Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Self-Organising Systems
Winter Semester 2025
Prof. Dr. Javad Ghofrani

# Task Sheet 3 - Analysis of machine behaviors

In this task sheet, we investigate and interpret the behavior of several black box systems. The architecture and components of such a system are hidden, so you can infer the system's functionality only based on the input stimuli and the resulting output responses.

Please save all plots to files, so they are available without executing the code. All results should be bundled in an archive file with your name as the file name.

**Deliverables**:

- A set of plots (saved to file)
- Code
- Descriptive explanation of your understanding
- Readme file to run the program

The submission should be bundled in a single zip file. The file should be named `*yourname*_sos_ex03.zip` Please check the LEA course page for your submission deadline.

## 1 Time series

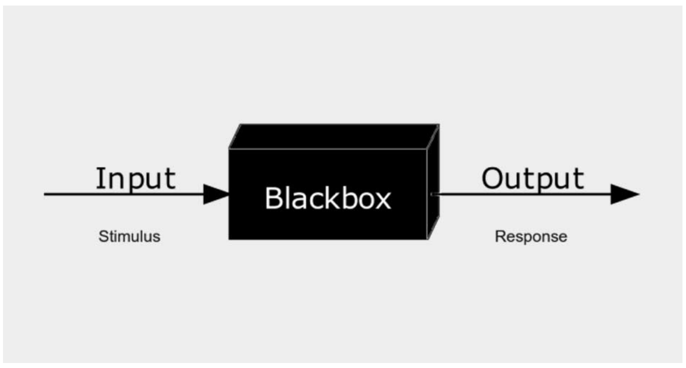

Figure 1: Blackbox system schematic

To train the analysis of machine behaviors, we start with a simple time series problem. Analyze the time series generated by black-box 1a and black-box 1b. What types of systems do they represent?

Both black boxes are implemented as C functions. Write a `main` function that generates inputs for the black boxes and saves the corresponding outputs. Visualize these outputs and explain what you see.

To access both functions, download the files `blackbox1.c` and `blackbox1.h` from the Moodle and use an `include` statement to access them in your `main` function. Document your steps in a detailed way. What were your initial hypotheses, and how did you decide to test them?

As a final result (apart from a documentation how you go to that result) we expect the type of function `boxA` respectively `boxB` are and, if you can determine it, their parametrization.

a) Analyze the time series of `boxA`.

b) Analyze the time series of `boxB`.

**Hint:** Both functions are obfuscated. The purpose of this task is not (just) to find out what type of system each function represents but also to see how you determined it. Therefore, "cheating" and deobfuscating the code will not give you a lot of points in this task.

## 2 Robot Behavior

For a student project, Prof. Javad has given one of his students free reign over his robot laboratory. They did not agree on a task for the student project, except that it is supposed to be about *Evolutionary Robotics.* While Prof. Javad attended a conference, the student finished their project and went on vacation. After the professor has returned, he is left with a setup for robot experiments and a readily trained **artificial neural network** (ANN) with fixed weights and network topology, but he does not know what the robot is supposed to do.

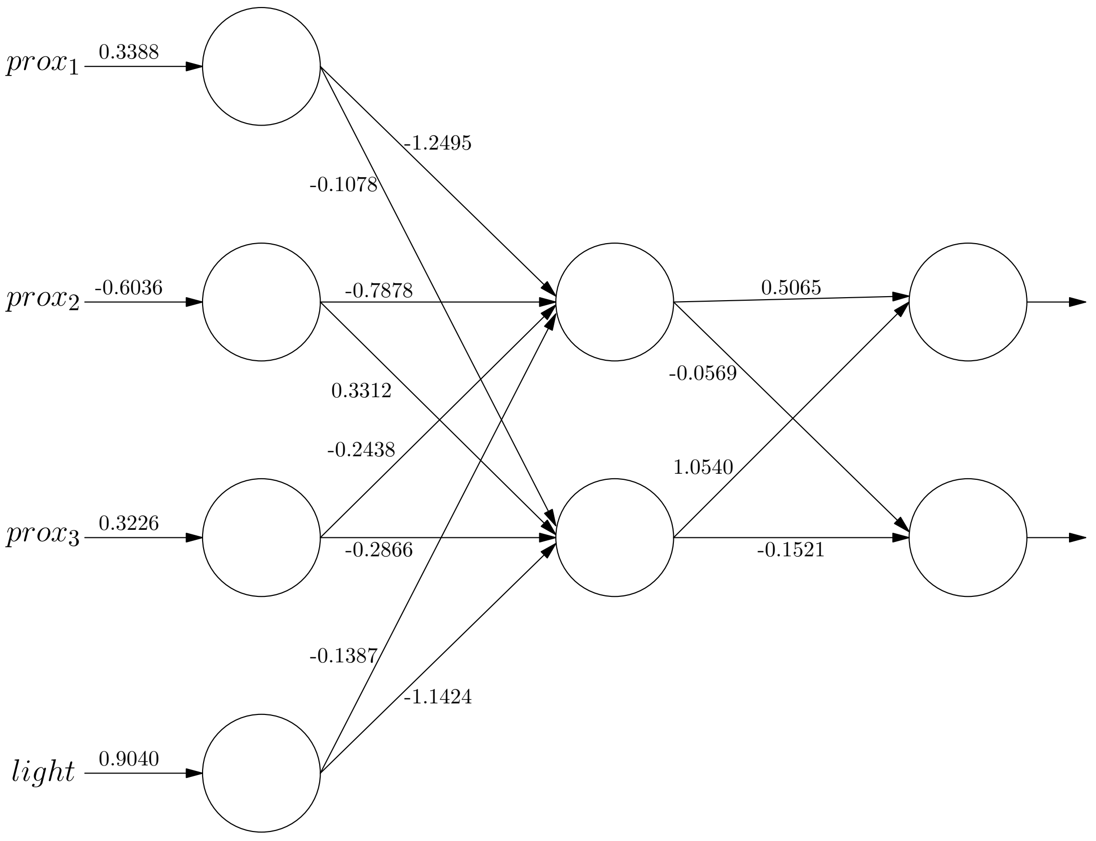

Figure 2: The ANN representing the analyzed robot behavior.

The experiment setup is the following: a $1 \times 1$ **robot arena** with walls (as seen in Fig. 3), a movable **light source**, and a differential drive **robot** with three obstacle sensors and a single light sensor.

The student has initialized the walls in the arena, implemented the obstacle sensors (using the given `raytrace` function, as seen in Fig. 4), and evaluated the given ANN. The ANN's inputs are the outputs of the robot's **three obstacle sensors** (left ($prox_1$), middle ($prox_2$), and right ($prox_3$)) and of the light sensor ($light$). The ANN's outputs are the **motor values** for the left and the right motor of the robot. The output of the **light sensor** is the squared distance between the robot and light source in meters.

The student has computed the **robot's movement** in each time step. The robot moves with the given constant `SPEED` in each time step. Turns happen instantly and do not consume a time step. The robot cannot leave the arena. If the robot leaves the arena, its position is reset to the edge of the arena.

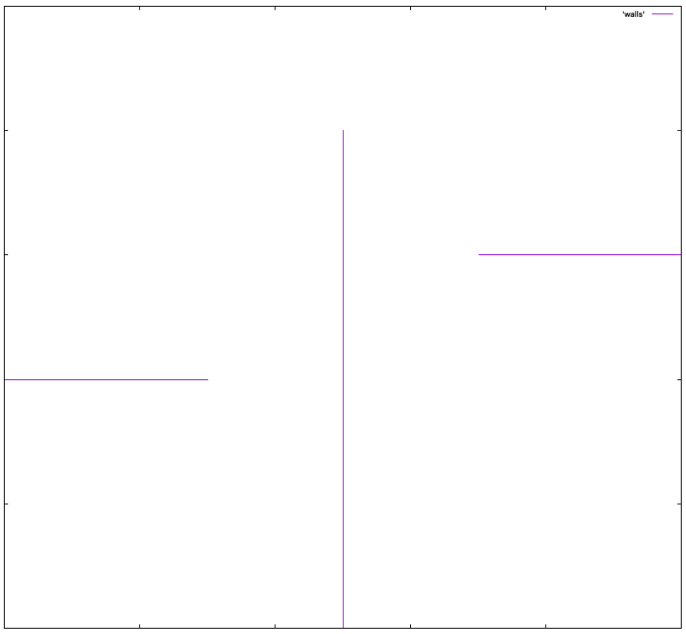

Figure 3: The arena setup for the robot experiments.

As a last defense, the student has implemented a **hardware protection layer** that activates in case one of the robot's obstacle sensors senses a value above 0.95. The angle between the middle and left or right obstacle sensor rays is 18° each. If the hardware protection layer is activated, the robot does not change its heading but moves backward (using negative `SPEED`).

To make the simulation more realistic, the student has added random, equally distributed **noise** between 0.00 and 0.02 to the inputs of the ANN.

a) To test and examine the provided ANN, you are provided with a simulation of the experiments in the `main` function of the given `robotSimulation.c` file.

   Add logging outputs of all relevant features of the robot (for example heading, speed and location) to the code at the appropriate places.

b) Plot and analyze a few single robot trajectories generated by the provided ANN. Test different initial positions of the robot and different positions of the light source. Try to find a good description of the robot behavior. Document your steps in a detailed way. What were your initial hypotheses, and how did you decide to test them?

c) Do an empirical study of the ANN by letting it run for several hundred times. Similarly to task 2b, use several different robot starting positions and light source positions. Keep track of the ANN's outputs for each of the robot's possible positions over time across all runs. This can be done by measuring a (2D-)histogram, for example, of the average of both ANN outputs summed for any observed position.

d) Analyze the ANN itself (shown in Fig. 2) and explain why your description in above task 2c is confirmed by the behavior defined via the weights of the ANN. Please document how you analyzed the ANN.

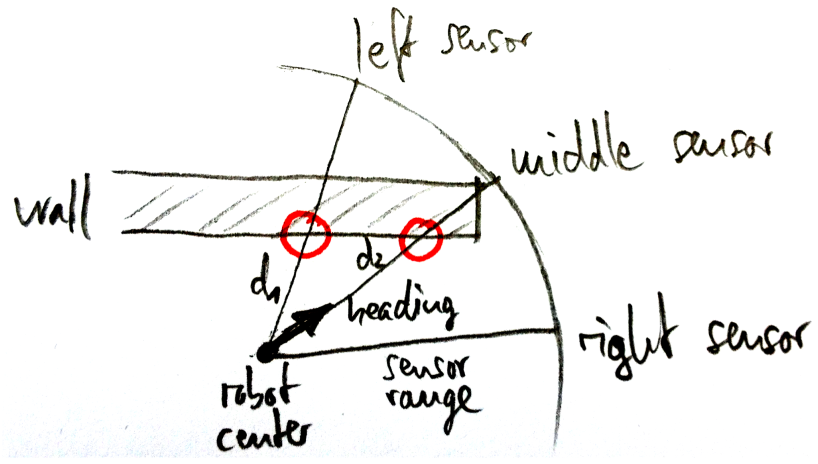

Figure 4: Sketch of robot, obstacle sensors, and detected distances to a wall. The `ray-trace` function returns distance values between the robot center and the collision points of the sensor rays with the wall (as an example shown in the sketch as $d_1$ and $d_2$ for the left and middle sensors).

e) You often gain deeper insights into a mechanism when you try to recreate it. Therefore, re-implement the described robot behavior using a finite state machine. You may reuse `robotSimulation.c`, but you may not use the ANN saved in `network`.

f) Compare your implementation's trajectories with those produced by the original code. For this, come up with a comparison metric. This metric should take both trajectories as inputs and output a number describing the (dis)similarity of the trajectories. Argue why your chosen metric is appropriate for this use case. Try to maximize the similarity of both trajectories and show the metrics values for several iterations of your code.

   A simple example for a metric could be the Euclidean distance of the robot positions measured at certain time intervals, e.g., every 20th time step.