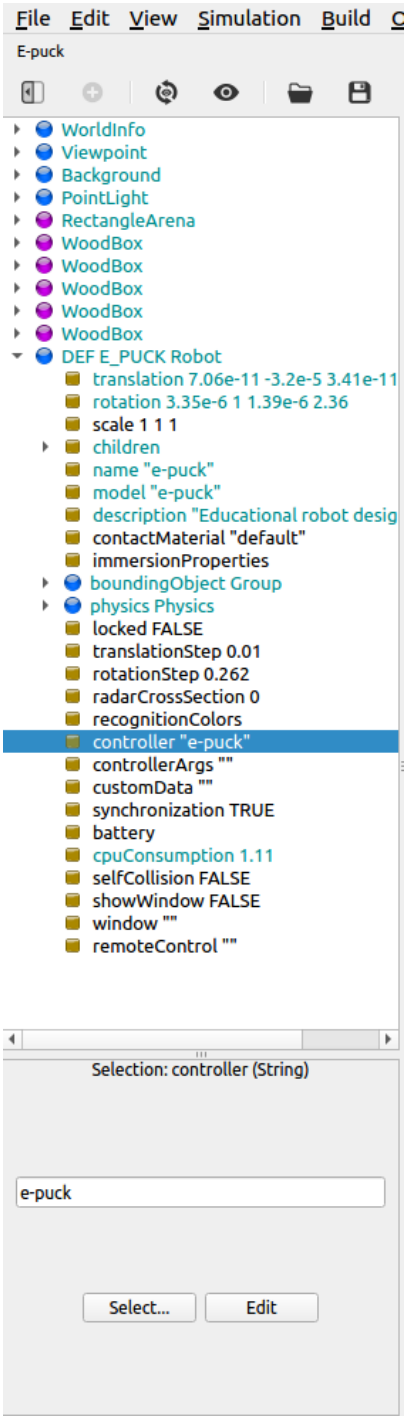

Figure 3- Locating the robot controller in e-puck scene tree.

5. Now you can build the project by clicking on the *Build* button: 
6. Now you can hit the *Run* button and the simulation will start. Note that the robot is not supposed to move at this point.

You can *Stop*, *Run* and *Revert* the simulation with the corresponding buttons in the Webots toolbar. Please try pressing all these buttons to see what they do:

- *Revert*: Reloads the world (*.wbt* file) and restarts the simulation from the beginning
- *Step*: Executes one simulation step
- *Pause*: Stops at the current simulation step
- *Run real-time*: Runs the simulation in real-time.
- *Run*: Runs the simulation
- *Run fast*: Runs the simulation at the maximal CPU speed (only visual rendering is disabled for better performance)

Next to these buttons you will see two numerical indicators (see Figure 4): The left indicator (a clock like *0:02:48:320*) shows the simulation elapsed time as *Hours:Minutes:Seconds:Milliseconds*. Note that this is simulated time (rather than the wall-clock time) emulating faithfully the potential real time progress that would be expected if the experiment was carried out in reality. It stops when the simulation is stopped.

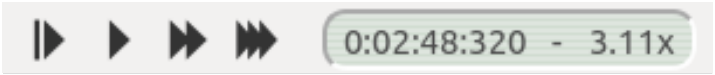

Figure 4: Elapsed time indicator and speedometer.

The right indicator (*3.11x*) is the speedometer which indicates how fast the simulation is currently running with respect to real time (wall-clock time). See how the elapsed time and speedometer are affected by the *Run* and *Fast* buttons.

3.2 Manipulating objects

Learn how to navigate in the 3D view using the mouse: try to drag the mouse while pressing all possible mouse buttons (and the wheel) combinations.

Various objects can be manipulated with the mouse: this allows you to change the initial configuration, or to interact with a running simulation. In order to move an object:

- Select the object and drag the green, red and blue arrows to move and rotate the object in different axis.
- Select the object, hold the *Shift* key and drag the mouse while pressing the left mouse button to shift an object in the xz-plane (parallel to the ground).
- Select the object, hold the *Shift* key and drag the mouse up and down, while pressing simultaneously the left and right mouse buttons (or the middle button), to lift and lower the object (alternatively you can also use the mouse wheel).
- To apply a force to an object, place the mouse pointer where the force will apply (without the object being selected), hold down the Alt and Control keys and left mouse button together while dragging the mouse.

- To apply a torque to an object, place the mouse pointer on it (without the object being selected), hold down the Alt and Control keys and right mouse button together while dragging the mouse.
- Applying force and torque works only for objects with mass. In the case of the world where you are working now, only the e-puck has mass (not the boxes).

Now, if you want, you may try all of the above manipulations with the e-puck and the obstacles both while the simulation is stopped or running. The mini-tutorial is finished.

4 Simulating sensors

4.1 Proximity sensors

A small *robot-window* for the e-puck is built in Webots. You can open the robot-window by double-clicking on the e-puck in the world view (or right-click on the robot and choose *Show Robot Window*). This window visualizes in real time the values of each of the sensors of the robot, classified by tabs: accelerometer, camera, differential wheel counters, distance sensors, and light sensors. To view the simulated sensor rays in the 3D scene, select View->Optional Rendering ->Show Distance Sensor Rays from the menu. When the simulation is running, you should see a sensor ray originating from the e-puck for each proximity sensor.

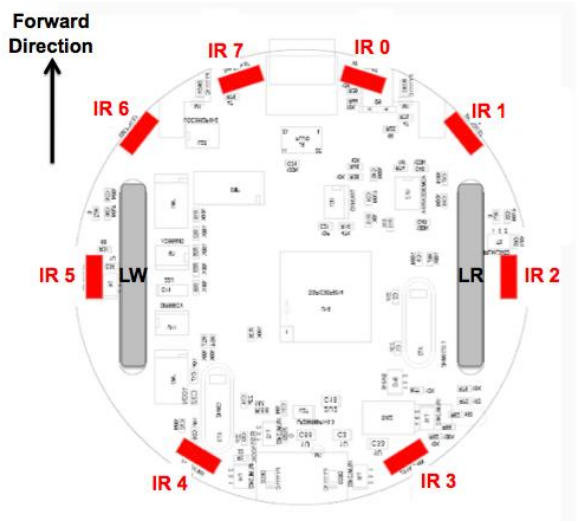

Figure 5: e-puck proximity sensors.

1. **(S):** Start the simulation by clicking the "run" button. Examine the control window (double click on the e-puck). Are the sensor measurements changing all the time (see the *DistanceSensor* tab for example)? Why? Is it the same on a real system?

2. **(S):** By moving objects around the e-puck, draw the response of the proximity sensor as a function of the distance to an obstacle. Note that the range of the e-puck's proximity sensors is fairly short. The step-by-step mode is also more convenient in order to study what is happening.

3. **(S):** Move your robot with your mouse and place it in a way, such that it is overlapping with an obstacle. What happens if an obstacle and the robot are interpenetrating? What are the proximity sensor measurements in this case (Hint: you can use F11 and F12 in order to switch between wireframe and normal mode in order to see what is happening inside bodies, the step-by-step mode is also more convenient in order to study what is happening)?

4.2 Accelerometer

4. **(S):** Another useful sensor of the e-puck robot which can be simulated in Webots is the 3-axis accelerometer. Load and run the world e-puck_slope.wbt. Open the robot control window and go to the accelerometer tab. In the world, select the big box on which the robot is standing. As you saw in the mini-tutorial, rotate the big box by dragging the blue and green circular arrows on the object. Examine the values in the control window. Identify which color represents each of the axis of the accelerometer.

4.3 Light sensors and the supervisor

The e-puck robot can be used to detect sources of light. In the following exercise, you will use your simulated e-puck to conduct a virtual experiment, collect data, and finally display them in MATLAB.

5. **(S):** Load the world e-puck_light_sensors.wbt. Do not forget to compile the controller of the e-puck (called e-puck_ls).
6. **(S):** Try to run the simulation. What message do you get on the console? Exceptionally in this world you will have to compile another controller, which is called the supervisor controller (in the Scene Tree click on the light_supervisor, then on controller -> Edit and then on the Build button). The supervisor controller is a special controller that is not used to control the robot but to influence the environment (e.g., modify the world's lighting conditions) or log information about the robot's and environment's state. Open the supervisor controller. Try to understand what the code is doing. Now, compile and build this code and run the simulation.

   When you begin the simulation, you should see two pulsing light sources of different colors (if you cannot see the lights, make sure the following option is checked: *View -> Optional Rendering -> Show Light Positions)*. Move the e-puck around the scene and observe the light sensor readings in the robot-window.
7. **(S):** The light sensor readings are being written to the file *controllers/e-puck_ls/ls_values.txt* (the absolute path is given in the Webots console). Leave the simulation running for some time in order to gather enough data, then simply stop the simulation without reloading the world. Open MATLAB, set the current directory to *lab_06*, then run the *ReconstructionWebots.m* file. As provided, this script loads the signal sampled in the Webots simulation and performs a linear interpolation ("first order hold algorithm").
8. **(Q):** Take a look at the FFT of the interpolated signal. How many frequency components do you observe?

5 Simulating mobile robots

Now we will do some experiments with moving robots. In particular, you will implement basic robot controllers that react to the environment in order to avoid obstacles.

5.1 Obstacle avoidance

Open the world *e-puck_obstacle.wbt* and look at the simulation environment. You can see the e-puck robot placed in the arena with several obstacles and walls scattered around. In this exercise the e-puck robot has to move in the arena while avoiding those obstacles. The controller that implements this behavior is written in *e-puck_obstacle.c* file.

9. **(S):** Run the simulation without changing the controller in *e-puck_obstacle.c* (do not forget to compile the robot controllers). How is the e-puck robot behaving? Is the robot avoiding the obstacles? Go to *e-puck_obstacle.c* file and identify the lines of code responsible for setting the wheel speeds of the robot.

10. **(I):** In that part of the code, implement a controller for the wheel speeds of the robot, based on a set of simple rules (if -> then -> else) on the values of the proximity sensors, so that the robot can avoid obstacles. Note that the values of the proximity sensors are stored in the vector `ds_values`, as seen in the lines of code 54-55 of *e-puck_obstacle.c*. Recall Section 4.1, where you saw how the values of these sensors changed when an obstacle came close.

5.2 Braitenberg vehicles

Valentino Braitenberg presented in his book “Vehicles: Experiments in Synthetic Psychology” (The MIT Press, 1984) several interesting ideas for developing simple, reactive control architectures and obtaining several different behaviors. In this lab, we want to see what we can achieve if e-pucks are programmed as Braitenberg vehicles. Figure 6 shows four different Braitenberg vehicles.

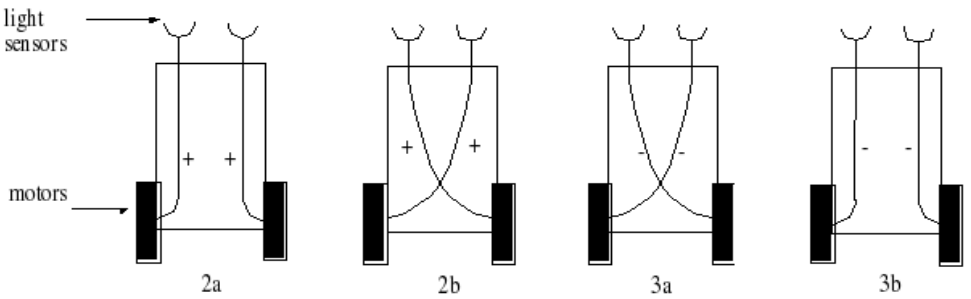

Figure 6: Four Braitenberg vehicles. Plus signs correspond to excitatory connections, minus signs to inhibitory ones. Vehicle 2a avoids light by accelerating away from it. Vehicle 2b exhibits a light approaching and following behavior, accelerating more and more as approaches

the source. Vehicle 3a avoids light by braking and accelerating away toward darker areas. Finally, vehicle 3b approaches light but brakes and stops in front of it.

11. **(Q):** Which vehicle depicted in Figure 6 do you expect to be most effective at avoiding obstacles if light sensors were to be replaced by proximity sensors, and why?

In the next questions, we are going to tune the parameters of a Braitenberg controller on proximity sensors (instead of light sensors) and exclusively using a linear perception-to-action map (i.e., essentially an 8x2 coefficient matrix, because we have 8 inputs – the proximity sensors - and 2 outputs – the motors of each wheel).

The principle of a Braitenberg controller is to directly compute the wheel speeds from the sensor values using a simple linear combination of parameters and sensor values:

$$\text{speed}_{\text{left}} = \sum_{i=0}^{n} \alpha_{\text{left,i}} \left(1 - \frac{\text{ps\_value}_{\text{i}}}{\text{ps\_range}}\right)$$

$$\text{speed}_{\text{right}} = \sum_{i=0}^{n} \alpha_{\text{right,i}} \left(1 - \frac{\text{ps\_value}_{\text{i}}}{\text{ps\_range}}\right)$$

where $\text{ps\_value}_{\text{i}}$ is the value of the i-th proximity sensor, ps_range is the acceptable range of sensor values, and $\alpha_{\text{left,i}}$ and $\alpha_{\text{right,i}}$ are two design parameters.

12. **(Q):** Open the world *e-puck_braitenberg.wbt*, and look at the controller *e-puck_braitenberg.c*. Observe the structure of the controller and identify the part of the code to be completed or modified. How does the code implement the Braitenberg controller described above?

13. **(Q):** What is the influence of the parameter, say, $\alpha_{\text{left,0}}$ on the speed of the left wheel? *Hint: explain what happens if $\alpha_{left,0}$ is large or small when an obstacle is detected by the proximity sensor* `ps0` *(recall Figure 5).*

14. **(I):** Modify the parameters of the controller *e-puck_braitenberg.c* so that you achieve these behaviors (see note in code and refer to Fig. 5 for sensor placement):

1. The robot is moving forward while smoothly avoiding obstacles.
2. The robot is moving forward while being attracted by obstacles.

Remember to "Build" your controller after you make a change: 

Hint: to tune Braitenberg parameters, start with all coefficients at zero. Use only the two proximity sensors at the front, before completing the solution using all proximity sensors.

5.3 The role of noise in robotics simulations

It is possible to configure the noise on sensor readings and motor outputs in the Webots simulator in order to model what happens in the real world. Real-world noise can cause poor performance on many algorithms which perform very well theoretically. However, if treated correctly, noise can also be used to obtain positive effects in some systems. In some cases, noise can even become an important ingredient of the algorithm. Hereafter, you will investigate the role of noise for escaping deadlocks in an obstacle avoidance situation.

15. **(S):** Open and run the *e-puck_no_noise.wbt* world; in this example the noise of all the proximity sensors is set to 0 and the robot is avoiding obstacles using a Braitenberg controller implemented in the *e-puck_braitenberg_solution.c* file. The objective of this exercise is for the robot to avoid the V-shaped obstacle without remaining stuck. Run this simulation **several times** (without changing the controller in the *e-puck_braitenberg_solution.c* file). Report the number of runs and the success rate of the robot.

16. **(Q):** What happens? Why?

17. **(S):** Now, open and run the *e-puck_medium_noise.wbt* world, which is exactly the same as *e-puck_no_noise.wbt*, except for the noise of the proximity sensors of the e-puck. Run this simulation as many times as in the previous experiment without noise. Report the number of runs and the success rate of the robot. Discuss briefly the differences with the previous experiment without noise.

18. **(Q):** What is the noise on the proximity sensors of the e-puck in the *e-puck_medium_noise.wbt* world *(Hint: try to expand the model of the e-puck in the scene tree and look for the lookup table of the proximity sensors, E_PUCK -> children -> EPUCK_PS0 -> children -> DistanceSensor -> lookupTable)?*

    Consult the following Reference Manual page to find out how to interpret the proximity sensor lookup table:
    https://www.cyberbotics.com/doc/reference/distancesensor

19. **(S):** Now, open and run the *e-puck_huge_noise.wbt* world. Run this simulation as many times as the previous experiment without noise. Report the number of runs and the success rate of the robot.

20. **(Q):** What happens now? What does it tell us about noise in robotics experiments?