Lab 3: Light Sensor

Remember back in Lab 1 when the Tribot was commanded to drive in a specific pattern that had the shape of a bow tie? Specific commands were passed to the motors to command how the Tribot should move. However, what would happen if the desired path were changed? All of the Move blocks in the program would have to be adjusted to account for the new path. This can become quite cumbersome especially if the path changes frequently.

What if the robot was designed so that it traced a line instead of following specific commands? The robot would still be able to follow the path that is laid out by the black line, and the code would not have to be changed whenever the path changes. This is an example of a situation where a feedback system can be quite useful. This lab will demonstrate the practical use of a simple feedback control system.

Lab Summary

A. Use the Lego Mindstorm NXT software to understand how the light sensor works
B. Create a program that follows a line using digital sensor feedback
C. Improve the line following program using analog sensor feedback
D. Challenges

Please note that the Test Pad is required for this lab. Please have this laid out on a surface where it is out of harms way.

Part A: Understanding the light sensor

(Skip to step 11 if you are already familiar with connecting to the Tribot)

1. Open the Mindstorm NXT software on your computer
2. Switch to your saved profile
3. Go to File >> New to create a new project
4. Press the orange, square Enter button on the Mindstorm NXT Tribot to turn the robot on
5. Using a USB Cable, connect the Tribot to a USB port on your computer
6. Detect the Robot by clicking on the NXT Window button on the lower right corner of the screen as shown here:

![NXT Window Button](image)

7. A connection dialog window will open. Click on the Scan button to detect the robot that is connected to the PC. The name of your robot should appear in the list on the left side of the dialog box.
8. Click on your robot, and then click Connect.
9. Close the dialog box, and navigate to the Complete palette found at the bottom of the palette bar on the left side of the NXT interface shown in the following image:

![Diagram of NXT interface with Light Sensor block highlighted]

10. Place a **Light Sensor** block found in the Sensor palette

11. Click on the Light Sensor block. The following configuration section should appear at the bottom of the NXT software interface:

![Diagram of Light Sensor configuration section]

**Questions to Consider:**

a) What is the average reading of the light sensor when it is detecting a white shade?
b) What's the reading for a black shade?
c) How do the readings change when the Generate Light option is turned off?
d) Based on your findings, what does this sensor specifically detect?
e) With the Generate Light option turned on, try testing the light sensor on the different colors displayed on the Test Pad. Which colors produce some of the highest readings?
f) Do you notice a pattern in these readings? What is it?
g) Are there any colors that are completely different that produce the same reading?
**Part B: Using the Light Sensor as a Switch**

There are two different ways that the light sensor can be used to provide feedback to a motion control system. One way is to use the light sensor as a binary or digital switch. This can be accomplished by checking if the value of the light sensor is above or below a certain threshold. As an example, any sensor value that is above 50 would be considered as an "on" state (white surface), and anything below 50 will be labeled "off" (black surface.)

One simple way to follow a line using a binary light sensor for feedback is to have the sensor follow along a path as shown below:

Notice that the sensor is not traveling inside the line, but instead riding along the edge of the line. Running the light sensor across the edge of the line provides the proper feedback to a motion control system because transitions are detected. Transitioning from light to dark and dark to light is the only way for the robot to know if it is following the line. If the robot were traveling along the center of the line or on a large black piece of paper, its reading would always be “off” the entire time that it moved around. This would result in poor position feedback for the robot, and would cause it to wander aimlessly around.

Assuming that the robot followed the upper edge of the line at all times, certain assumptions can be made:

- Whenever a transition from “on” to “off” is detected, the robot must have just crossed into the line.
- Whenever a transition from “off” to “on” is detected, the robot must have just left the line.

Therefore, we can command the robot to turn itself accordingly to stay along the edge of the line by:

- Turning left until an “off” to “on” transition is detected
- Turning right until an “on” to “off” transition is detected

The following diagram shows when the commands to turn left or right would be executed as the sensor takes readings:

Keep in mind that these commands are only valid if the robot is on the left edge (upper edge in this diagram) of the line. Having the robot follow the right edge (lower edge in this diagram) of the line would not work unless the logic for turning is reversed. This problem will be fixed later on in this lab.
With all this said, let’s build a line follower program!

1. Create a new NXT program.
2. Save the program as “NXT Lab 6b.rbt”
3. A logic variable will be used to toggle between turning left and right. Place a **Variable** block found in the Data palette onto the sequence beam. This block will set the initial state of **Logic 1** to true. Configure the block as shown:

![Variable Block](image)

In programming practice, it is always a great idea to initialize all of the variables to some known state before they are used to prevent any garbage data that may be present from confusing the program.

4. Place a **Loop** structure found in the Flow palette after the Variable block. Configure it as shown:

![Loop Block](image)

5. Place another **Variable** block inside the Loop. Configure it to read the contents of **Logic 1**.
6. Insert a **Switch** structure found in the Flow palette after the read Variable block and configure it as shown:

![Switch Block](image)

7. Wire the **Value** output of the read Variable block to the input pin of the lower left edge of the Switch structure. This enables the **Logic 1** variable to toggle between the cases inside the Switch.
8. Place a **Move** block found in the Common palette inside the **true** case of the Switch, and configure it as shown:
9. Place another **Move** block inside the *false* case of the Switch, and configure it as shown:

Notice that the left turn block has a little more power than the right turn block. This was intentional so that it will respond better to curves in the line that go to the right.

Please note that this sample program is being designed with the following assumptions in mind:

- The robot will be tested on a loop that is drawn out on the Test Pad 8527.
- The Tribot will be following along the outside edge of the loop.
- The Tribot will be moving around the loop in a clockwise direction.
- The Tribot will be lined up at the beginning of a straight portion of the loop
- The initial orientation of the Tribot will be parallel with the loop line

10. Place a **Wait** block after the **Move** block in the *true* case of the Switch, and configure it as shown:

11. Place another **Wait** block after the **Move** block in the *false* case of the Switch, and configure it the same way as the previous **Wait** block, but change the **Less Than** symbol to a **Greater than**.

12. Place a **Variable** block after each of the **Wait** blocks in both cases of the Switch. Both of these blocks will be writing a logic value to variable **Logic 1**. The *true* case of the Switch will write **False** to **Logic 1**, and the *false* case of the switch will write **True** to **Logic 1**.

The finished program should look like this:
Effectively the program architecture that has just been created is something called a state machine. The following is a state diagram that describes the operation of the program:

A state machine is a type of program architecture where a function is performed in a state, and then a choice is made on which state to go to next based on criteria. In this example, Logic 1 is determining which case, or state, of the Switch structure should be executed for the current iteration of the Loop. As soon as that particular case finishes executing, a value is passed to Logic 1 on where to go next. The Loop iterates, the value of Logic 1 is read to determine which state to go to next, and the appropriate case or state is executed.

13. Save your program, and download it to the Tribot.
14. Place the Tribot onto the Test Pad on the outer edge of the loop, along a straight portion.
15. Run the program, and watch how the robot attempts to follow the edge of the line.

Questions to Consider:

a) Was the Tribot able to follow along the edge of a line?
b) How well did the robot follow the line when the robot reached the curve?
c) What do you predict will happen if the robot had to follow a left curve in the line? Try it by turning the Tribot around, and placing it along the inner edge of the line. See how well it can make a left turn with the current program.
d) Do you think there is room for improvement? If so, list all of the problems that you noticed in the robot’s performance.
Part C: Using the Light sensor as an analog sensor

Part B has demonstrated how to use the light sensor as a binary source of feedback for motion control. Due to the fact that the sensor has the ability to return a value that ranges between 0 - 100, better control of the robot can be achieved.

In order to use the analog reading of the light sensor properly, it is important to understand how it decides what the value is going to be. To keep this concept simple, assume that the light sensor takes the average light intensity that is detected over a small area. Therefore, as the sensor passes over the edge of a black line, more of the line will cover this small area over time. This effectively lowers the intensity value returned by the sensor. The same thing will occur when moving from a dark area to a light area. The following diagram illustrates this concept:

This behavior can allow more precise control by targeting a value that is between black and white. Suppose that the range of possible sensor values are represented as shades of gray. The following diagram is an interpretation of the edge of a line as far as the light sensor is concerned:
The numbers represent intensity values that would be returned if the light sensor were placed over that particular shade of gray. Picking a midpoint value of 50 as the target edge would grant tighter control of robot because more information on the sensor’s location is being returned to the program. To deliver more precise edge performance, all that needs to be done is a subtraction between the current reading of the sensor and what the sensor should be reading. In this case, the target sensor reading is set to 50. Therefore, a line following robot can be achieved if the motors are set to adjust themselves in order to get the light sensor reading close to 50.

Here are some steps to build a simple program that demonstrates this concept:

1. Create a new NXT program, and save it as “Lab 6b.rbt”
2. Place a Loop structure onto the sequence beam, and leave its settings default.
3. Place a Light Sensor block inside the Loop, and leave it in its default state.
4. Place a Math block after the Light Sensor block, and configure it to perform a subtraction.
5. Connect the Intensity output of the Light Sensor block to the A input of the Math block.
6. Select the Math block, and type “50” for input B. 50 will be the desired sensor reading, or setpoint, for the line following algorithm.

The result of this calculation will be the difference between where the robot is, and where it should be. This information can be used to directly control the direction and power of the wheels. Since this subtraction can produce a negative answer, the direction of the wheels can also be toggled.

7. Place another Math block after the subtraction Math block, and configure to perform a multiplication.
8. Tie the Result output of the subtraction Math block to the A input of the multiplication Math block.
9. Select the multiplication Math block again, and set B to 2. This will act as an amplifier of how strongly the wheels should turn to correct its path.
10. In order to be sure that the robot is always moving forward, place another Math block after the multiplication Math block, and configure it so that it will add 30 to the multiplication result.
11. The last step is to determine if the wheels should rotate forward or reverse. Place a Compare block after the addition Math block, and tie the Result of the addition block to the A input of the Compare block. Configure the Compare block so that it will check if A is greater than 0.
12. Place a Motor block after the Compare block. Determine which Motor output corresponds to the left wheel (typically its Motor C) and configure the Motor block to command that wheel.

13. Expand the cabinet on the Motor block by clicking on the lower left edge of the block.


15. Connect the Result output of the addition Math block to the Power input of the Motor block.

16. Place another Motor block after the previous Motor block, and configure it to control the other wheel (typically motor B) to always move in the forward direction at a power level of 30.

The final program should look like this:

17. Save your program, and download it to the robot.

18. Place the robot back onto the Test Pad on the left edge of the black line, and run the program.

Questions to Consider:

a) Would you say that the robot has better line following performance than before?

b) Try increasing the multiplication factor from 2 to 3. Download the modified version of the program, and observe the results. What differences in behavior do you observe?

c) Try decreasing the multiplication factor to 1, and rerun the program. What differences in the behavior compared to a multiplication factor of 2 do you observe?

19. Try reversing the direction of the robot on the track, and observe how it handles a left curve in the course with a multiplication factor of 2. Notice how the robot responds differently to a left curve compared to a right curve of equal radius.

**Part D: Challenge**

Hopefully you have noticed some odd turning behavior from step 19 of part C. The main reason why the robot responds differently to left curves is because only the left wheel is readjusting to the light sensor’s readings. As a result, the pivot point of the robot is sitting at the right wheel. The robot attempts to make a right turn by slowing the left wheel down enough so that the right wheel makes the turn. The following diagram illustrates what must be adjusted to achieve proper turning:
In order to move the pivot point to the center of the robot, the light sensor data must be somehow delivered to wheel B's motor.

Your challenge is to make the necessary adjustments to the code so that the robot adjusts the velocity of both wheels instead of just one for proper responsiveness to left and right line curves. The robot should be able to make a left turn just as easily as a right turn of equal radius. The only line path that the Tribot is required to take is the loop that is drawn out on the Test Pad.

Good Luck!