Lab 2

November 14th, 2022

In this lab, you get started with developing Java programs for the Lego Mindstorm EV3 robot using the LeJOS Application Programming Interface (API). Please find this checklist for preparation hints and additional instructions for this lab. Solve the tasks on this sheet **at home using the simulator**!

Simulator

A Mazebot simulator was requested by students several years ago, and in 2020, it has finally been realized and released by Peter Oppermann. The simulator has not only helped save the teaching during the pandemic, but it has also been proven to accelerate the software development process for the students. Therefore, we strongly encourage using the simulator throughout the course.

The LeJOS setup instructions have already described how to get started with the simulator. Remember, if you want to test your code in the simulator, you have to run the `MainSimulated.java` file, where you have to specify the program to simulate, by calling its main() method in the main() of `MainSimulated.java`. See the example, how it was done for `Task1.java`. To start the simulation, right click on `MainSimulated.java` and **Run As** → **Java Application**.

General Remarks

- The project `/workspace/DISS_Mazebot` which you copied during the LeJOS setup will be your main project for the rest of the course. Maintaining a good package/folder structure is crucial to allow reuse of your code:
    - One option is to create a new package for every lab, and a new class with a separate main function for every task, i.e. naming the class according to the task, e.g., *Task1*. In the following labs, you can copy classes between packages and eventually restructure your code. *This method is recommended for beginners.*
    - Alternatively, you can name and structure your packages/classes directly in a way, that allows re-use and integration in later tasks. *This method is recommended for experienced programmers.*
- Essentially, all programs developed during the labs rely on libraries from the **LeJOS API.** In the lab sheets, we will provide you some hints for the appropriate classes/methods to use, but you are required to study LeJOS API documentation on your own, to find suitable classes/methods that solve the tasks, and eventually use the API documentation to understand how to use and implement the classes/methods.

Mazebot Handling

Here are some hints for handling the Mazebot during the lab:

- **Boot:** Press the middle button for about 1-2 sec to power on the robot. Wait until the operating system has completely booted up, after which a menu will be displayed.
- **Upload Program:** To upload your Java programs, connect the robot using the provided USB cables to your computer, then right click on the desired .java file in the Eclipse project explorer and select **Run As** → **LeJOS EV3 Program**.
- **Emergency Stop:** If your program is stuck in an infinite loop while executing on the robot, you can stop the program execution by pressing the **down arrow and center buttons** on the robot simultaneously.
- **Shutdown:** Press the **back button** (top left) until the shutdown menu pops up. Select "yes" using the right/left buttons and "confirm" using the middle buttom.
- **Hard Reset:** Press **middle and right** button together with on finger, and the "back" button on the **top left** with the other finger all simultaneously until the device reboots

Task 2.1: Hello LeJOS (revisited)

To begin with, let's revisit the Hello LeJOS example, which was given in the setup tutorial. By searching the API documentation → `lejos.hardware.lcd` → `LCD` you will find a description for the method that displays a string on the LCD

```
LCD.drawString(String s, int x, int y)
```

The parameter *s* represents the string you want to print and *x* and *y* specify the column and row on the LCD.

If you run only this code line on the real robot, you will notice that the program returns to the main menu immediately. To see the result on the screen, a delay has been included after printing a string on the LCD

```
Delay.msDelay(int milliseconds)
```

Optional Task (do the other tasks before): You can implement a banner, where the string moves over the screen. You must clear the LCD before writing to a new position, and you need to wait for a certain time before you move to the next position. You can use `LCD.clear()` for this task. You need to know how many characters fit on the screen. Therefore, the LCD class provides the constants

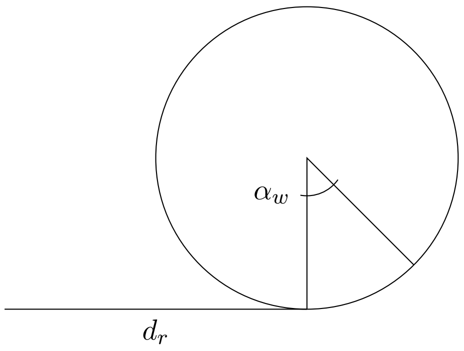
- *LCD.DISPLAY_CHAR_WIDTH* and
- *LCD.DISPLAY_CHAR_DEPTH*.

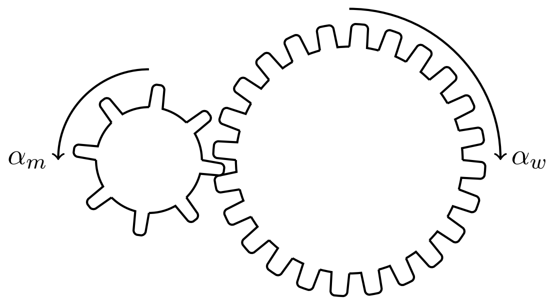
Task 2.2: Straight Driving

Now it's time to get the robot moving. Your task is to write a program that makes the robot drive straight ahead for a fixed given distance input and then stop.

First, think about the angle (α_w), by which both wheels have to turn in order to move the robot by a given distance d_r. Then, you need to understand how the movement of the wheels is connected to the movement of the motor. Each wheel has its own motor. The motor is connected to a small gear with 8 cogs, and the wheel is attached to a larger gear with 24 cogs. Derive a formula to compute the turning angle of a motor α_m necessary to move the robot by a given distance d_r. Implement a method that performs this conversion.

Hint: The robot's wheel diameter is 5.4 cm.





The cables from the motors and sensors connect with the brick at ports. At the front of the brick are the motor ports, enumerated *A* to *D*, on its back are the sensor ports, enumerated *1* to *4*. The right motor is connected to port *C*, and the left motor is connected to port *B*. To call motor methods, you must first instantiate a motor object. The motors for the left and right wheel are represented in the LeJOS API as *EV3LargeRegulatedMotor*. Instantiate two objects of the class, one object for the left motor and one for the right motor. The constructor takes the motor port as an argument.

Here are some code snippets that are useful to complete the task. For an exact description of the methods and their parameters, look again into the LeJOS API documentation.

```java
// Create an instance of the class EV3LargeRegulatedMotor
EV3LargeRegulatedMotor rightMotor = new EV3LargeRegulatedMotor(MotorPort.C);
// Set the desired motor speed
rightMotor.setSpeed(100);

// Start turning the motor forward
rightMotor.forward();
// or backward
//rightMotor.backward();

//...

//check if motor is still busy moving
rightMotor.isMoving();

//...

//stop the motor
rightMotor.stop();
```

Note, that it is also possible to let the motor rotate by a fixed desired angle:

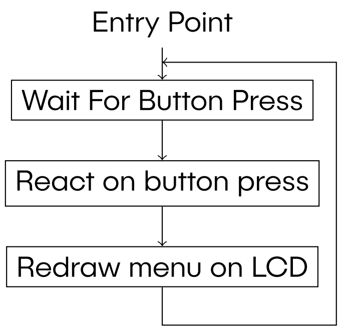
```java
  //...
  //Rotate a given angle, pay attention to the second parameter!
  rightMotor.rotate(180, true);
```

Task 2.3 : Sound & Buttons

The robot has a speaker and can produce sound. In this task, first implement a method that generates a beep. Consider the methods

- `Sound.setVolume(int volume);`
- `Sound.playTone(int frequency, int duration);`

Once the beep works, your next task is to implement a menu, that is displayed on the screen. The menu should allow a user to set the frequency, the volume, and the duration of the beep with the buttons on the brick. The general concept of the menu loop and the menu on the LCD are shown below.



```
Menu
Volume    50
Frequency 400
>Duration  500
Play
```

In the menu, the user should be able to switch between the options by using the `UP` and `DOWN` buttons. When the user is on an option with a numerical value, he or she can increase or decrease the value with the `RIGHT` and `LEFT` buttons. When he is on the *play* option, a press on the `ENTER` button should play the beep with the selected parameters.

The buttons can be accessed via the *Button* object. The method `Button.waitForAnyPress()` blocks the program execution until any button is pressed. Consequently, you need to find out which button is currently being pressed. For example, to check if the Up-Button is pressed, you can use `Button.UP.isDown()`.

Think about the robustness of your implementation. Where do you store the current setting, and which datatypes do you use? In which value range is each setting meaningful? What happens in case of unexpected button presses? Make sure the menu only allows settings within this value range and handles user input that is out of this range adequately.

Attention: In the *simulator*, you can push the buttons of the EV3 robot by clicking on them in the graphical user interface. It may be a bit confusing, that you have to click the button once to press it down, and click it a second time to release it. However, only this way it is possible to klick multiple buttons on the robot simultaneously in the simulation. The buttons have reddish color, when they are in pressed down state, and greyish color, when they are in released state.