Lab 4

December 9th, 2020

Bonus Points

In this lab **tasks 4.1 to 4.3** will be evaluated. You can earn up to **2 bonus points** but only if, in addition to the code, you **draw a UML class diagram** representing the classes and interfaces of your implementation. You should draw the UML diagram yourself. It can be either hand drawn or with the use of a graphics editor of your choice. Just automatically generating the UML chart via Eclipse won't be sufficient. Please upload the chart in an appropriate file format (.pdf, .jpeg, .png).

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.

Submission Deadline: December 20th, 23:59h

Introduction

In this lab, you will implement the turning functionality for the robot. Read through the tasks in this sheet 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 (data hiding).

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

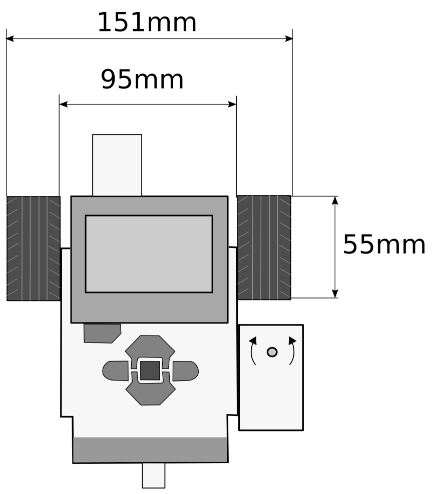
- `void setSpeed(int degreesPerSecond)` and
- `void turn(int degrees)`,

where `degreesPerSecond` is the rotational speed of the robot, and `degrees` is the relative angle that the robot should rotate. Both methods should be able to handle positive and negative input values in order to set clockwise (CW) and counter-clockwise (CCW) rotation.

Task 4.2 : Simple Turning

Develop a class that implements the interface defined in task 4.1. Turning 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.

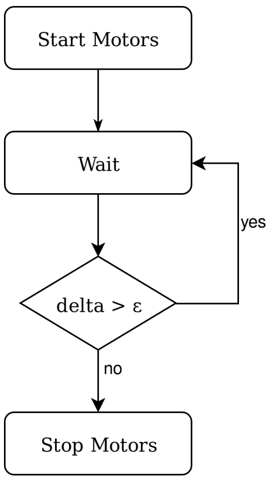
Test your solution. Is the result satisfyingly accurate, when you turn the robot around {90, 180, 360} degrees?



Hint: On the real robot using the exact wheel center distance may not be the optimal parameter, which is due to tire/friction effects. We tried to implement this behavior in the simulator, so you need to tune this parameter in order to get more accurate results.

Task 4.3 : Turning with Gyroscope

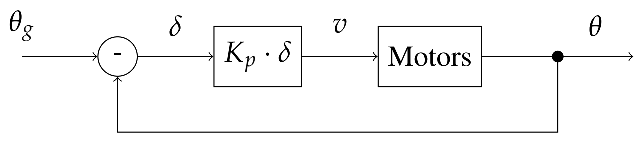
The gyroscope on the robot provides the current orientation of the robot. The orientation is set to zero, when the gyroscope is initialized. You can use this orientation as feedback to turn the robot more accurately.
Implement the turning with feedback in a new class. This class should also implement 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