Task sheet 1 – Getting started

Deadline : 22.10.2025

Objectives:

- Implement a simple robot simulator and test it with simple Braitenberg vehicles
- Implement proximity sensors and control a robot by simple rules

1 Simulator and Braitenberg vehicles

The idea is to simulate Braitenberg's vehicle 2 (fear and aggression).

a) To do so we first need a simulator[1] with the following properties:

- space is a torus (Pac-Man-like, what exits to the left re-enters at the right etc.)
- the vehicle/robot is represented by position (i.e., the robot's center), heading, and speed
- we assume a light source on the torus that generates a light intensity field which simply decreases linearly with the distance from the light source (i.e., cone-shaped); your implementation should allow to determine the light intensity for any position in space

  Grid resolution 1mm
- try to think of a method how to update the robot's position and heading over time (we discretize time); you should limit the maximal distance that the robot is able to cover in one time step and especially the angle that it can turn per time step.

  5 degrees and 1 sqaure per time step

b) Next we implement Braitenberg's vehicles. First we need two light sensors. Most important is their position relative to the robot's center and the robot's heading. We would like to have one sensor to the robot's front left and one to the robot's front right. Make yourself acquainted with the geometric conditions (sketch!) and implement the calculations of the sensor positions depending on robot position and heading. It's essential that the difference $\Delta s = |s_l - s_r|$ in the light intensities measured by the two sensors is actually detectable ($\Delta s > 0$, if necessary re-parametrize the light source and the distance between the sensors accordingly).

Now we have to implement the motor control. The idea of the Braitenberg vehicles is based on a differential drive. Here we implement that in a rather abstract way. We have a velocity v_l for the left wheel and a velocity v_r for the right wheel. Each of them is proportional to the current sensor reading from the left sensor s_l or the right sensor s_r respectively. In the case of the aggressor, v_l is proportional to s_r and v_r is proportional to s_l. In the case of the 'scaredy-cat' (fear) it's the other way around. We say that the change of the robot's heading ϕ per time step is simply determined by $\Delta\phi = c(v_r - v_l)$ for a constant $c > 0$.

Try to get everything running, play around with parameters, check the robot trajectories for different initializations, and for both vehicles.

[1] You could check out Python packages such as pygame https://www.pygame.org/wiki/about

2 Proximity sensors and control by rules

We reuse the code from task 1 with two changes. First, space is bounded now, we assume a square surrounded by walls. In addition, we add also walls within the square (see Fig **??**). Second, we replace the light sensors with three proximity sensors.

a) A proximity sensor detects the distance to obstacles within a certain range. The range r of our sensors should be about 15% of the square length. One sensor should point to the front left (similar to the light sensor before), one just straight forward, and one points to the front right (see Fig **??**). Imagine the sensors as beams which is a relatively good abstraction of infrared sensors. One option to implement a function that determines the distance of the robot to an obstacle (here it's just our walls) is therefore to follow a virtual line pointing away from the robot's center in a defined angle to the robot's current heading (cf. ray tracing). Another option would be to calculate a lookup table beforehand. Say there is an obstacle within sensor range r at a distance d (i.e., $d \leq r$) and it is actually detected by the sensor. Then the sensor should return the value $1 - d/r$. If the sensor detects no obstacle within range then it should return zero.

b) Now we define a simple controller that lets the robot navigate this little environment. The task is to avoid collisions and to explore the environment. Try to implement simple if-then rules by hand (no learning, no evolution yet) that react to particular sensor conditions and change the robot's heading accordingly.

Test your implementation, initialize your robot with different positions/headings, make sure that it does not pass through walls or leaves the square (check the implementation of your sensors).

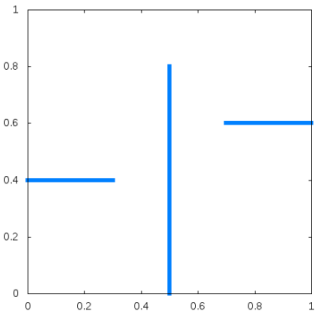

Figure 1: Example of a simple environment with walls.

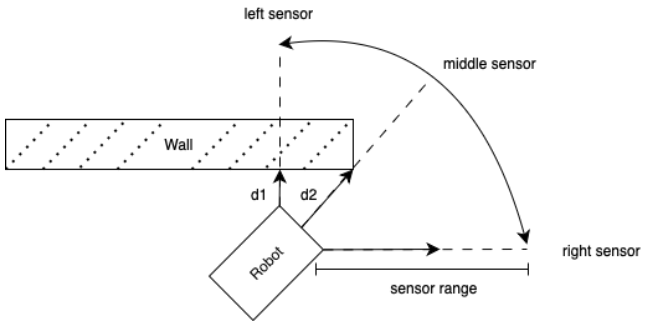

Figure 2: Sketch of robot, sensors, and detected distances to a wall.

Submission Instructions:

- Please prepare a PDF document with the full names of all group members, a list of the tasks and subtasks you have completed (with written answers if applicable) and/or a list of tasks/subtasks you have not completed
- If your solution is in the form of code, please write instructions on how to execute your code
- Make sure to add any relevant plots/diagrams/figures to the document
- All your code (either in C/C++ or python)
- Please zip your submission in a single file named:

  `evoRobo_sheet1_YOURLASTNAME1_YOURLASTNAME2.zip`

For task 1 please turn in:

- optional but very preferable: include a video where you go through and explain your code
- A plot of the light distribution in space (ideally as perceived by the robot's sensors)
- a plot and/or video of a typical trajectory for vehicles 2a and 2b (fear and aggression) as generated by your own code

For task 2 please turn in:

- Optional but very preferable: include a video where you go through and explain your code
- A plot and/or video of a typical trajectory for your robot navigating/exploring the arena as generated by your own code and your hand-coded controller