Technical University of Crete
School of Electrical and Computer Engineering

INF 412 – Autonomous Agents – 2023

Instructor: M. G. Lagoudakis **2nd Laboratory Exercise** Deadline: 26.11.2023, 23:59

Introduction
Having already had a first experience with the robotic simulator Webots, you are now able to design simple controllers (controllers) for different robotic systems, taking into account their special characteristics, in terms of sensors and actuators. The objective of this lab is to familiarize yourself with **e-puck**, a typical model of a wheeled mobile robot with differential motion, and to develop a basic controller for safe navigation within a simple maze.

Installation
To work with the Webots simulator, you will need to download:

- the installation file for the latest version **2023b** from www.cyberbotics.com (installation files are available for Windows, Linux, Mac)

You probably have already done this from the previous lab exercise, so you are good to go.

The e-puck Robot
The e-puck Robot (https://e-puck.gctronic.com) is a small wheeled robot, only 7 cm in diameter, 5 cm in height and under 200 grams in weight, moving with the use of differential drive on two wheels. It was originally designed by Michael Bonani and Francesco Mondada in collaboration with three research laboratories of the Ecole Polytechnique Federale de Lausanne (EPFL) in Switzerland. Both the hardware and the software of e-puck are open source, however it is manufactured and released commercially by GCtronic. In terms of sensors, it has 8 infrared sensors on its periphery for measuring the distance to nearby obstacles, a three-dimensional accelerometer, three microphones, and a low-resolution color camera with 640×480 pixels, of which only a subset of 40×40 pixels can be processed at each time. Its actuators are two stepper motors, one for each wheel, with a total of 1000 discrete positions spread evenly over one full spin of each wheel, a speaker, and a series of LED diodes on the periphery of the body, on the body and on the camera. Various add-ons are also available: turret with 1D or 2D omni-directional camera for optical flow study, ground color sensors for line tracking, turret with colored diodes LED for optical communication, and magnetic wheels for vertical climbing.

Manuals
The main manual available for e-puck is an article published in 2009 at the international conference Robotica 2009. You will also find it in the lab material on eClass.

- Mondada, F., Bonani, M., Raemy, X., Pugh, J., Cianci, C., Klaptocz, A., Magnenat, S., Zufferey, J.-C., Floreano, D. and Martinoli, A. *The e-puck, a Robot Designed for Education in Engineering.* Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions, 2009.

Important technical information is however included in the relevant section (E-puck) of the Webots Cloud (https://webots.cloud/proto) linked from the Assets section in the Webots User Guide. Take a look to see the placement of distance sensors on the robot's body, the relationship between distances and measurements, and the interfaces provided in the code for movement and distance measurements.

Rat's Life
The Rat's Life (ratslife .org) is a robotics competition officially held in the years 2009–2011 in the simulation environment Webots. Essentially, it is a survival game, where two e-puck robots (rats) compete against each other for resources scattered around an unknown (random) maze. Like rats in nature, the two robots must search for food that will allow them to outlive their opponent. In our case, each robot can be "fed" at one of four power stations placed at random locations within the maze, where its battery can be charged. Once it locates and approaches a power station, the robot draws available energy, revives, and the station is no longer available for charging for a period of time, until it has accumulated new energy and is ready to deliver it. Therefore, the robot will have to explore the maze, looking for other stations as well, while at the same time it will have to remember which energy stations it has previously located, so that by following suitable routes, it will be constantly refueled. At the same time, the opponent does the same, so even if our rat successfully reaches a station, it is possible to find it (temporarily) empty. Time runs mercilessly, energy dwindles relentlessly and the search for food continues. The key question is the following: which of the two rats will be able to live longer? Rat's Life is a really smart competition, where the basic problems of a robotic agent come into play: sensor data processing, motion control, path

planning, localization, mapping, decision making strategy. Rat's Life has also been implemented and run in a physical environments with Lego mazes and real e-puck robots.

Procedure

Start Webots on your computer[1]. Then, copy the folder `webots/projects/samples/contests/ratslife` to some user space on your disk, without Greek characters in the path (all references hereafter will be to your copy). From `File`, choose to open the `ratslife.wbt` world, located in the `ratslife/worlds` folder. Once you start the simulation, the random maze will be generated in the GUI and you will see the two e-puck robots starting at random positions. The two robots are controlled by Java code located in the `ratslife/controllers/Rat0` and `ratslife/controllers/Rat1` folders respectively. Press `Clean` and `Build` to build the executables (and to make sure everything is working properly with Java – check also https://www.cyberbotics.com/doc/guide/using-java in the Webots User Guide).

Experiment

Watch the simulation for a while and observe the rat's behavior. Also, open `Rat0.java` (`Rat1.java` is the same, only the name is different) in the edit window and try to understand the rationale contained in the code. Ignore the parts that deal with the camera and the LEDs (you can also comment out the corresponding lines of code) and focus only on the parts that deal with distance measurements and wheel movement. The provided controller essentially implements a Braitenberg Vehicle[2], where sensor values are fed directly to one or both actuators, after some weighting with suitable weights to yield a desired navigation behavior. There are two weight vectors in the code; locate them and try to understand their role. In each control cycle, the robot reads the values of the distance sensors and calculates some values for the wheel velocities. Notice how these values change when the robot decides to initiate a turn in place (no translation, only rotation) to avoid obstacles. Play with the weight values and test your own weight vectors. Also, change the code so that the robot implements a left (or right) wall following navigation behavior, where the robot moves throughout the maze, always following the wall on the left (or right) side. In other words, the robot is constantly moving along the wall, always with its left (or right) "hand" resting on the wall on the left (or right), which is roughly what you would do, if someone puts you in a totally-dark maze. What is special about such a behavior, when used in a maze? Search online or elsewhere to find the answer. At each code change, press `Clean` and `Build` to rebuild the executable and restart the simulation with `Reset`. Delete the other robot from the scene tree on the left, if needed, to reduce the computational load, and possibly some walls, if you want to have more space for your robot. Save the modified world as a separate file with your own filename.

Exercises

Now, it is your turn to program a different and interesting navigation behavior on the e-puck rat of Rat's Life. Let's say that your rat is a weird one and likes to move in reverse (backward motion)! Modify the code of `Rat0.java` to implement the **left** wall following behavior, but with **reverse** (backward) movement! Be careful, because the layout of the distance sensors at the back of the robot is different compared to the front, apart from the fact that there are fewer sensors at the back! So, you will have to create your own weight vectors and probably change the way the robot turns. By double clicking on the robot you can see in a separate window the current values of the sensors at any time. By holding down `Shift`, you can move the robot with the mouse and place it wherever you want in the maze. You may notice that the initial behavior sometimes gets the robot "stuck" at certain positions. Of course, you will have to make sure that this will not occur! But, if it unexpectedly does occur, how could you deal with it? Just think about whether a bit of randomness is sometimes useful. Your final code should work in any random maze, as generated by the controller at the beginning. You should constantly monitor in the simulation whether your goal is being achieved and correct accordingly. When you succeed, also modify the code of `Rat1.java`, so that the other rat implements **right** wall following again with **reverse** movement and place them both in the maze to race.

Report/Delivery/Grading

Compress the `ratslife` working folder, which also contains your code (`Rat0.java` and `Rat1.java`). Record a video of your rats' final behavior through Webots (see `File`), but make sure the video file size is kept small, say under 10MBs. Write a short report (maximum one page, in PDF) describing your work and answering the question posed above about left (or right) wall following. Finally, submit code, report, and video as three separate files via eClass through the Assignments section. Your grade will be determined by the completeness of your work.

[1] Select Pause for Startup Mode from within Tools-Preferences to prevent the simulation from starting automatically.
[2] See related text (Braitenberg Vehicle) on Wikipedia or in the lab material on eClass.