Lab 5: Maze Challenge

January 4th, 2021

Introduction

Bonus points In this lab, you can earn up to **6 bonus points**. You have time to solve the maze challenge until Sunday, January 24th. In total, you are required to submit three things for the full bonus points:

- The source code for the maze traversal program
- A UML class diagram
- A presentation video

Winner of the
Mazebot Challenge
Design and Implementation of Software Systems
smartPORT @ TUHH
2020: Maxine Musterfrau and Max Mustermann

The source code should be in a state that it is directly compilable without having to make changes, e.g., in the `Main.java`. As usual, commit and push it to the master brach of your group's Git-repository.

Attention: According to the regulations at the TUHH, we are only allowed to **include your bonus** in the final grade if we have your matriculation number to uniquely associate the bonus points with your written exam. Therefore, please **commit a text file containing your names and matriculation numbers** to the root of your repository.

The UML class diagram should show all classes and interfaces you implemented for your solution. Classes and interfaces that were provided from `LeJOS` or `Java` can be drawn as empty classes, i.e., by omitting the variables and methods. You can ignore the classes that are only relevant for simulation. Include your UML class diagram as jpg, png or pdf file in your repository.

Finally, you are required to record a short **five minute video**, where you explain your solution. Tell us first, how your algorithm for traversing the maze works conceptually (ca. 1 minute), then explain the source code and what problems you are still facing, or what could still be improved. It is important, that **both team mates** participate in this video and have an equal amount of talking time. We reserve the right to adjust the points of a member if he or she is not actively participating. There will be a folder in StudIP for every group, which can only be accessed by members of this group. Please upload your video there.

Bonus points will not only be granted for speed and for successfully navigating through the maze; but you can also earn points if your robot fulfills basic requirements, i.e., driving, turning, and sensing, reliably.

In summary, the distribution of bonus points is as follows:

- basic functionality: **up to 3 points**
- solving the challenge within the time limit: **up to 1 point**
- explanations, code quality, and UML diagram: **up to 2 points**

Additionally, all solutions will be subject to a plagiarism check afterwards. We will use your solution uploaded to your Git repository *before* the deadline (Sunday, January 24th, 23:59h).

The challenge will be evaluated in the simulation environment with an undisclosed maze. However, we will also flash your programs on the real robots and record a video for each solution. This gives you a chance to see how your solution performs in real conditions. We will share the videos with you later on. Note, that the behaviour on the real robots will **not** influence grading.

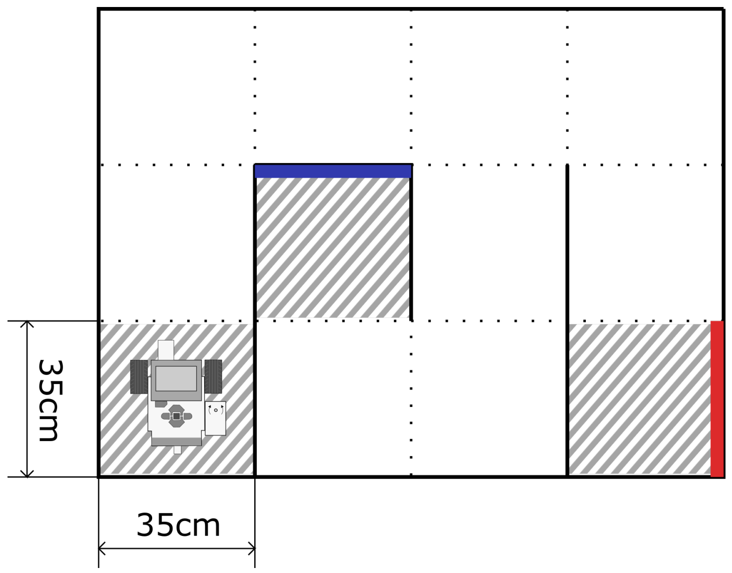

Figure 1: The final maze will have the same tile size, but the wall configuration and starting location will be different. It can have a maximum of 4 times 4 tiles, but may also be smaller.

Award & Ceremony From all submissions, we will select the three best solutions. The selection will not solely be based on speed, but we will also consider other criteria. We will then show how the robots perform in the maze during the last lecture date on January 27th together with the one-minute conceptual explanation of the solution from the recorded video.

The winner will be decided by a live popularity voting of all participants, and the trophy will be engraved with your names and displayed permanently at our institute.

The Challenge

In the final challenge, we will place your robot in an undisclosed maze. The goal is to find a target wall with a specified color within the maze. When your program starts, it should show a menu in which the target color can be selected. The colors are limited to the basic colors red, green, and blue.

Once the user selects a color and presses the enter key on the robot, it should begin to search the maze for the correspondingly colored wall. The moment the robot starts moving, it should produce a beep, and once the robot has found the target wall, it should produce another beep. We will measure the time between the beeps to enforce the time limit (5 minutes, tentative). The time limit is measured at normal simulation speed, of course.

Since there may be more than one colored wall, you must make sure that you detect the right color. If the robot beeps in front of any other wall, this will result in a penalty. It is sufficient to use the color IDs that the color sensor of the robots provides.

Figure 1 shows an example of a maze. The final maze will not be the same, but you can make the following assumptions.

- It will be built in a grid system, where each tile is 35 cm $\times$ 35 cm in dimension.
- It consists of maximum 4 $\times$ 4 tiles.
- It will be closed, so that the robot cannot leave the maze at any location.

- The maze does not contain loops.
- The target wall will be (one of the three walls) in a dead end. As an example, the tiles with striped background in Fig. 1 are candidates for a final wall.
- There may be other colored walls in the maze, but only one that has the specified color (of the target wall).
- The start position is not revealed before the final date. You can, however, assume, that your robot will be placed in the center of a tile at start.

Hints During the previous lab sessions, you have used and implemented the basic functionality of the robot, that you can reuse to solve the final challenge. However, to ensure that you can explore the whole maze and don't miss any branches, you have to think of a suitable strategy.

Maze Simulator

Since you can only test your solution in the simulator, we will only consider the performance in simulation for grading. For testing, you can choose different mazes. The simulator release contains several mazes as png-files and we will provide some additional mazes soon, so stay tuned for a new release over the next days. You can also configure gitlab to notify you about new releases in the project by clicking the little bell-symbol on the project page (here) and selecting *custom* and then tick *release*.

However, when you change the maze file, you need to make sure, that you also adapt the real width and height of the maze in the code. Consider the following lines of code in the `SimulatedMain.java` file:

```java
MazebotSimulation sim;
GuiMazeVisualization gui;
sim = new MazebotSimulation("Mazes/TestArea.png", 1.5,  1.5);
gui = new GuiMazeVisualization(1.5, sim.getStateAccessor());
sim.setRobotPosition(0.75, 0.75, 90);
```

The constructor of `MazebotSimulation(string path, float width, float height)` takes three arguments, where the first is the path to the maze png file relative to the main project path. The second and third are the width and height of the maze in meters. So if you choose a different maze-file, which is not quadratic anymore, you need to change these parameters accordingly. Examplarily, consider a maze that is three tiles wide and 4 tiles high. The width would then be $3 \times 35\,\mathrm{cm} = 1.05\,\mathrm{m}$, while the height is $4 \times 35\,\mathrm{cm} = 1.4\,\mathrm{m}$. If the width and height of the png image does not fit the given width and height, the simulation will not perform correctly. Same applies for the initialization of the `GuiMazeVisualization(float mazeWidth, SimStateAccessor a)`. For more details, see the `wiki` of the simulator. Note, that you can vary the starting position using the `setRobotPosition(double x, double y, double orientation)` command. Where x and y coordinate are given in meters and the orientation is given in degrees.

As you might have experienced already, the simulation does not have any collision detection. So if you ignore the distance sensor output and just keep driving straight through a wall, we consider that a violation of the rules and that will result in point deduction.

In case you encounter any problems or find bugs, do not hesitate to contact us. To do so, submit an issue to our issue tracker in Git. We will try to fix them timely.

—Good luck and have fun—