Lab 2

November 16th, 2019

In this lab, you familiarize yourself with the Lego Mindstorm robot and the LeJOS Application Programming Interface (API). Unfortunately, we will not be able to use the Robots because of the special circumstances this semester. However, we have build a Simulator, in which we can run the code as on the real robot.

Please make sure you already installed Eclipse and LeJOS and the LeJOS plugin for Eclipse as described in the *Eclipse Tutorial* sheet from StudIP. You should also know how to use Git (see the *Git Tutorial* in StudIP).

During the lab, you need to call the LeJOS API. We will give you hints for the appropriate methods in the lab sheets, but you are required to look at the detailed API definition at http://lejos.org/ev3/docs/ to exactly understand what the methods do and which parameters they expect.

Prerequisites

Important: Once the **team registration** phase is over (**Thursday, November 12th, 23:59**), we will create Git repositories for each group. They will be available on Friday evening. Each repository will contain an empty LeJOS EV3 project, which is already set-up so that you can directly start coding. Clone your group-specific repository and import it into your Eclipse workspace. You can do that within Eclipse, as described in the Git Introduction sheet. You can start in the class `MainSimulated`, which is in the package `de.tuhh.diss.lab.sheet2`. The main function is shown below.

```
1  public static void main(String[] args) {
2      MazebotSimulation sim = new MazebotSimulation("Mazes/TestArea.png", 1.5,
   1.5);
3      GuiMazeVisualization gui = new GuiMazeVisualization(1.5, sim.getStateAccessor());
4      sim.scaleSpeed(1);
5      sim.setRobotPosition(0.75, 0.75, 90);
6
7      sim.startSimulation();
8      gui.startVisualization();
9
10     // Here goes your Code!
11
12
13     Delay.msDelay(100);
14     sim.stopSimulation();
15
16 }
```

It is important, that you place your code at the correct position, so that the simulator can execute it. For now, running just this program should show the simulator window, but the robot is not doing anything yet. For each task in this sheet, create a new class in this package by right-clicking on the package and select *New* → *Class*. Name the class according to the task, e.g., *Task1* and don't forget to check to box for creating a `static void main(...)` method. Then, you can call this function in the simulator by replacing line 10 in the above code snippet by `Task1.main(new String[] )`. Then, right click at the `MainSimulated` class in the Package Ecplorer, select *Run As*→*Java Application*. Now your program should run.

If you are interested in learning how the simulator works or can be used in other projects, take a look at the Wiki: https://collaborating.tuhh.de/cuy1171/diss-mazebot-simulation/-/wikis/home

Task 2.1 : Hello World

Your first task is to use the API to print a string on the LCD screen. Inspect the LCD's methods in the LeJOS API documentation. Use the method

```
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 the code, you will notice that the program returns to the main menu immediately. To see the result on the screen, you need to include a delay after you have printed the string to the display. You can use the `Delay.msDelay(int milliseconds)` method to wait for a given time. Implement a delay of 5 seconds after printing to the LCD.

Bonus: 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

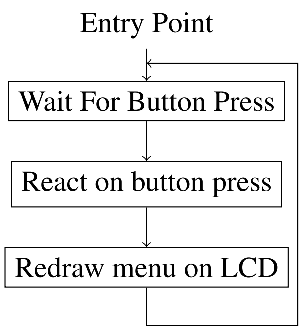
- *LCD.DISPLAY_CHAR_WIDTH* and
- *LCD.DISPLAY_CHAR_DEPTH*.

Task 2.2 : 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 MIDDLE 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()`.

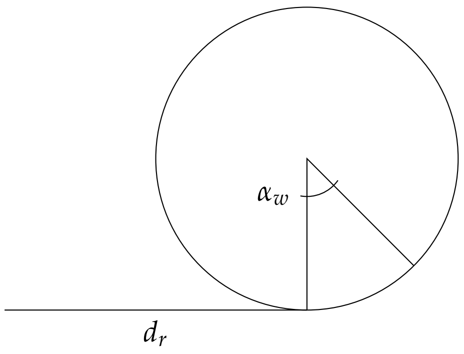
Think about the robustness of your implementation. Where do you store the current setting, and which datatypes do you use? In which value range is each setting meaningful? What happens in case of unexpected button presses? Make sure the menu only allows settings within this value range and handles user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

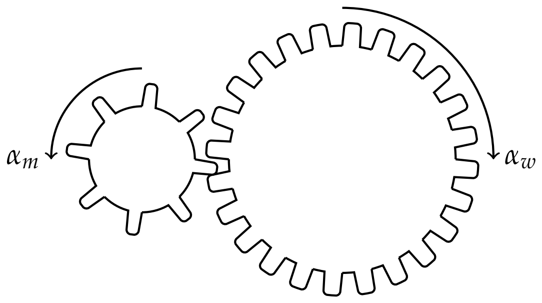
Attention: In the simulator, you can push the buttons of the EV3 robot by clicking on them. It may be a bit confusing, that you have to click the button once to press it down, and click it a second time to release it. However, only this way it is possible to klick multiple buttons on the robot simultaneously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

Task 2.3 : Straight Driving

Now the robot will finally move. Your task is to make it drive straight for a fixed distance 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.





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
// Get motor object
EV3LargeRegulatedMotor rightMotor = new EV3LargeRegulatedMotor(MotorPort.C);
// Set speed of motor (does not start turning yet)
rightMotor.setSpeed(100);
// Start turning forward or backward
rightMotor.forward();
rightMotor.backward();
//Rotate a given angle, pay attention to the second parameter!
rightMotor.rotate(180, true);
//check if motor is still busy moving
rightMotor.isMoving();
```