Lab: Wall Following

Submission instructions: Students are expected to work in their assigned lab groups. The lab consists of three components: demonstration, lab report, and code submission. Instructions on the demonstration can be found in this handout. Lab reports and code submission must follow the guidelines established in this handout and for the course. For more information, see the ECSE211SubmissionInstructions.pdf on MyCourses.

Design objectives

1. Design and implement a **wall** following system that can navigate around a sequence of blocks and obstacles that form a **wall** containing gaps, concave corners, and convex corners.
2. Implement Bang-Bang and P-type controllers as part of the **wall** following system.
3. Evaluate the design and compare the behavior of the two controllers.

Design requirements

The following design requirements must be met by your **robot**:

- **Robot**
  - Must be able to complete a lap around a series of blocks while maintaining a fixed distance from the **wall**, i.e. a bandcenter.
  - Must account for right-angle convex corners.
  - Must account for right-angle concave corners.
  - Must account for gaps in the **wall**.
  - *Note: gaps will be no larger than the shortest width of a block, i.e. 10 cm.*
  - Must be able to drive around an arbitrarily shaped **wall**, *made up of at most 7 blocks*.

- Controllers and sensors
  - Must implement a Bang-Bang controller.
  - Must implement a P-type controller.
  - Must use the ultrasonic sensor for range finding.
  - Must be able to perform the course using each controller type.

Demonstration (30 points)

The design must satisfy the requirements by completing the demonstration outlined below.

Design presentation (10 points)

Before demoing the design, your group will be asked some questions for less than 5 minutes. You will present your design and answer questions to test your individual understanding of the lab concepts. Each person will be graded individually.

You must also present your workflow, an overview of the hardware design, and an overview of the software functionality. Visualizing software with graphics, such as flow charts, is valuable.

Bang-bang controller (10 points)

- Place the **robot** at one corner of a sequence of blocks as shown in Figure 1.
- Start the **robot** with the Bang-Bang controller.
- The **robot** must follow the **wall** continuously for 2 consecutive laps.
- The **robot** cannot touch the **wall** at any point, including its wires.

P-type controller demo (10 points)

- Place the **robot** at one corner of a sequence of blocks as shown in Figure 1.
- Start the **robot** with the P-type controller.
- The **robot** must follow the **wall** continuously for 2 consecutive laps.
- The **robot** cannot touch the **wall** at any point, including its wires

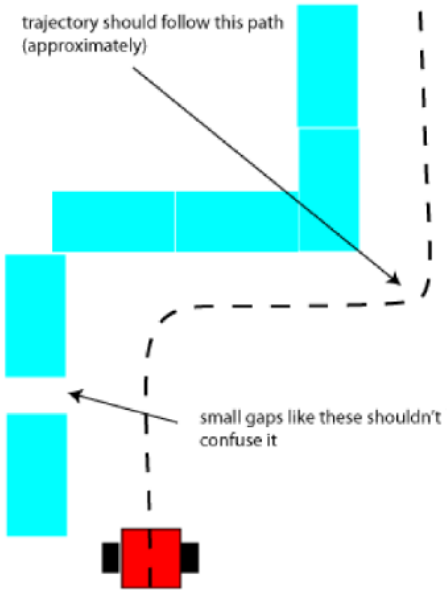

Figure 1. Course and robot placement.

Provided materials

Physical material

In the lab, blocks that will be used to build the **wall** are provided. The blocks will be placed per the setup specified in the requirements by a TA.

Sample code

A package of sample code is provided which contains the following:

- BangBangController.java
  - A skeleton class for building the Bang-Bang controller.
  - Drives the **robot** and takes in data from the ultrasonic sensor.

- Main.java
  - The main class that runs the **robot**.
  - Creates and run the ultrasonic polling thread.

- PController.java
  - A skeleton class for building the P-type controller.
  - Drives the **robot** and takes in data from the ultrasonic sensor.

- Printer.java
  - Provides a menu to select the controller types.
  - Outputs information to the LCD screen.
  - Runs in a thread.

- Resources.java
  - Defines the static constants used in the program.
  - Defines the ports used by motors and sensors.

- UltrasonicController.java
  - Provides an interface for the controllers to access the ultrasonic data.

- UltrasonicPoller.java
  - Polls the ultrasonic sensor. Runs in a thread.
  - Provides information through the UltrasonicController interface.

Implementation instructions

1. Using the BangBangController.java as a base, design and implement a Bang-Bang controller.
2. Using the PController.java as a base, design and implement a P-type controller.

Report Requirements

The following sections must be included in your report. For this lab, you will not collect any quantitative results; *all data is qualitative*. Answer all questions in the lab report and copy them into your report. For more information, refer to ECSE211SubmissionInstructions.pdf. **Always provide justifications and explanations for all your answers.**

Section 1: Design Evaluation

You should concisely explain the overall design of your software and hardware. You must present your workflow, an overview of the hardware design, and an overview of the software functionality. You must briefly talk about your design choices before arriving at your final design. Visualizing hardware and software with graphics (i.e. flowcharts, class diagrams) must be shown. Make sure to mention how you arrived at tuning your Bang-Bang controller and P-type controller constants (i.e. bandcenter, bandwidth, P-type constant).

Section 2: Test Data

This section describes what data must be collected to evaluate your design requirements. Collect the data using the methodology described below and present it in your report.

Testing the P-type controller constant

1. Choose 2 values above and below your P-type controller constant used in the demo.
2. Run the **robot** using the **P-type controller**.
3. Note its performance, i.e. band center and oscillation behavior, for the 2 cases.

Bang-Bang controller test (*3 independent trials*)

1. Place the **robot** at the starting corner of a **wall**.
2. Ensure the **wall** contains convex corners, concave corners, and gaps.
3. Run the **robot** using the **Bang-Bang controller**.
4. Check if it completes a lap without touching the **wall**.
5. Note its performance, i.e. band center and oscillation behavior for each trial.

P-type controller test (*3 independent trials*)

1. Place the **robot** at the starting corner of a **wall**.
2. Ensure the **wall** contains convex corners, concave corners, and gaps.
3. Run the **robot** using the **P-type controller**.
4. Check if it completes a lap without touching the **wall**.

5. Note its performance, i.e. band center and oscillation behavior for each trial.

Section 3: Test Analysis

Compare the performance of both controllers. Make sure to refer to your test data.

- What happens when your **P-type** constant is different from the one used in the demo?
- How much does your robot oscillate around the band center?
- Did it ever exceed the bandwidth? If so, by how much?
- Describe how this occurs qualitatively for each controller.

Section 4: Observations and Conclusions

- Based on your analysis, which controller would you use and why?
- Does the ultrasonic sensor produce false positives (detection of non-existent objects) and/or false negatives (failure to detect objects)? How frequent were they? Were they filtered?

Section 5: Further Improvements

- What software improvements could you make to address the ultrasonic sensor errors? Give 3 examples.
- What hardware improvements could you make to improve the controller performance? Give 3 examples.
- What other controller types could be used in place of the **Bang-Bang** or **P-type**?

Frequently Asked Questions (FAQ)

1. **What is a "lap"?**
   A single **lap** is defined as a closed trajectory that the **robot** must follow and keep its distance from the **wall** before arriving in its starting location. In **Figure 1**, one whole lap would mean following the **wall** on the right side (the dotted line) and continuing around the end to follow it along the left side until reaching the starting point.

2. **What is meant by "design presentation"?**
   Before a lab demo, you and your partner will briefly present your design. This can include a basic visualization of how your code functions, such as a flow chart. You will then be asked a series of questions. These could be related to the lab tutorial, the initial lab code. For this part, a grade of 10 signifies full understanding, 5 signifies satisfactory understanding, while 0 shows no understanding at all. Note that memorized answers are discouraged and both partners should be responsible for understanding the design and their associated code, even if one of them did not write all of it.

3. **Are partial points awarded in this lab demo?**
   No partial points are awarded. Possible demo points: {0, 5, 10, 15, 20, 25, 30}.

4. **Should my robot follow the wall in an anti-clockwise direction (like Figure 1)?**
   No, it may follow any direction as long as it remains consistent throughout the demo.

5. **Are reverse wheel rotations allowed?**
   Yes, you may reverse your **robot** at any point in the demo.

6. **Is there a time limit on the demo?**
   No, but your TA can conclude your demo if your **robot** is moving extremely slow.

7. **Will the wall setup look exactly like Figure 1?**
   No, it could be different. However, all setups will include at least one gap, one convex corner and one concave corner.

8. **Can I make additional Java classes?**
   Yes, you can add any classes as necessary.

9. **Can the speed of each motor be varied?**
   Yes, the speed of each motor can be changed, provided it continues to implement the correct controller type.

10. Why does the EV3 brick keep crashing after running our code?

Common problems include:
(1) no sensors or motors connected,
(2) motors and sensor ports do not match the ones used in your code,
(3) project is not an leJOS EV3 project.

11. Why does my Java code not upload or run on the EV3 brick?

Common problems include:
(1) no IP address selected from *Preferences → leJOS EV3 → Connect to named brick* [Solution: set the same IP address as shown on your EV3 brick]
(2) not using the correct JRE which is JavaSE-1.7 [Solution: right-click on your project → *Properties → Java Compiler →* is 1.7 selected? If not, create a new project]
(3) not using a leJOS project in the eclipse IDE [Solution: create a new *leJOS EV3 Project*],
(4) checking the box under *Preferences → leJOS EV3 → Use ssh and scp* [Solution: uncheck that box]
(5) bad USB connectivity [Solution: try using other USB ports and uploading your code multiple times].