Lab 3
See, sing, follow, reckon

Table of Contents

Aim

The goal of the lab is to familiarize you with the different types of loops and control statements offered by the C language. This lab also introduces you to the 3π+'s reflectance sensors and timers.

Submission Deadline

You have the periods of 17 and 24 September 2025 to work on the laboratory. The lab period of 1 October will be used for midterm 1. You must submit your laboratory work before 23:59 on Tuesday, 7 October 2025; see the ***Submission requirements*** section at the end of this document for the required contents. You must perform all demonstrations by 8 October at the beginning of the lab period, at the latest.

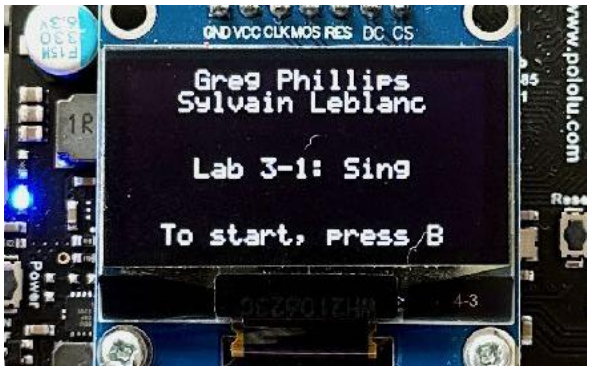
General requirements

Each of the four lab parts described below will need to be in a separate CLion project.

You should aim to have Labs 3-1 (See) and 3-2 (Sing) completed by the start of the second week of the lab.

Any sound your robot makes must be brief and quiet.

When run, each program must display an introductory screen similar to figure 1, then wait for the user to press and release button B before starting. The introductory screen must include your lab group members' names, the lab title, and "To start, press B", as shown in Figure 1.



```
Greg Phillips
Sylvain Leblanc

Lab 3-1: Sing

To start, press B
```

Figure 1: Example introductory screen.

Understanding the 3π+ line sensors

The 3π+ has five downward-facing reflectance sensors that can be used for line following. These are described in section 5.5 of the Pololu 3π+ 32U4 User's Guide, and the sensor API is in the `LineSensors` section of the 3π+ API documentation.

There are several ways to access the values read by the sensors. For labs 3-1 and 3-2 **you must use** the `LineSensors`' `read` member function, which gives raw sensor readings.

For labs 3-3 and 3-4 you may use `read` if you wish, but you will likely achieve better results with `readCalibrated`. Calibrating the line sensors is explained in lab 3-3.

You may *not* use `readBlackLine` or `readWhiteLine`.

When you call `read` or `readCalibrated` you will need to pass it an array of five `uint16_t`. By default, the 3π+ IR emitters will be on during the read, which provides more reliable readings and is what you want. For example:

```
// ...
LineSensors lineSensors;
// ...
void loop() {
    static uint16_t readings[5];
    // ...
    lineSensors.read(readings);
    // ...
}
```

Note that `readings` is declared as **`static`**. This means that its values will be retained between successive calls to `loop` rather than a fresh array being created each time `loop` is called.

After calling `read` or `readCalibrated`, the result values will be in the `readings` array.

Lab 3-1: See

Write a program called `see.cpp` that works as follows:

> Each time the user presses button B, the program takes a single raw reading using the `LineSensors`' `read` member function and displays the value of the centre reflectance sensor on the screen.

Use the 3π+ with your program to measure the value detected by the sensors for the ten squares of the greyscale sheet. Greyscale sheets will be provided in the lab, but you can print your own copy from the PDF provided with lab 3 on Moodle.

Make a graph of the values with the grey density (0-100) on the abscissa (x-axis) and the measured reflectance on the ordinate (y-axis). You can use the program of your choice to draw the graph. You must submit the graph with your code.

When your program works and you have completed the graph, call the instructor to demonstrate your program's functionality.

Lab 3-2: Sing

Write a program called `sing.cpp` that causes the 3π+ to emit a note based on the grey level that is detected by the central reflectance sensor (measured as described in lab 3-1), higher pitched for lighter grey, lower pitched for darker grey. The rating must be updated every 30 milliseconds and there must be 10 notes in the range. The measured reflectance must also be displayed and continuously updated on the screen.

When your program works, call the instructor to demonstrate its functionality.

Lab 3-3: Follow

Write a program called `follow.cpp` that makes the robot follow a line on the floor. The line will be black electrical tape 17-19mm wide and you can assume the floor will be light-coloured (like the tiles in the lab).

For labs 3-3 and 3-4 you may either use `readCalibrated` or use the values you determined in lab 3-1. Use of `readCalibrated` will make your control system significantly more robust, and is strongly recommended. **You may *not* use** `readBlackLine` or `readWhiteLine`.

Setting timeout

See the 3π+ API documentation for the `LineSensors setTimeout` member function. The shorter the timeout, the more rapidly your sensors will take a reading and the better your robot's ability to track the line – however, if the timeout is too short, non-black parts of the floor may be sensed as black. Choose wisely.

Line sensor calibration

If you intend to use `readCalibrated` your robot must first calibrate the sensors. The 3π+ `LineSensors` API provides a `calibrate` member function for this purpose. Each time `calibrate` is called, it reads the current value of each sensor and compares the reading to the lightest value that sensor has previously measured with `calibrate` during this program execution, and the darkest value it has measured. If the new reader is lighter than the previous lightest, or darker than the previous darkest, that values are updated.

So, a reasonable strategy for calibration would be to have the robot rotate left and right such that each sensor passes over both the black line and the light floor, calling `calibrate` repeatedly.

See `LineSensors`' `calibrate` member function in the 3π+ API documentation.

Starting condition

You can assume that your robot will be placed so that the central sensor is on top of the black line and facing the direction it initially needs to go, as shown in figure 2. The floor to the left and right of the tape will be light coloured.

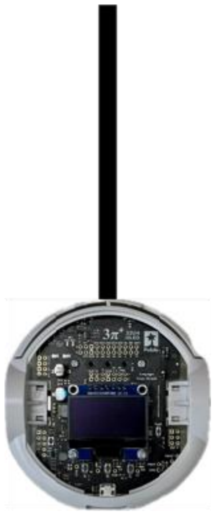


Figure 2. Starting position.

Operation

1. If you intend to use `readCalibrated`, (strongly recommended), your robot must first perform a sensor calibration as described above, then return to the starting position. During this process, the display must show “Calibrating sensors”.
2. When calibration is finished (or immediately if you are not calibrating) the robot should begin to drive in the direction it was originally facing.
3. Your program must then move the 3π+ forward along the line at a moderate speed, steering as necessary to stay on the line. ***You must use only the three central sensors for this task.***
4. Your program must allow the robot to follow straight lines and curved lines where the curve may be as tight as an 8 cm radius (even tighter if possible).

5. If your robot comes to a T-intersection (see figure 3), it must make a sound (beep), turn around, and follow the line back in the direction it came from.

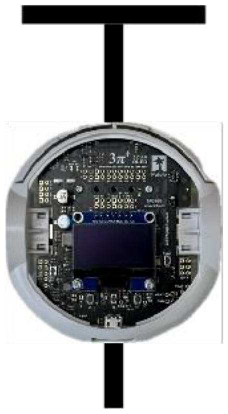


Figure 3. T-junction.

6. If your robot goes off the line completely, it must make a sound (beep), display an appropriate message, and stop.

When your program works, call the instructor to demonstrate its functionality.

Lab 3-4: Reckon

Write a program called `reckon.cpp` that will drive the robot forward a user-selected distance, using information gathered in a distance calibration step to determine when to stop.

1. If using `readCalibrated`, perform a sensor calibration as in step 1 of lab 3-3.
2. Display "Ready to measure, press B" and wait for a press of button B.
3. Place the robot on the guide line, a few centimetres behind the distance calibration start line as illustrated in figure 4. All lines will be black tape and the lead edges of the calibration lines will be 120 cm apart. The space between the calibration lines and the guide line will be sufficient that the calibration lines will be visible to the extreme left and right sensors, but not to the middle three sensors, assuming your robot is centred on the guide line.

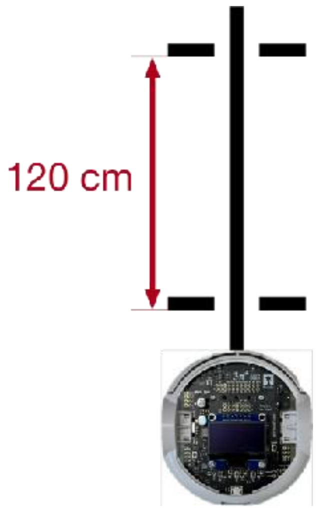


Figure 4. Distance calibration track (not to scale).

4. When the user presses and releases button B, the robot must move forward at a moderate speed following the guide line. You can reuse the line-following code from your lab 3-3.
   4.1. When the robot gets to the second calibration line, it must stop.
   4.2. Your robot must measure the shaft encoder count between reaching the lead edges of the two lines. This gives you the encoder count to travel 120 cm. See `Encoders` in the 3π+ API documentation.

5. The robot must display “Ready” on the first line of the display and

   <   Go   > on the bottom line. The middle line of the display must initially show 10. Pressing button C repeatedly must cycle this display through the values 10, 30, 60, 100 and 200. Pressing button A must do the same, in the opposite order.
6. When the user presses button B, the robot must display “Driving” on the screen, move forward in an approximately straight line the selected distance in centimetres (not line following), stop, and beep, after which it must return to step 5. It must determine how far to drive using the information gathered in step 4.

When your program works, call the instructor to demonstrate its functionality.

Questions

Question 1. Imagine we tell you that we wrote a program for the 3π+ that does the following:

1. The line detector is calibrated.
2. The robot receives the command to move forward.
3. The code enters a loop that waits for the program to detect a dark surface before stopping.

Explain in your own words what problem(s) exist with this program.

Question 2. Explain how you chose an appropriate value for the `setTimeout` function, as described in lab 3-3 “Setting Timeout”.

Submission requirements

You must submit a single, well-formatted PDF document with a document title, your lab group members ’names, and a date, containing:

- your answers to the questions,
- the graph you created in Lab 3-1, and
- copies of your source code for all four programs. See the lab 2 instructions for tips on producing PDF copies of your code, and remember your code must be in dark letters on a white background.

You must also submit the source code `.cpp` files for your four programs, named as specified in each part. Proper documentation and coding practices are mandatory and worth marks! Check the course’s ***C Coding Style*** page for the requirements.

Submit the five files (one `.pdf`, four `.cpp`) as individual files (not zipped) to the Lab 3 page on Moodle.