Lab 4: Navigation

Submission instructions: Students are expected to work in their assigned lab groups. The lab consists of three components; demonstration, lab report, and code submission. Instructions on the demonstration can be found in this handout. Lab reports and code submission must follow the guidelines established in this handout and for the course. For more information, see the ECSE211SubmissionInstructions.pdf on MyCourses.

Design objectives

1. Design a system that allows the robot to drive to a specified set of coordinates, known as waypoints.
2. Implement the design using previous implementations of the Odometer.
3. Evaluate the design and ensure it can navigate effectively around a field.

Design requirements

The following design requirements must be met:

- The robot must navigate through a series of specified points using the minimal distance.
- A list of waypoints must be provided in one location in the code.
- The robot must use the grid system found on the play floor, where the origin is the starting corner of a tile.
- Waypoints will be given with respect to the tile grid system. For example, (1, 0) will be located 30.48 cm to the right of the origin (0, 0).
- When turning to a waypoint, the robot must use the minimal angle needed to turn to it.

Demonstration (30 points)

The design must satisfy the requirements by completing the demonstration outlined below.

Design presentation (10 points)

Before demoing the design, your group will be asked some questions for less than 5 minutes. You will present your design and answer questions designed to test your individual understanding of the lab concepts. Each person will be graded individually.

Simple Navigation (20 points)

Your robot will be placed in a corner of the playing field with walls, exactly as in the previous lab. Once again, your robot should localize and navigate to (1, 1). From that point on the 4x4 tile grid, the robot must travel through 5 waypoints. The TA will specify the waypoints during the demo (see **FAQ**). You will add them to your code and upload your solution to the robot. An example is shown below in **Figure 1** using a **minimal angle** needed to turn at each waypoint. A **maximal angle turn** must be avoided (see **FAQ**).
It is highly recommended that the robot localizes every few waypoints (at least twice), both to increase accuracy and to help in preparation for the final project

- **10 points** are given for turning using a **minimal angle** **(marked in green in Figure 1)** at all the waypoints. **0 points** are given if the robot turns using a **maximal angle** **(marked in red in Figure 1)** at any of the 5 waypoints.
- **10 points** are given for reaching the last waypoint within an error tolerance of **3 cm** using a *Euclidean distance measure*. The *Euclidean distance error* **ϵ** is defined by the formula shown below, where X_D and Y_D represent the destination waypoint, and X_F and Y_F represent the final position of the robot. Penalties are defined below.

$$\epsilon = \sqrt{(X_D - X_F)^2 + (Y_D - Y_F)^2}$$

BONUS: Simple Navigation on larger area (10 points)

10 bonus points will be awarded if the robot, starting from the (1, 1) point on the 8x8 tile grid, can reach the last waypoint within an error tolerance of **3 cm** using a *Euclidean distance measure*.

The following point grid for navigation is used:

- **[0, 3] cm → 10 points**
- **(4, 6] cm → 5 points**
- **(6, ∞) cm → 0 points**

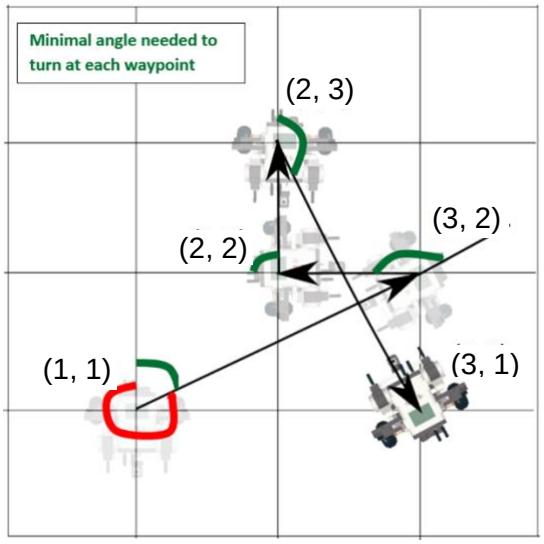

Figure 1: Example robot path 1 (labelled)

Provided materials

Sample code

No sample code is provided for this lab. Instead, follow the guidelines given below. Create a `Navigation` class that has at least the following methods:

- `travelTo(double x, double y)`
  This method causes the robot to travel to the absolute field location (x, y), specified in tile points. This method should continuously call `turnTo(double theta)` and then set the motor speed to forward (straight). This will make sure that your heading is updated until you reach your exact goal. This method will use the odometer.

- `turnTo(double theta)`
  This method causes the robot to turn (on point) to the absolute heading theta. This method should turn a *MINIMAL* angle to its target.

Physical material

In the lab, there will be floors with the grid available.

Implementation instructions

1. Adjust the parameters of your controller, if any, to minimize the distance between the desired destination and the final position of the robot. Simultaneously, you should aim to minimize the number of oscillations made by the robot around its destination before stopping**.**
2. Write a program to travel through a set of waypoints. For example, **Figure 2** shown below follows (3, 2), (2, 2), (2, 3), and (3, 1) in order.

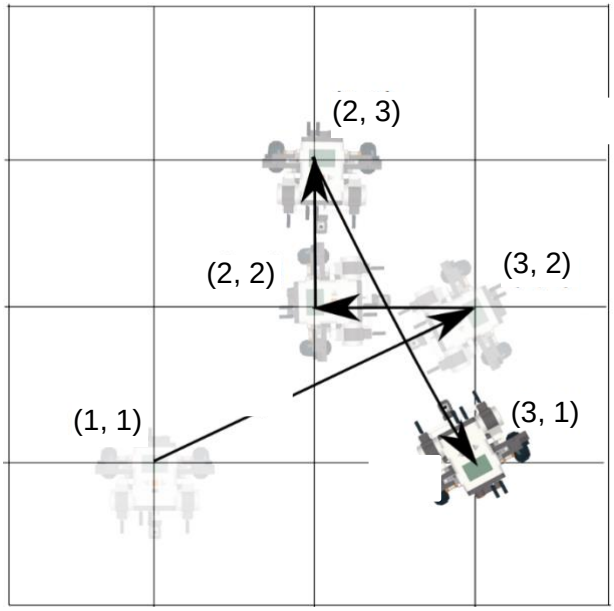

Figure 2: Example robot path 1.

Report Requirements

The following sections must be included in your report. Answer all questions in the lab report and copy them into your report. For more information, refer to ECSE211Submission Instructions.pdf. **Always provide justifications and explanations for all your answers**.

There is a page limit of 6 pages, in 12 pt font, (approximately 3000 words) for this report, excluding images and tables. You can either include images/tables in the report (they will not be included in your page count) or create an appendix and reference it in the report. The appendix will not count as part of your page count.

Section 1: Design Evaluation

You should concisely explain the overall design of your software and hardware. You must present: (1) your workflow (2) an overview of the hardware design (3) an overview of the software functionality. You must briefly talk about your design choices before arriving at your final design. Visualizing hardware and software with graphics (i.e. flowcharts, class diagrams for workflow and software, and pictures for hardware) must be shown.

Section 2: Test Data

This section describes what data must be collected to evaluate your design requirements. Collect the data using the methodology described below and present it in your report in a formatted table.

Navigation test (*8 independent trials*)

1. Program the robot to travel to waypoints (3, 2), (2, 2), (2, 3), and (3, 1) as in **Figure 2**.
2. Note down the destination waypoint (X_D,Y_D) in cm.
3. Place the robot on the origin of a grid tile.
4. Ensure there are no obstacles in the robot's path.
5. Run the program to drive the robot through the waypoints specified in Step 1.
6. Record (X_F,Y_F) which is the final position of the robot.

Section 3: Test Analysis

- Compute the Euclidean error distance ϵ between (X_F,Y_F) and (X_D,Y_D) for each trial in a table.
- Compute the mean and sample standard deviation for error ϵ computed in the **Navigation test**. Be sure to use the show general formulas and sample calculations.

Section 4: Observations and Conclusions

Answer the following questions:

- Are the errors you observed due to the odometer or navigator? What are the main sources?
- How accurately does the navigation controller move the robot to its destination?
- At which waypoints did you decide to localize and why? What are the advantages and disadvantages to localizing at every waypoint?

Section 5: Further Improvements

What steps can be taken to reduce the errors you discussed above? Identify at least one hardware and one software solution. Provide explanations as to why they would work.

Frequently asked questions (FAQ)

1. **What is a minimal angle?**
   Referring to the (1, 1) waypoint in **Figure 1**, its **minimal turning angle** is approximately 60° clockwise, as opposed to a **maximal turning angle** of 300° counterclockwise.

2. **What is meant by "design presentation"?**
   Before a lab demo, you and your partner will briefly present your design. This can include a basic visualization of how your code functions, such as a flow chart. You will then be asked a series of questions. These could be related to the lab tutorial or relevant lectures. For this part, a grade of 10 signifies full understanding, 5 signifies satisfactory understanding, while 0 shows no understanding at all. Note that memorized answers are discouraged and both partners should be responsible for understanding the design and their associated code, even if one of them did not write all of it.

3. **Which maps will be used for this lab's simple navigation demo?**
   Your robot must navigate using one of the following maps starting from the point (1, 1):
   Map 1: (1, 3), (2 ,2), (3, 3), (3, 2), (2, 1)
   Map 2: (2, 2), (1, 3), (3, 3), (3, 2), (2, 1)
   Map 3: (2, 1), (3, 2), (3, 3), (1, 3), (2, 2)
   Map 4: (1, 2), (2, 3), (2, 1), (3, 2), (3, 3)

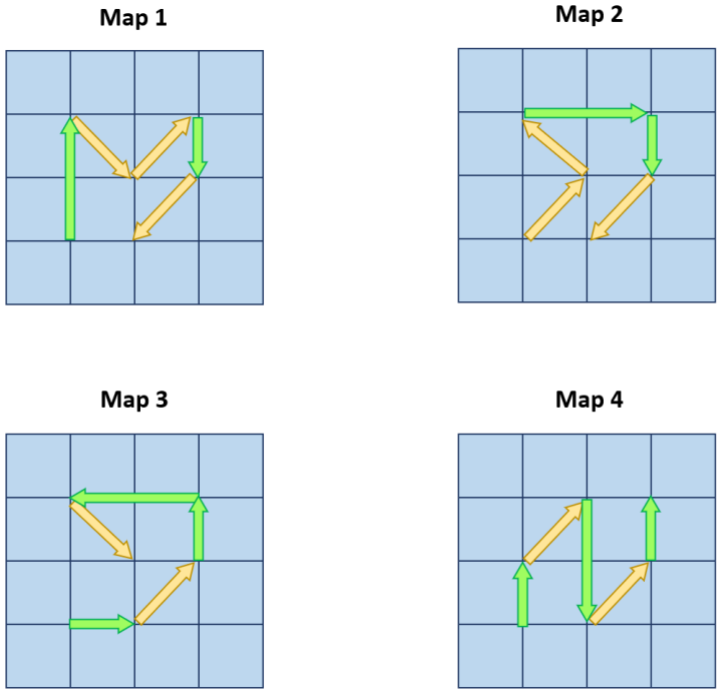

Figure 3: Simple navigation maps

4. Which maps will be used for this lab's bonus navigation demo?

Your robot must navigate using one of the following maps starting from the point (1, 1):

Map 1: (1, 7), (4 ,4), (7, 7), (7, 4), (4, 1)
Map 2: (4, 4), (1, 7), (7, 7), (7, 4), (4, 1)
Map 3: (4, 1), (7, 4), (7, 7), (1, 7), (4, 4)
Map 4: (1, 4), (4, 7), (4, 1), (7, 4), (7, 7)

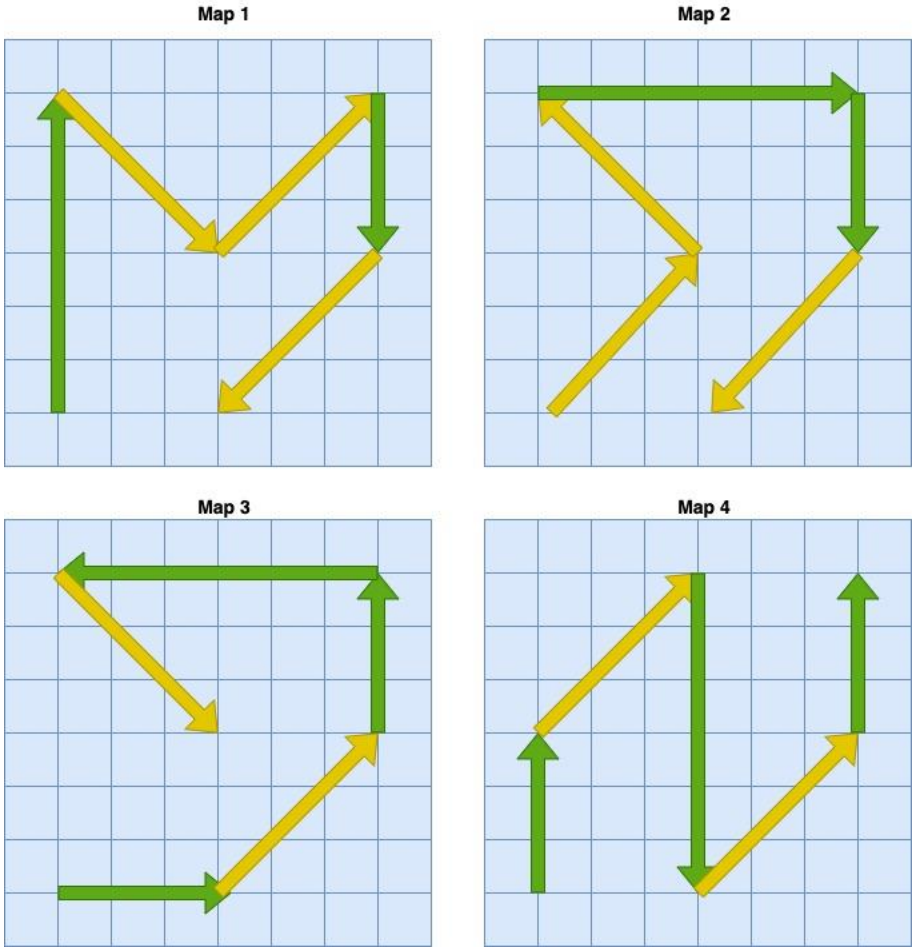

Figure 4: Bonus navigation maps

You must encode all these maps in your code, so you can easily select the correct map to use at demo time. You may create a custom class (recommended) or simply use a 2D array.

5. **How do I test my logic on my computer before running it on the brick?**

Example: You decided to make `turnTo()` return true when the robot rotates clockwise. To develop this method, start by writing the logic to compute the minimal angle. You should be able to test this code on your computer, since you did not yet add the code that moves the motors. To test this logic, call it with different values, eg printing `turnTo(-90)` should show false. Once it is correct, commit your changes.

Note that the `turnTo()` and `travelTo()` methods rely on the odometer state, which is not updated on your computer. So you will want to temporarily use an array of values that you provide instead:

```
static boolean turnTo(double angle) {
  // try multiple values, not just these
  double[] odoValues = new double[] {3, 2, 180};

  // replace the above line with this when you have finished testing
  // double[] odoValues = odometer.getXyt();

  // use odoValues to compute minimal angle (you can make a helper method)

  // after you confirm your logic works, write the code that moves the
  // motors in the correct direction
}
```

6. **Why do I need to test my logic on my computer before running it on the brick?**

In the real world, engineers often need to make sure their logic works before running it on real hardware. As you have seen in the first three labs, testing only by running the code on the brick takes a long time, since it takes a long time to execute even one method call, even though its logic runs in milliseconds. By following a design process such as the one described above, you can save time by catching bugs early.