Lab 4

December 9th, 2019

Bonus Points

For this lab, you can earn up to **2 bonus points**. To receive full credit, draw a UML class diagram representing the classes and interfaces you implement in Tasks 4.1 to 4.3. We will discuss the solutions with every group individually in the second lab session. During this session, we will directly grade the solution and assign bonus points. Therefore, make sure the UML diagram is ready in your assigned lab session in the week from **December 16th to December 20th**. You don't need to have the actual implementation finished until then, a rough skeleton of your classes is sufficient.

In the UML diagram, we expect to see all the classes you implemented yourself with their methods and member variables. For the types that are provided by lejOS, you can draw an empty class, i.e., neglect the methods and variables.

Introduction

In this lab, you will implement the turning functionality for the robot. Read through the Tasks in this lab and prepare the UML diagram **before you start coding**. That means you have to think about which classes and interfaces you need to solve all Tasks and what member variables and methods these classes will have. Also consider encapsulation, i.e., which member variables and methods need to be public, which should be private.

Note: The only method that needs to be `static` in this lab is the main method.

Task 4.1 : Interface for Turner

We want to implement two different ways to turn the robot. Both ways should be interchangeable with each other. Therefore, both should implement a common interface.

Create a new interface and call it `Turner`. The interface should provide the two methods

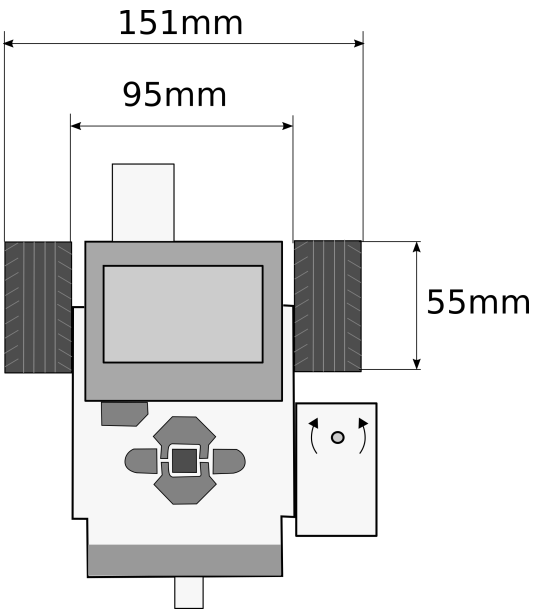
- `void setSpeed(int degreesPerSecond)` and
- `void turn(int degrees)`,

where degrees refers to the orientation of the robot.

Task 4.2 : Simple Turning

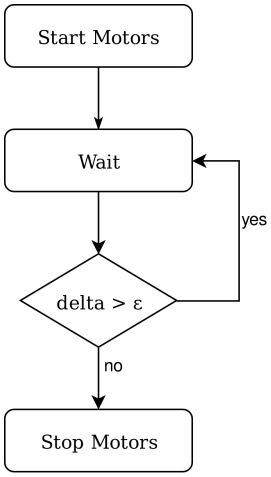
Develop a class that implements the interface defined in Task 4.1. To turn on the spot can be achieved by turning both wheels in the opposite direction with the same speed. You can use the physical properties of the robot (wheel diameter, axis width, gears) to calculate the angle, the motors have to turn. Please refer to the measurements shown in the picture below.

Test your solution. Is the result satisfyingly accurate, when you turn the robot around {90, 180, 360} degrees? If yes, try the same program on a different ground material, e.g., put the robot on the floor and decide if it is still accurately turning. How can the ground material influence the result?



Task 4.3 : Turning with Gyroscope

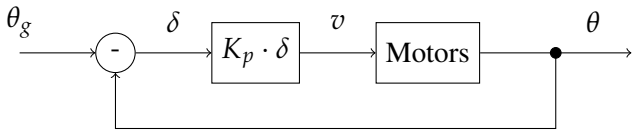
While testing your solution, you have probably noticed that the accuracy varies with different factors, such as ground properties or turning speed. To reliably turn around a given angle, you will need to use feedback.
The gyroscope on the robot provides the current orientation of the robot. The orientation is set to zero, when the Gyroscope is initialized. You can use this orientation as feedback to turn the robot more accurately.
Implement the turning with feedback in a new class. This class should also implement the interface from Task 4.1. One way to use the gyroscope feedback is, to start the motors and continuously poll the angle from the gyroscope. Once the desired angle has been reached, stop the motors.



Task 4.4 : Proportional Controller

The feedback-controlled `Turner` from the previous exercise only stops the motors once the target angle has been reached. However, due to the delayed sensor readings and inertia of the robot, this tends to overshoot. In this Task, develop a proportionally controlled `Turner`.

A proportionally controlled `Turner` adapts the turning speed proportional to the delta between the target orientation and the current orientation. Therefore, the robot will turn fast when the delta is large but slows down when getting closer to the target.



- θ: Current Orientation
- θ_g: Goal Orientation
- δ: Deviation
- v: Motor Speed