LEGO® MINDSTORMS® NXT Lab 2

In Lab 1, the Tribot was commanded to drive in a specific pattern that had the shape of a bow tie. Specific functions 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 movement function 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 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 MINDSTORMS NXT software to understand how the light sensor works
B. Create a program that follows a line using digital sensor feedback
C. Challenge: Improve the line following program using analog sensor feedback

Part A: Understanding the Light Sensor

1. Open the MINDSTORMS NXT software on your computer
2. Open the light monitor VI (**Lab 2 Part A.vi**)
3. Power on the NXT and connect it to a USB port on your computer.
4. Run the VI and observe the waveform chart. Move the light sensor over different surfaces and observe the changes in value.
5. Click the **Generate Light?** control and observe how the values change.

Questions:

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? Is the pattern different when the **Generate Light** option is turned off?

g) Are there any colors that are completely different that produce the same reading? Do the readings match both when the sensor's light is on and off?

h) Test the light sensor on the different colored regions present on the competition course (i.e., white, blue, and black). Note the range of values observed for each color with the **Generate Light** option both on and off. Note these ranges on the classroom's whiteboard.

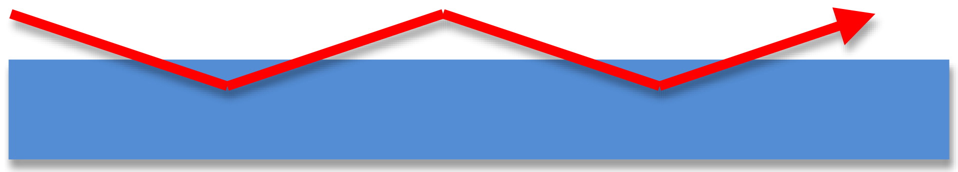
i) BONUS: Suppose there was a need for an application where the tribot had to distinguish between two different colors that causes the light sensor to produce the exact same readings. What could be done to improve the sensor's ability to distinguish between these two colors? (Hint: The answer to question (f) is a clue on how to solve this problem.)

Part B: Using the Light Sensor as a Switch

(Note: We will be using the competition course and programming the NXT to follow the blue/white border around the outside of the course.)

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" (blue surface), assuming that 50 is half way between the typical readings for white and blue.

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 blue 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.

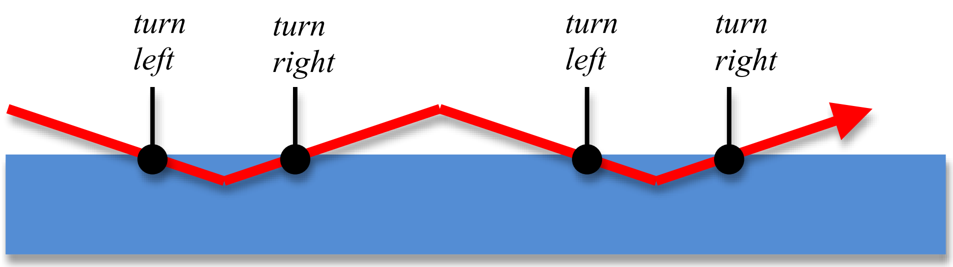
Assuming that the robot followed the upper edge of the line at all times, certain assumptions can be made:

- 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:

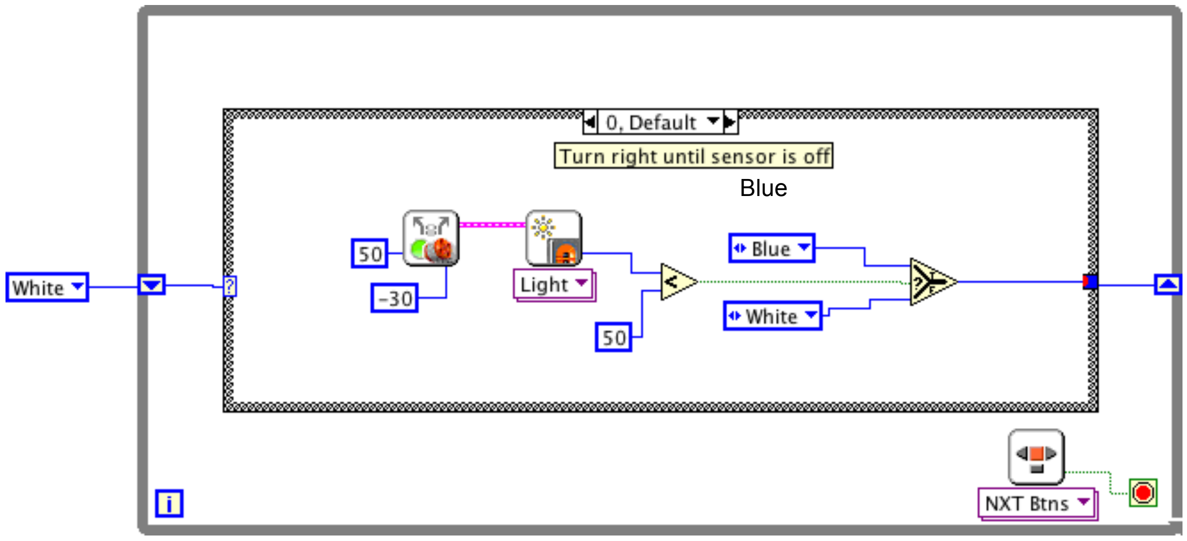
- Turning left until an "off" to "on" transition is detected
- Turning right until an "on" to "off" transition is detected

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 (left to right on upper edge in this diagram) of the line. Having the robot follow the right edge (right to left on upper edge in this diagram) of the line would not work unless the logic for turning is reversed.

Now we will build the digital line follower program.



1. Open LabVIEW and create a new file.
2. Save the file as **Controller.vi**.
3. Place a **While Loop** on the block diagram
4. Wire an **NXT Buttons** function to the loop condition.
   a)Navigate to **NXT I/O** and select the **Sensor** function.
   b)Set the polymorphic selector to **Read NXT Buttons.**
   c)Wire the **Yes/No** terminal to the loop condition.

   The **NXT Buttons** function allows us to stop the program without pressing the gray button, which aborts the program. Similar to using stop buttons on VIs run on the local computer, this allows the program to end itself rather than forcing a termination.

5. Place a **Case Structure** inside the while loop.

   To create the **White** and **Blue** cases, we will now create a constant with 2 predefined, selectable values. This constant is called a **Ring** or **Enum**.

6. Outside the left edge of the while loop, place a **Ring**.
   a) Navigate to **Numerical** and select **Ring Constant**.

7. Create two values called **White** and **Blue**.

a) Right click the ring constant and select **Edit Items**.
b) Double click the first blank line in the **Items** column and enter **White**.

Pressing **Enter** on the numpad will accept changes to the item. Pressing **Enter** on the main keyboard will accept changes and create a new item. If you pressed **Enter** on the numpad, click on the **Insert** button to create another item. Otherwise, continue to step c.

c) Enter **Blue** as the second item.
d) Select **OK** to accept the entered item names.

8. Wire the ring constant to the case structure through the left border of the while loop.

   The case structure will automatically populate cases based on the ring constant, and should now have two cases: **0** (**Default)** and **1**.

9. In the **0** (white) case, place a **Steering On** function with a **50** power constant and a **-30** steering constant.
10. In the **0** case, place a **Light Sensor** function block.
    a) Navigate to **NXT I/O** and select the **Sensor** function.
    b) Set the polymorphic selector to **Read Light >> LED On**
    c) Set the input to the **Output Ports** terminal to correspond to the motor ports you are using.
    d) Place a **Less?** function beside the light sensor function and wire it to the **Scaled Value** terminal.
    e) Right click the other input on the **Less?** function and create a constant of value **50**.

11. In the **0** (white) case, place a **Select** function.
    a) Navigate to **Comparison** and select **Select**.

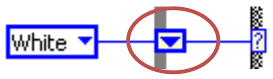
12. Wire the **X < Y?** terminal of the **Less?** function to the **S** terminal of the select function.
13. Wire the **NXT** terminals of the **Steering On** and **Light Sensor** functions together.
14. Create two copies of the ring constant created in steps 6-7 and place them above and below the wire created in step 12.
15. Wire the ring constants to the **T** and **F** terminals of the select function.
16. Change the ring constant wired to the **T** terminal to **Blue**.
    a) Position the mouse over the ring constant until your mouse cursor changes to a hand.
    b) Click to call a menu of available values.
    c) Select **Blue**.

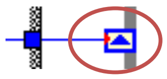
17. Wire the output of the **Select** function to the right edge of the while loop (passing through the right edge of the case structure).
18. Select everything inside the **0** (white) case and copy it to the clipboard.
19. Switch to the **1** (blue) case and paste the code copied from the **0** (white) case.
20. Change the power constant on the **Steering On** function to **50** and the steering constant to **+30**.
21. Wire the output of the **Select** function to the node on the edge of the case structure. It should change from a blue outlined box to a solid blue box. The solid blue indicates that all cases have been properly wired to the node.

    The output of the select function will be a value of either blue (**1**) or white (**0**), depending on the value read by the light sensor. This value needs to be passed on to the next iteration of the while loop to ensure that the robot will execute the correct code case.

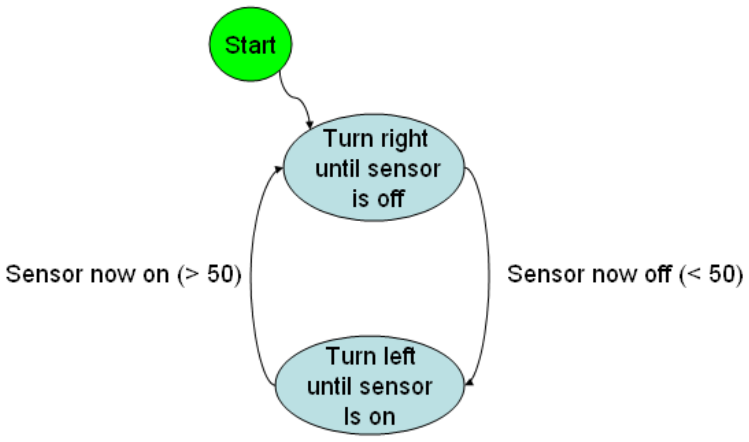
22. On the right edge of the while loop, right click the terminal of the wire created in step 16 and select **Replace with shift register**.

A **Shift Register** passes any data type to the next iteration of the while loop, and appears as a pair of up/down arrows, one on each side of the loop. When program is executed, the shift register is initialized by the ring constant outside the while loop. Inside the loop, the Select function passes its state decision to the shift register (based on the light sensor reading). When the loop executes again, it will read the shift register value from the last iteration and use that value to determine what case to execute. Each case ends with a decision of what case to execute next. That decision is passed to the shift register and the loop repeats again.





The program architecture that has just been created is 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, the **Select** function is determining which case, or state, of the case structure should be executed for the next iteration of the Loop. As soon as that particular case finishes executing, a value is passed to the shift register on what to do next. The loop iterates, the value of the shift register is read to determine which state to go to next, and the appropriate case or state is executed.

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

- The robot will be tested on the competition course laid out in our classroom.
- The Tribot will be moving around the course in a counter-clockwise direction.
- The initial orientation of the Tribot will be parallel with the blue/white boundary.
- A light sensor reading of 50 is about midway between the typical readings for 'white' and 'blue'.

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

Questions:

a) Was the Tribot able to follow the edge of the line?

b) How well did the robot follow the line when the robot reached a curve or corner?

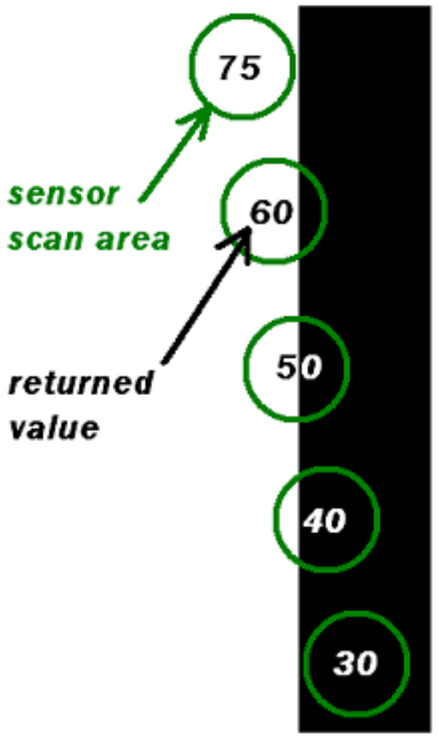
c) What do you predict will happen if the robot had to follow the line in the opposite direction? Try it by turning the Tribot around.

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: (Challenge) Using the Full Resolution of the Light Sensor

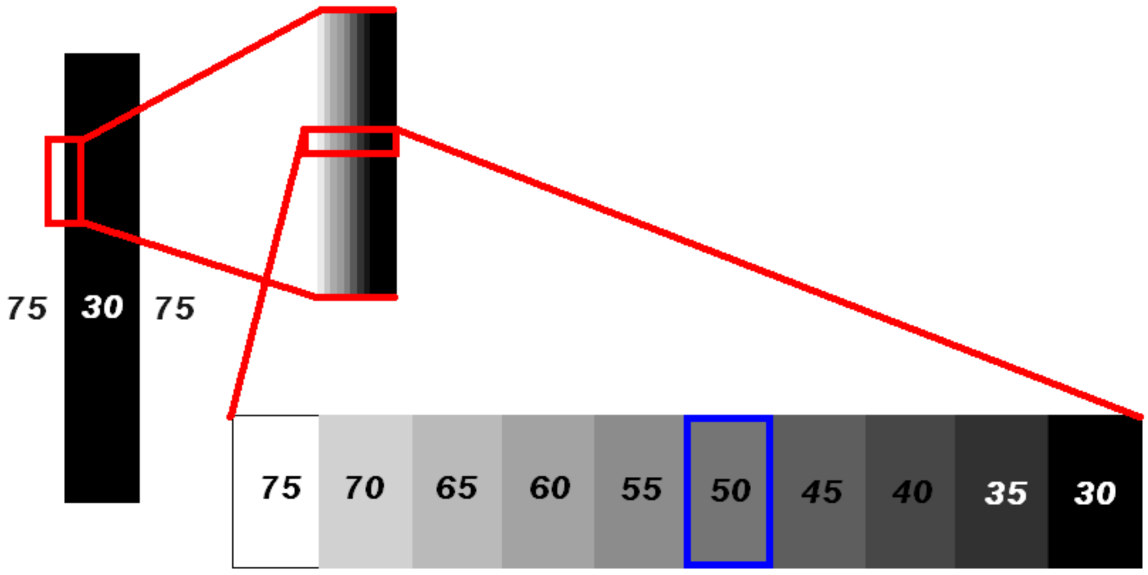
Part B has demonstrated how to use the light sensor as a binary source of feedback for motion control. Due to the fact that the sensor has the ability to return a value that ranges between 0 - 100, better control of the robot can be achieved.

In order to use the full resolution (i.e., gray scale 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 blue 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 blue 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 could be returned if the light sensor were placed over that particular shade of gray. For this case 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.

For this challenge, you will build a *simple* program that demonstrates this concept. To complete this challenge, the tribot must successfully travel around the outer edge of the competition course.

For an additional bonus, design your tribot to travel around the edge of the track faster than any other tribot in the class. Also, design your tribot to pass over the black lines and continue following the blue line on the other side.

Helpful hints:

- Use your current knowledge of the light sensor and motor controls to obtain the tribot's current position and act accordingly to reach the desired location.
- You may need to perform simple math to translate position to movement.
- Remember, this program should be *simple*. Be careful not to overcomplicate it.

Good Luck!