Lab 4

December 5th, 2023

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

Bonus Points

- In this lab **tasks 4.1 to 4.3** will be graded. You can earn up to **2 bonus points**, if and only if, in addition to submitting the code, you **draw a UML class diagram** representing the classes and interfaces of your implementation.
- The UML diagram can be created either hand-drawn or using a drawing tool of your choice (a free tool for drawning UMLs and that runs directly in the browser: https://app.diagrams.net/). Note, that auto-generating the UML chart via Eclipse is not allowed / will **not** be sufficient.
- The UML diagram must match your implementation, i.e. we expect to see all the classes you implemented yourself including their methods, member and class variables. Don't forget the classes and **types provided by lejOS**. However, you don't need to list their methods and variables, but you may just draw an empty class.
- The UML diagram should be uploaded to the root directory of your git repository named, e.g., **Lab4_UML**. Accepted file formats are: pdf, png, jpeg.
- Again, make sure that your solution also runs successfully in the simulator.
- **Submission Deadline:** Upload your solution to GitLab until **December 22nd, 23:59h**

Task 4.1: Interface for Turner

An interface allows you to implement two different methods for rotating the robot, while both methods are interchangeable.

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.

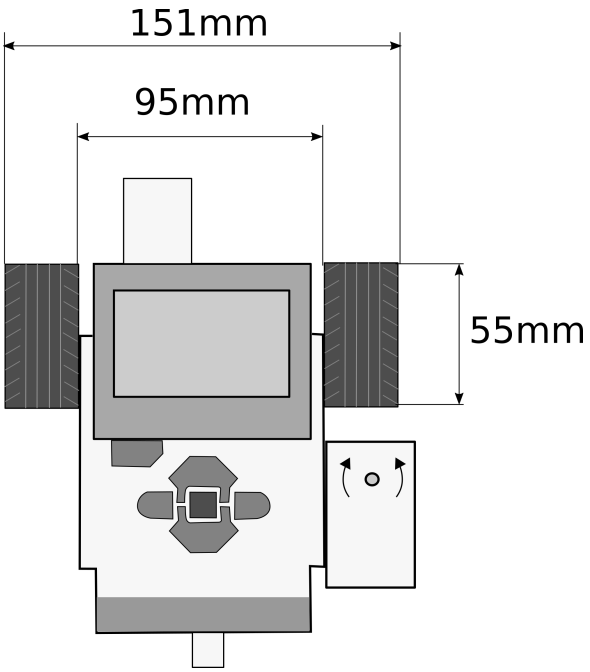
TUHH Hamburg University of Technology

Prof. Dr.-Ing. Bernd-Christian Renner
Institute for Autonomous Cyber-Physical Systems

Task 4.2: Simple Turning

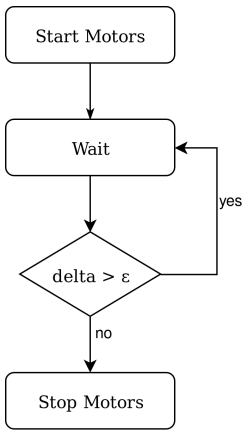
Develop a class that implements the interface defined in task 4.1. Turning on the spot can be achieved by turning both wheels in opposite directions with the same speed. You can use the physical properties of the robot (wheel diameter, axis width, gears) to calculate the angle that the motors have to 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 instead of the table 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 environmental conditions, 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.

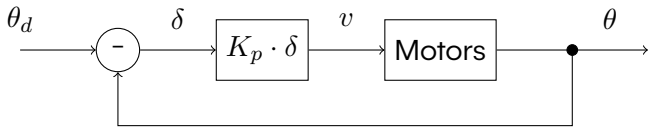


TUHH Hamburg University of Technology
Prof. Dr.-Ing. Bernd-Christian Renner
Institute for Autonomous Cyber-Physical Systems

Task 4.4 : Proportional Controller

The feedback-controlled `Turner` from the previous exercise 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, hence avoiding overshoot.



- θ: Current Orientation
- θ_d: Desired Orientation
- δ: Deviation
- v: Motor Speed

TUHH Hamburg University of Technology
Prof. Dr.-Ing. Bernd-Christian Renner
Institute for Autonomous Cyber-Physical Systems