Lab 5

January 8th, 2018

Schedule

Date	Time	Room
January 8th, 15th, 22nd and 29th, 2018	11:30 - 13:00	Q117 and Q120
January 8th, 15th, 22nd and 29th, 2018	13:15 - 14:45	Q117 and Q120
January 12th, 19th, 26th and February, 2nd, 2018	9:45 - 11:15	Q117 and Q120

Please bring your Windows-notebooks, if you want to flash your robot!

Organization

In this task, you can earn up to **6 bonus points** for the exam. The result of your team work depends on

- functionality (2 points),
- code quality and interview performance (4 points).

The time in the lab is intended for asking questions concerning your implementations and working with the robots and not for solving fundamental programming tasks: work on your implementation at home and test your code with the robots in the lab.

Please note that the robots can only be flashed on Windows operating systems. Please ensure that at least one team member has a Windows notebook with working LeJOS and Eclipse. In case this is not possible, you can flash the robot with us on our computers.

Read the following sections carefully on how to submit and how to earn full credit.

Functionality

Dependent on the functionality level of your implementation, you can earn up to 2 points.

- 0.5 points (basic functionality with major issues)
- 1.0 points (complete functionality, slight issues)
- 2.0 points (complete or extended functionality, only minor issues)

Code quality

To earn the full amount of bonus points for code quality, refer to the already provided comments on code quality in Lab 4 and to our style guide available via StudIP. Please read them carefully. For this lab, we will especially focus on code re-usability and an appropriate level of abstraction when grading.

Interview performance

You have roughly 5 min to present your solution. Afterwards, we will ask the whole team about your solution code. This means every team member can be asked about every line in your code. Ensure that everyone knows the whole functionality of your program. Even if you split your work into sub-parts (and we encourage you to do that), make sure everyone is able to explain the code of the other team members.

Submission

This time, you will submit your solution directly after your interview. Read the following steps carefully:

- You can present and submit your solution on or before the last day of the semester, but only in your assigned lab group. You are allowed to present and submit only **once**. We do not allow you to submit and present your reworked code again after your submission.
- The **latest dates for interview and submission** depend on your lab group:
  - Lab Group 1 and 2: **January, 29th, 2018, 11:30 - 13:00**
  - Lab Group 3 and 4: **January, 29th, 2018, 13:15 - 14:45**
  - Lab Group 5 and 6: **February, 2nd, 2018, 9:45 - 11:15**
- Prepare a USB stick with a ZIP-File named with your group name, e.g. `3A` or `6F`. This file has to contain
  - the entire project folder (of Eclipse project, not the entire workspace),
  - and a *readme.txt* containing names and matriculation numbers of all team members.
- The bonus points given to you after the interview are subject to plagiarism check. If we find two identical or highly similar submissions, both teams will loose their bonus points.

Overview, hard- and software introduction

Overview

In this exercise, you develop a Java application for a given LEGO robot to implement a 2D-plotter. The plotter must be able to draw two different shapes (a triangle and a sailing ship) in different sizes selectable via a user interface. Figure 1 shows the robot to be used. Your software runs on a LEGO NXT Intelligent Brick using the LeJOS firmware version 0.9.1 beta.

The general procedure is as follows: The user chooses one of the two predefined shapes and selects the size of the shape. The robot is placed on a given coordinate grid, c.f. Fig. 1, and after a calibration phase, the robot starts drawing at a predefined position and with the size selected by the user.

The robot

The robot has three motors: one motor drives the robot forward and backward. Another one moves the swivel arm, on which the pen is mounted, left and right. The third motor moves the pen up and down (by coiling a rope on a reel). Three sensors are installed for calibration purposes: a touch sensor allows to determine when the motor driving the swivel arm is in its rightmost position. Another touch sensor

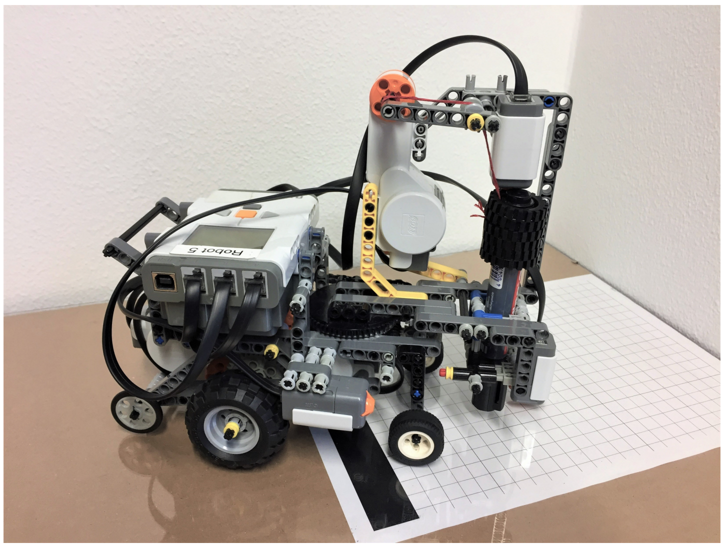


Figure 1: LEGO Mindstorms plotting robot.

detects if the pen reaches its maximum upper position. A light sensor enables the robot to find a black bar, which marks the start of the drawing area.

Thus, to properly address all motors and sensors, ensure that all parts of your robot are attached to the following ports:

Port	Functionality
SensorPort.S1	Touch sensor of swivel arm
SensorPort.S2	Touch sensor of pen
SensorPort.S3	Light sensor
Motor.A	Swivel arm left / right
Motor.B	Pen up / down
Motor.C	Wheels = Robot forward / backward

The robot may take mechanical damage when motors (especially the motor for the swivel arm) are driven beyond their boundaries, therefore:

Be careful when driving the motors for pen and arm! Always make sure that they are not driven beyond their physical boundaries!

Before you start

Before you can program the robots, you need to install the LeJOS SDK. To install LeJOS on your computer, you can follow the tutorial at http://www.lejos.org/nxt/nxj/tutorial/Preliminaries/GettingStartedWindows.htm. You will need a working Java Development Kit (JDK). We recommend you to use a 32-bit version of JDK, as there are problems with a 64-bit version. The robots will be programmed by USB, so you will need the USB driver. The Bluetooth stack is not needed.

To start LeJOS programming directly from Eclipse, the simplest way is to install the LeJOS plug-in for Eclipse. Follow the step-by-step tutorial given in

`http://www.lejos.org/nxt/nxj/tutorial/Preliminaries/UsingEclipse.htm#7`

to obtain a fully working program environment for your LEGO Mindstorm robot.

A pre-configured Eclipse project is provided to simplify LeJOS programming. Download the file *diss-lab5-template.zip* from Stud.IP. Then start Eclipse and choose the menu item *File→Import*. In the upcoming dialog, unfold *General*, select *Existing Projects into Workspace* and click *Next*.

In the next dialog, check the item *Select archive file* and click *Browse...* to select *diss-lab5-template.zip*. The project *diss-lab5* appear in the list of projects. Then click *Finish* to import the project into your workspace.

Now choose menu item *Window→Show View→Ant*. A new window titled *Ant* appears (on right hand side, near *Outline* in case you use the default layout). Drag and drop the file *build.xml* from the project *diss-lab5* into the *Ant* window. A new item called *Plotbot* should appear. Unfold *Plotbot* and double-click *link* to link your project. This generates the Nxj file in subfolder *build* of your program. If you want to upload your program to the robot and run it directly, use the *uploadandrun* item.

Tasks and required functionality

Tasks

Write a Java program that allows your robot to draw two shapes, also depicted in Fig. 2:

- A triangle
- A sailing ship

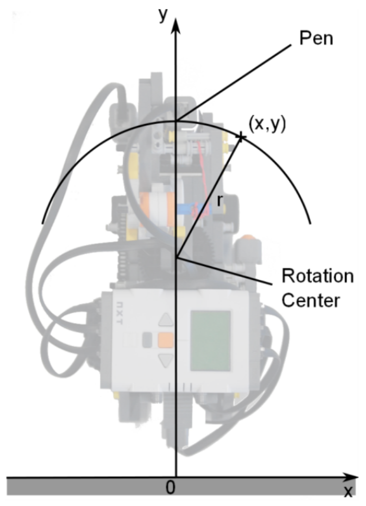
Your program has to pass through the following phases:

- User interaction (choose shape and size)
- Calibration
- Drawing the chosen shape

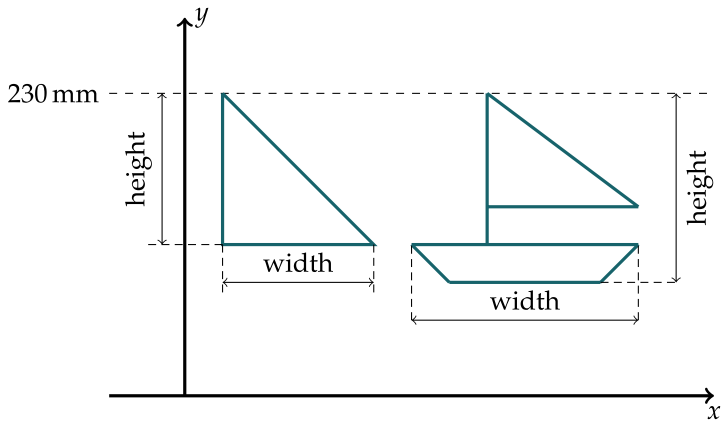
The following sections give a rough overview of the different phases, for detailed descriptions and implementation details refer to Section *Hints and details*.

User interaction

For interaction with the user of the robot you can use the LCD and the buttons of the NXT intelligent brick. Implement a user menu which allows the user to select the shape to be drawn and to input the size of this selected shape (in mm). After entering these inputs, they have to be passed to your drawing routines. Details on how select the right sizes and aspect ratios can be found later.



(a) Coordinate system



(b) Shapes

Figure 2: Coordinate systems and shapes to be drawn

Calibration

The initial position of the robot is with its driving direction perpendicular to the black bar and the swivel arm facing straight in driving direction. Before starting any drawing task, the robot needs to bring its motors in a predefined starting position. For calibration, the robot drives backwards until the light sensor detects the black bar marking the initial $y = 0$ position. Note that the grid on the drawing plate is not used for orientation while drawing.

Use the touch sensor of the swivel arm during calibration to obtain the maximum angle of the arm. You can assume that the swivel arm of the robot is in a central position when your program starts. The position of the pen is calibrated by the touch sensor above the pen.

It is **absolutely mandatory** that your robot finds out about the maximum angle the swivel arm can travel to make sure it is never driven beyond this point. The NXT motors can easily break the robot apart; it is very annoying and time consuming to repair it.

Coordinate System

Your software uses a Cartesian coordinate system to define locations in two dimensions x and y. As shown in Fig. 2, the y-direction is defined as the direction in which the robot moves forward and backward. The position $y = 0$ is located at the upper edge of the black bar (shown in gray). The x-direction is orthogonal to y and has its $x = 0$ value in the middle of the robot. Note that the x-position cannot be calibrated and thus results from the initial position of the robot. You can freely choose the initial x coordinate (e.g. only drawing on the right side of the y axis (allow only $x \geq 0$)) of your shape as long the relative dimensions (width and height) hold.

Shapes

The main task of the robot is to draw the two shapes shown in Fig. 2. One shape is a triangle, the other one is a sailing ship. Both shapes have to be drawn with the upper edge located at $y = 230\,\text{mm}$. The triangle is determined by height and width and the ship by parameter width.

For the ship, you can freely chose the height as long as width $\geq$ height holds. It is sufficient if only the width has to be entered in the user interface. You can choose an appropriate value for the height internally in your code then. It is sufficient if your robot can draw one shape after another. We do not require you to draw both shapes side by side.

Hints and details

This section provides some hints and details that help you through this lab. Please note that you are allowed to use a different design approach or alter some details. However, to get the full amount of bonus points, we expect you to have a clear structure in your code. It should be easy to reuse and extend your code. You will have to define reasonable classes and interfaces.

General Hints

- Think before you code! Identify main working steps you have to complete, e.g. menu, drawing and motor control. Those tasks are quite independent from each other, which gives you the opportunity to work in parallel.
- If you have any questions, feel free to ask them! However, always have a look at the LeJOS API Documentation first. You can find it here: `http://lejos.sourceforge.net/nxt/nxj/api/`. Use the lab time to ask questions that could not be answered by the documentation, as we cannot guarantee to have time outside the lab hours.
- If your program throws an exception, three numbers are displayed on the LCD. These inform you about the type of exception that was thrown and where it was thrown. You will see a string such as *Exception: 16 at 33:25*, where the first number is the ID of the exception class, the second is the method number, and the third is the so called PC value. In the *Ant* window in Eclipse you find a *debugtool* when opening `Plotbot`. After double clicking on *debugtool*, Eclipse will ask you for the three numbers and then displays the name of the exception class as well as the class name, method name, and line number in which the exception was thrown. Analyze this to understand what went wrong.
- To reset the NXT brick of your robot, simultaneously press the orange enter button and the dark gray clear button.

Programming constants

For your calculations you need some mechanical constants of the robot, which we provide here. Feel free to add more or adjust these constants if you need to. When using these values in your program think about a reasonable naming and define them as constants.

- Distance from arm's rotation center to pen: 80 mm
- Distance from arm's rotation center to light sensor: 105 mm

- Wheel diameter: 56 mm
- Gear ratio of arm motor: 84
- Gear ratio of wheel motor: 5
- Light sensor threshold for detecting the black bar: ~ 500

User interface

Selection menu

To implement a user interface for your program, you can use the LCD and the four buttons of the robot. LeJOS provides the class `TextMenu`, which can be used to implement a selection menu on the robot. You can use the following (incomplete) code snippet as a starting point:

```java
private static final String TITLE = "Select :";
private static final String[] ITEMS = {"Calibration","Triangle","Ship"};
private TextMenu menu = new TextMenu(ITEMS,1,TITLE);
int selection = -1;
do {
  selection = menu.select();
} while(selection < 0);
while(Button.ENTER.isDown()) {}
Plottable toDraw = null;
if (selection == 0) {
  toDraw = new Calibration();
} else if (selection == 1) {
  toDraw = new Triangle();
} else if (selection == 2) {
  toDraw = new Ship();
}
```

Input number menu

In addition to the `TextMenu` provided by LeJOS, you also need a menu to input the desired size of the shape. As this is not provided by LeJOS you have to implement it on your own. However, you can use `Button`, `ButtonListener` and `LCD` provided by LeJOS to read out the state of the buttons and to print variables to the LCD. Use these provided classes to implement a menu that can be used to input numbers for determining the size of the shape.

Conceptual design

Coordinates

Before you start drawing a shape, you should think about how you can abstract the motor angle at the real robot into an xy-coordinate system that is used by your program. It is a good idea to implement a

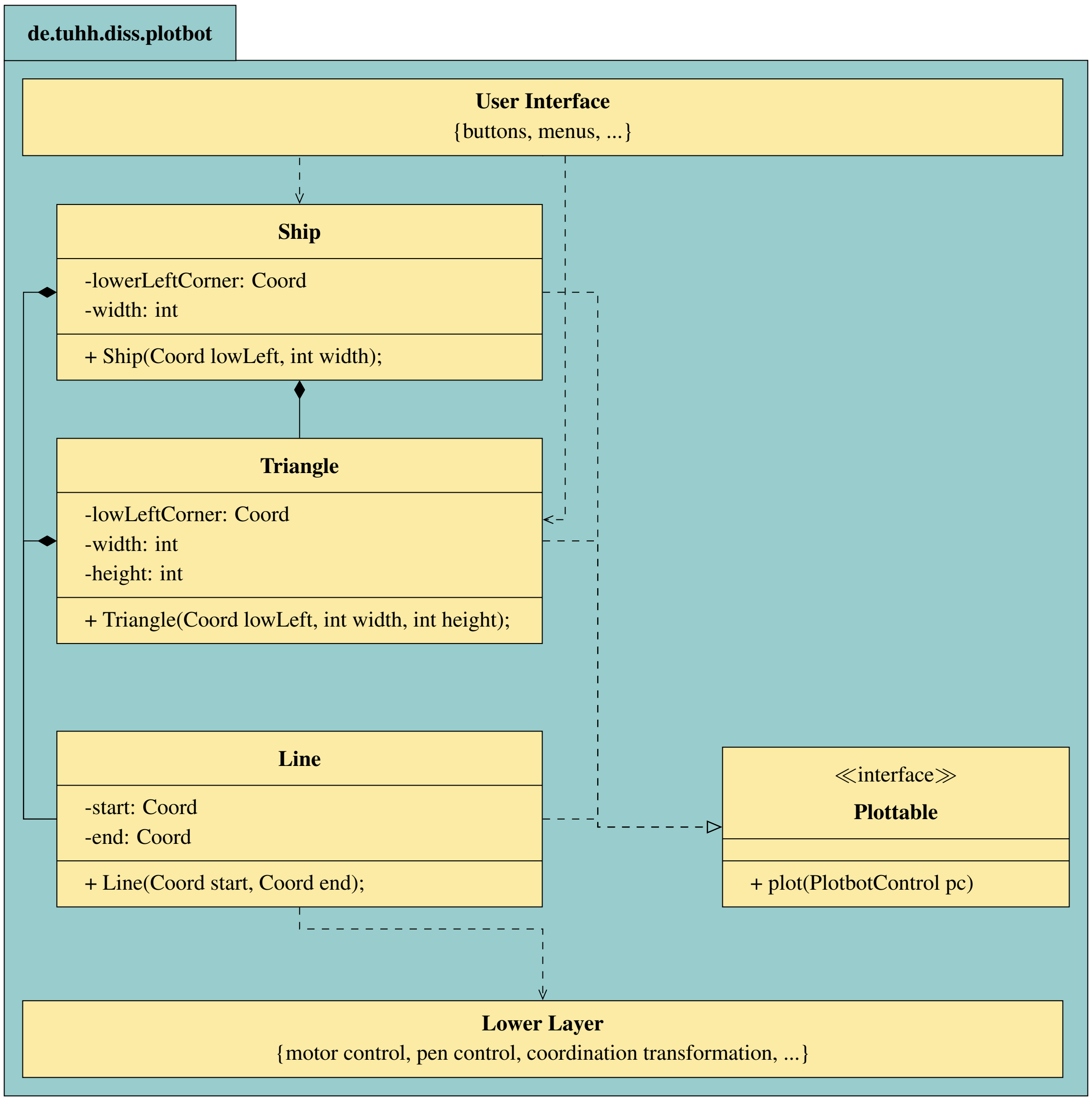


Figure 3: A suggestion on how to design your classes.

class `Coord` that represents a coordinate in your abstract xy-coordinate system. You should then write two methods `angleToXy` and `xyToAngle` that translate motor angles to xy-coordinates and vice versa.

For transformation you should use the provided constants and basic trigonometry. The needed functions are implemented in Java's `Math` library. Note that all trigonometric `Math`-functions in Java operate on radians, in contrast to the motors, which operate on degrees. Thus, a conversion is needed. You should also note that for calculation of x-coordinates only the angle of the arm motor has to be taken into account, whilst, for calculation of y-coordinates motor angles of arm and wheels are needed.

Lines

After you have abstracted the real-world robot motor movements to an xy-coordinate-system you should implement a class that is able to draw arbitrary straight lines. As lines are fully defined by a start and an end point, we suggest to write a class `Line` with members `start` and `end` of type `Coord` with a constructor `Line(Coord start, Coord end)`. Instances of this class can then be used to draw lines. As the robot can only move in y-direction, it is quite easy to draw vertical lines, but hard to draw horizontal or even diagonal lines. Its a good approach to split the line in several discrete drawing steps, which will be drawn one after another.

In the real-world robot, you will likely face difficulties drawing straight horizontal lines due to the slack of the robot arm. As there is some clearance between the gears, the arm has to be moved slightly further when the direction has changed. It is possible to compensate this in your code. You are welcome to tackle this issue, but you should create a working solution first.

Shapes

After you have successfully implemented the drawing of lines you can use these lines to draw more complex shapes. For example, the triangle is defined by just three points which you connect by the previously created lines. Another way of defining the triangle is by providing a coordinate of the lower left corner as well width and height.

Finally, compose a class `Ship` from the previously created shapes, lines and triangles. The constructor of this class should be called with `Ship(Coord lowLeft,int width)`. The ship shown in Fig. 2(b) is then drawn, when the method `plot()` of an `Ship` object is called.