SWE4211 Lab 8: Sensor Usage

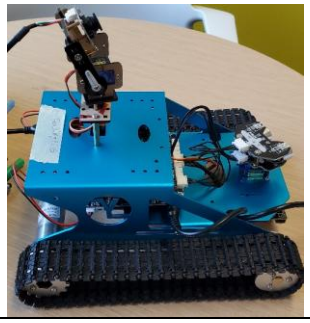
1 Introduction

Last week, we started controlling our robot. Now we want to add some features to it and measure some performance characteristics for the robot. We will continue with multiple queues, but now we will have sensors controlling the operation of our robot.

2 Lab Objectives

- Integrate multiple queues into a control system.
- Use input sensors to manage a robotic system.
- Add threads to an existing system to extend basic functionality.
- Measure CPU utilization.
- Measure robot performance characteristics

3 Laboratory Equipment Needed

1 stopwatch 	Yahboom Robot for Raspberry Pi Smart Car AI Starter Kit 
Robert Charger 	1 Screwdriver 
	1 Micro SD Card with Rasperbby PI OS Installed 

4 Deliverables / Submission

When completed, commit the final code and push it to the repository. In addition, submit a short video which demonstrates the following

1. The robot will stop when driving from a white surface and encountering a black line
2. The robot can generally follow a curving line as it drives, generally staying on the line.
3. Pressing the M key and enter key on the robot console mutes the horn if it is sounding.

Finally, submit one report (also in Canvas) with the following contents:

1. Introduction
   a. What are you trying to accomplish with this lab? This section shall be written IN YOUR OWN WORDS. DO NOT copy directly from the assignment. This section should be several grammatically correct sentences.
2. Discussion
   a. Include an expanded block diagram, showing the inputs and outputs the system now has.
3. Screen captures
   a. Include a screen capture of your software building without warnings. To do this, do a "make clean" and then a "make all" in the Eclipse environment or on the console.
4. Measurements
   a. Record the data related to time and stopping distance versus the control variable.
   b. Record in a table the calculated values for speed and acceleration of the robot versus the control variable. Be sure to show how you calculated the values, as the formulas will need to be algebraically manipulated into the correct structure to give you an appropriate answer.
   c. Include the 4 plots generated, labeling as is appropriate.
   d. Record the cpu usage with HTOP and the P command for line sensing both enabled and disabled. Be sure to reset if line following was enabled before disabling it.
5. Discussion
   a. Of the 4 items you are looking at related to mechanical performance, are any of them close to constant across the PWM duty cycles as measured or calculated? Based upon the physical system, should any of them be close to constant? Note that with experimental data, a constant likely will not be exactly constant due to experimental variance.
   b. For this experiment, where is the largest source of error in the data collected?
   c. Do the percentages make sense for the line sensing task when line sensing is enabled and disabled?
   d. Why are the percentages different between htop and the P command?
   e. Given what you know about the design of the system and the task rates being employed, what is the best and worst-case latency that your robot might encounter as it goes over the line and decides to stop? Based on the highest speed recorded, what difference in stopping distance MIGHT THIS CAUSE?
6. Things gone right / Things gone wrong
   a. This section shall discuss the things which went correctly with this lab. Include key things that came simply in completing this lab and experiences from other labs which helped to complete this exercise.
   b. This section shall also discuss the problems that occurred completing this lab. Include issues with C++ programming that occurred, issues with networking, or other problems encountered when successfully completing the lab.
7. Conclusions
   a. What have you learned with this experience? What improvements can be made in this experience in the future?

This material should be submitted as a single pdf file. If you have any questions, consult your instructor.

5 Key New Features

5.1 Sensing Lines

The robot is equipped with a set of line sensors mounted on its underside, as shown in Figure 1. These sensors help the robot follow a path by detecting the contrast between a black line and a white background. Each sensor uses an infrared LED to emit light toward the surface. When the surface is white, the light reflects and is detected by the sensor; when the surface is black, the light is absorbed and not reflected, as illustrated in Figure 2.

Each sensor outputs 1 when it detects black and 0 when it detects white. The outer sensors (far left and far right) indicate if the robot is drifting off the line: if the left sensor reads 1, the robot steers left; if the right sensor reads 1, it steers right. The middle sensors help maintain alignment by signaling when the robot is centered over the line. For example, a reading of 0100 means the line is slightly left of center, so the robot should adjust left, while 0110 indicates the robot is perfectly centered.



Figure 1 Yahboom line sensors

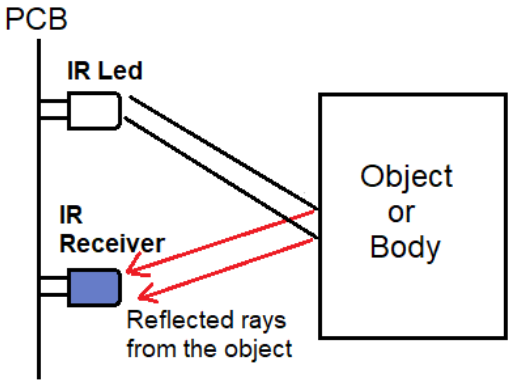


Figure 2 IR Sensor Operation (https://components101.com/ir-led-pinout-datasheet)

6 Designing our software

6.1 Line Detection

The first task is responsible for reading the four-line sensors and calculating a line score. This score can be positive or negative, with its magnitude indicating how far the robot is from being centered on the line. To improve reliability, the task stores sensor readings in an array for filtering, reducing the impact of erroneous data.

The second task runs at a slower rate and uses the filtered data from the first task to execute the steering algorithm. This ensures smoother and more accurate adjustments.

The line detection unit itself is implemented as a periodic task with its own queue and can be enabled or disabled via network messages. If the system is enabled and all four sensors detect black, the unit will issue a single stop command to the robot controller.

Table 2 shows the network commands that can be received over the network.

Table 2: Commands and actions for line following.

Incoming Command from the network	Hex Value	Description
START_LINE_SENSING_CMD	0x02000051	This command will cause the robot to start line sensing. When line sensing, the robot will read the line values, stopping if all sensors are black. Line sensing must be enabled to have line following performed.
STOP_LINE_SENSING_CMD	0x02000052	This command will stop line sensing. When doing this, it will also effectively stop line following as well.
ENABLE_LINE_FOLLOWING_CMD	0x02000055	This command will cause the robot to start to try and follow the line. For this to be active, Line Sensing must also be enabled.
DISABLE_LINE_FOLLOWING_CMD	0x02000056	This command will cause the robot to stop using line following.

6.2 Overall Design

The overall design for the System is shown in Figure 3. In the repository, many files have been provided to you, as is shown in Green. As with previous labs, the files you have written before which are unchanged are in blue. Those which require small changes are yellow. Those which are new are pink.

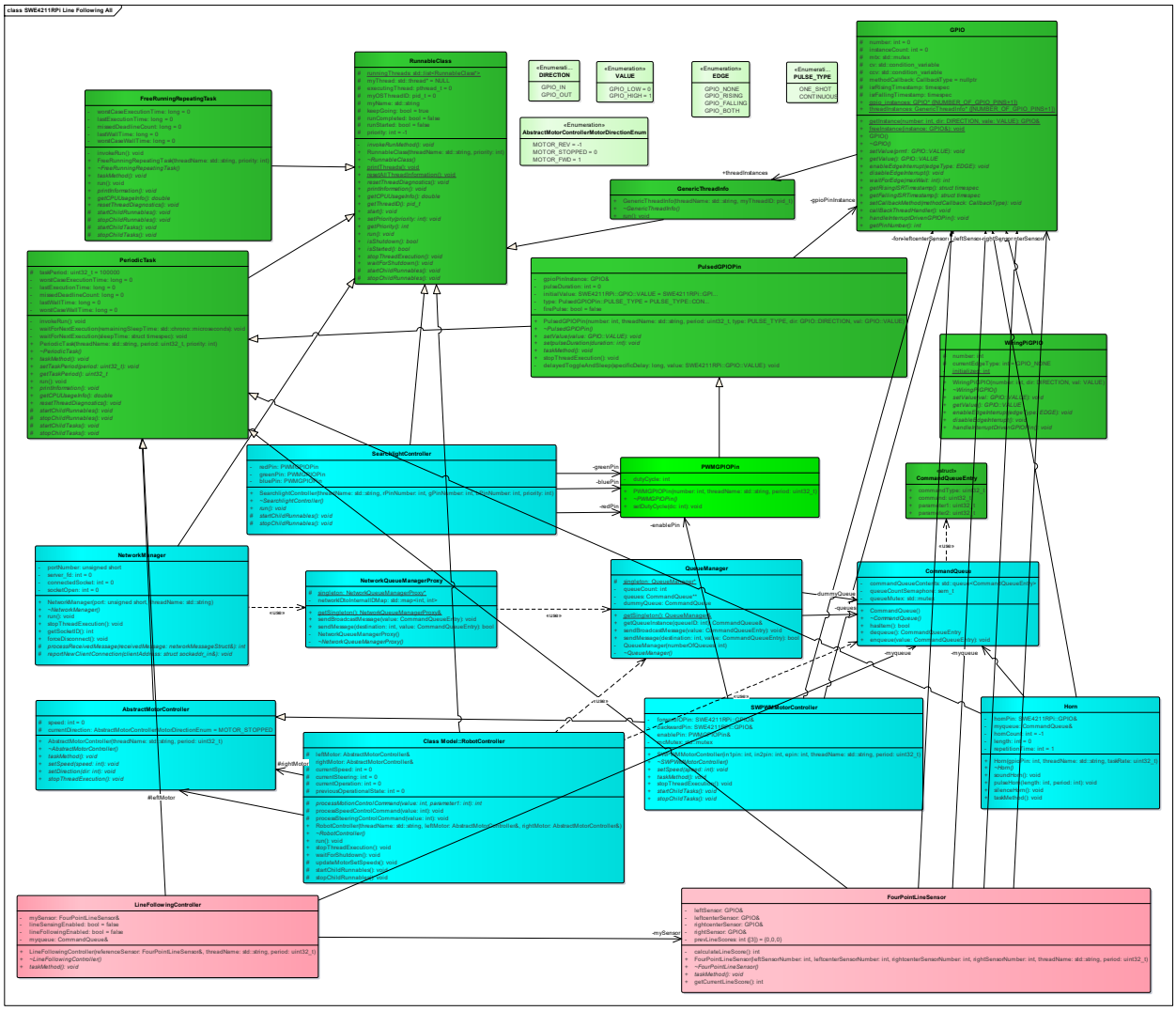


Figure 3 UML Design for the project. Note the pink represents completely new classes in this exercise, blue represents classes you have written previously and should not have to revise, yellow represents classes you wrote previously which will need modification

7 Recommended Implementation Sequence

For this lab, the default does not include line following in the configuration. It is recommended that you implement empty placeholder classes that will compile and then enable the feature for debugging. From there, one partner can work online sensing while the other works on the control logic.

8 Testing and Experimentation

With the robot now a bit more advanced, we want to do some performance analysis on the system. Once you have finished all the code, we will try to more accurately measure our robot performance as well as more accurately measure stopping distances and deceleration.

8.1 Screen Diagnostics

One of the things we want to start paying attention to is the software performance of our system. To do this, there is a set of diagnostics that is available to you in the console. When the program is running, if you press P, it will print out a set of diagnostics describing the priorities and executions of each task.
A sample diagnostic report is show in Figure 10, though your output may be slightly different. As the term continues, we will use this screen for different purposes.

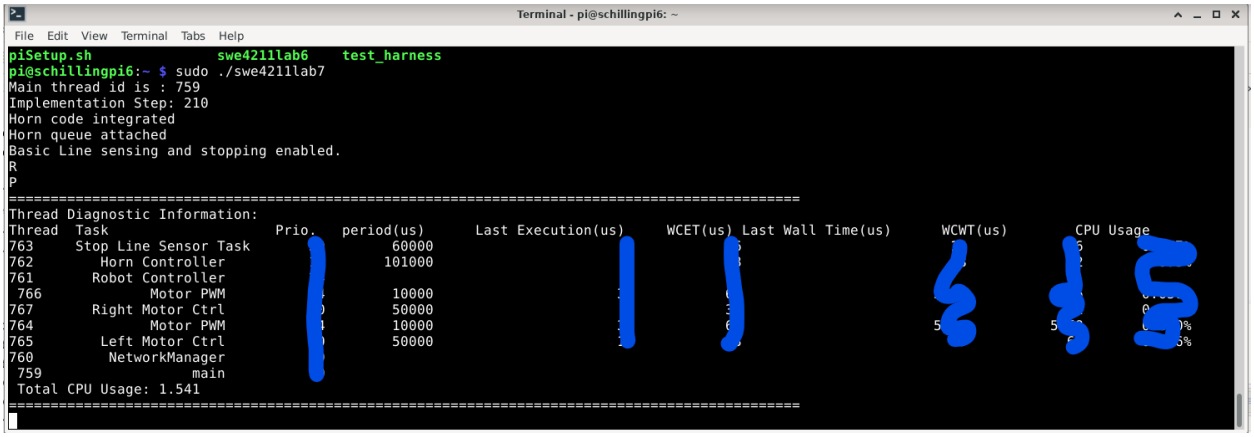


Figure 4 A diagnostics report and diagnostics reset. The top of the screen shows the initial diagnostics output. For each task in the system, the thread id is printed out in the first column, followed by a name and the priority of the thread. For periodic tasks, the task period (TP) is given. The next column (LET) gives the duration of the task in CPU time the last time the task was executed. The next column, WCET, gives the worst-case CPU execution time for the task. CPU usage for the task would be WCET / TP. The last two columns give the wall clock time for the last execution and the worst-case wall clock time. The wall clock time must be less than the task period, as if it is not, then a deadline has been missed. Then we have the CPU usage here.

8.2 Tuning the Line Sensors

On the top side of the line sensor device, you will find 4 potentiometers (i.e. adjustments). These adjust the sensitivity of the line sensors. When the line sensor is over white, the light will be off. When it is over black, it should be on. To adjust them, set the robot down on a while surface and turn the adjustment until the light comes on. Then back off the adjustment roughly ¼ of a turn, turning the light back off. Repeat for all 4 sensors. Then place the robot on a black surface and

verify that the lights turn on. Note that these sensors really need a true white surface to work most reliably.

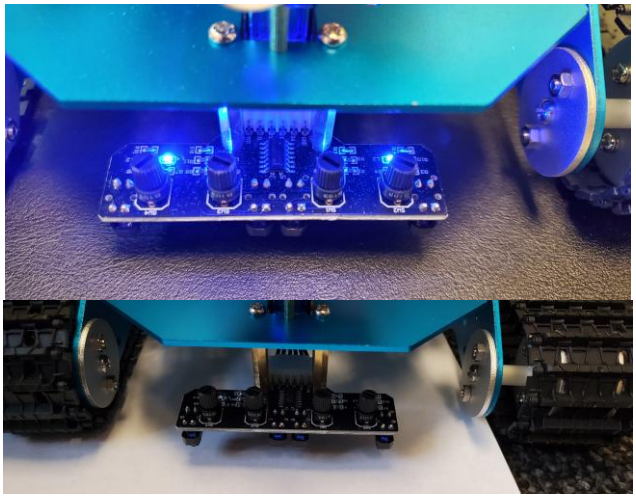


Figure 5 Adjusting the line sensors. Turn the knobs to adjust sensitivity.

8.3 Stopping Distance Measurement

In lab, there is a setup to automatically test stopping distance. With the robot on the white track, set it behind the start line and set the PWM to a slow speed of 26%. Enable auto stopping. Start the robot driving forward. Time how long it takes for the robot to go from start to finish. The distance is 2.5 meters. Also record the stopping distance beyond the stop line. Repeat at duty cycles of 50%, 76%, and 100%.

From this recorded data, calculate the speed of the robot and the deceleration that the robot is capable of. The speed is calculated as distance over time, and acceleration is calculated by using the two formulas given here. (Hint: You'll need to do some algebraic manipulation to get the formulas into the right forms.)

$$\mathbf{s} = \frac{1}{2}\mathbf{a}t^2 + \mathbf{v_o}t + \mathbf{s_0}$$

$$V_f = V_0 + at$$

Plot the 2.5-meter run time, the runoff distance, speed, and acceleration for the robot on different graphs versus the PWM settings.

8.4 Line following

You will also want to check that your robot can follow a line. Note that the robot may or may not be capable of following the line at all PWM values. As it is designed right now, there may

be issues at higher PWM settings, which is not unexpected. There is a track or two in the lab that you can use for testing.

8.5 CPU Usage Comparison

We also want to look at CPU usage and line tracking. Make certain that line tracking is disabled. Then, start htop and look at the CPU usage. In the Unix shell, type R to reset the diagnostics and then P to print out to the console the diagnostic screen. Record what you see for the line sensing thread for both htop and the P output. Then enable line sensing. Wait about 15 to 20 seconds and then repeat the data recording by processing the P key and looking at the line sensing thread. Is the overall line sensing feature requiring much CPU usage to operate as it is setup right now?