Programming with Sensors

In the last lab, we discussed how we could make the robot move using the green "Move Block." Today, we're going to incorporate more sensors to better implement the Sense/Plan/Act feedback cycle. We're going to focus on the three most useful sensors: touch, light, and ultrasonic.

Sensors

This was briefly discussed in the introduction to today's lab, but I've included the information here in case I left things out or went too fast. If you feel comfortable with what the sensors do, feel free to skip it and move on to the next major section.

Touch Sensor

This is the most basic type of sensor—it can tell if the orange tip is depressed or not. The NXT-G programming environment does a little bit of background work to make this sensor detect three types of actions:

1. Pressed – this happens whenever the orange tip is pressed in. It *does not* need to be "released" beforehand. That is, if you just hold the button down, it will always be counted as "pressed."
2. Released – this is whenever the orange tip is *not* pressed in (basically, the default state of the touch sensor).
3. Bumped – this occurs when the touch sensor is pressed and released within a short timeframe (i.e. 1 second or so). There's some background code that counts the number of "bumps" the touch sensor receives, which can make this setting behave in not-so-easy to predict ways.

Light Sensor

This is a really useful sensor for anything related to electrical-tape lines. Although it cannot detect color directly, it reads *grayscale* values by the amount of reflected light back into the photodiode from the red LED. Dark surfaces reflect less light, and thus have lower sensor readings. Light surfaces reflect more light, and thus have higher sensor readings.

In a perfect world, a black surface would reflect *no* light, and a white surface would reflect *all* light. In case you haven't noticed, this world isn't perfect. To compensate for this, we can **calibrate** light sensors. Calibration tells the light sensor what "should" be 0% reflected light and what "should" be 100% reflected light. You probably don't have to worry about this, but if you want a little extra challenge, see the calibration handout (also on the course website).

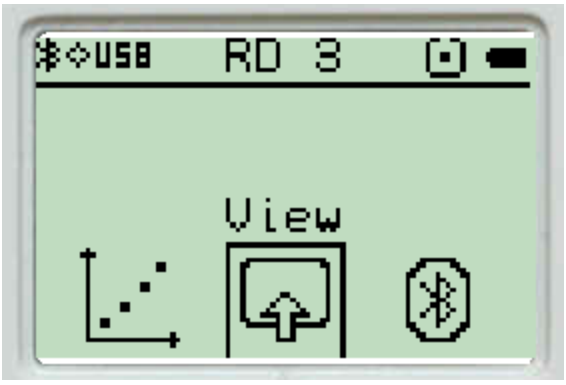
Ultrasonic Sensor

The ultrasonic sensor measures distance using sound, just like SONAR or echolocation. It sends out a pulse of inaudible, high-frequency sound and determines the time it takes to receive an echo. Based on this time and the speed of sound, it can measure the distance to the reflecting surface.

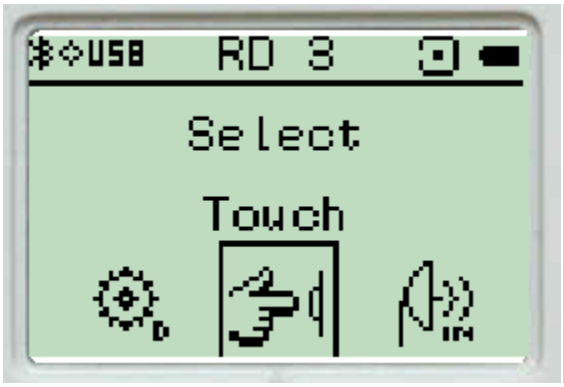
The sensor is accurate to about ±1 inch. Also, the sound wave expands in a conical manner. Thus, as you're further away, it has a wider "field of vision."

Walkthrough: View Menu

Sometimes, you'll want to be able to see what your robot's sensors are currently reading; for instance, what does the light sensor consider "dark," and what does the light sensor consider "light?" You can view current sensor values with the **View** menu on the NXT brick. From the main NXT menu, navigate left or right until you reach **View**, and press enter:



Next, select your desired sensor and port:



You will now see the current value of the appropriate sensor. Try this with your various sensors.

Exercise 0: Moving Forward and Turning

If you didn't get to it before, please go back and work the moving forward walkthrough from yesterday's handout.

After completing that, use the move block's turn slider and some trial-and-error to find the proper "Duration" setting to make your robot turn (approximately) 90 degrees to the left or right.

Exercise 1: Using the Move/Wait/Stop Design Pattern

Using the Move/Wait/Stop design pattern from lecture, create an appropriate program for each sensor type discussed. For instance, have the robot move forward until it detects an obstacle (e.g. a hand) or a black line on the table—we should have placed a strip of electrical tape near each group by now. (You can do this in multiple programs.)

(Design pattern included below.)

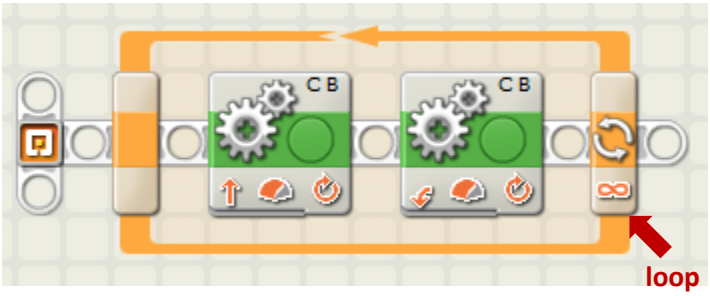
Some sensor
(wait block)

Duration: Unlimited Direction: Stop

Remember that a "Wait" block essentially "pauses" the program at that point and doesn't proceed until the condition in that block is satisfied.

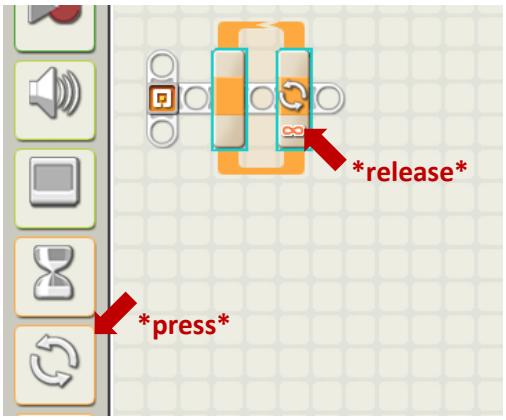
Walkthrough: Loops

If you've worked with programming in any language before, you're probably familiar with the concept of **loops**. A loop is simply a way of telling the computer/robot to repeat a set of instructions until some condition is reached. In NXT-G, loops are represented with a block that "contains" other blocks:

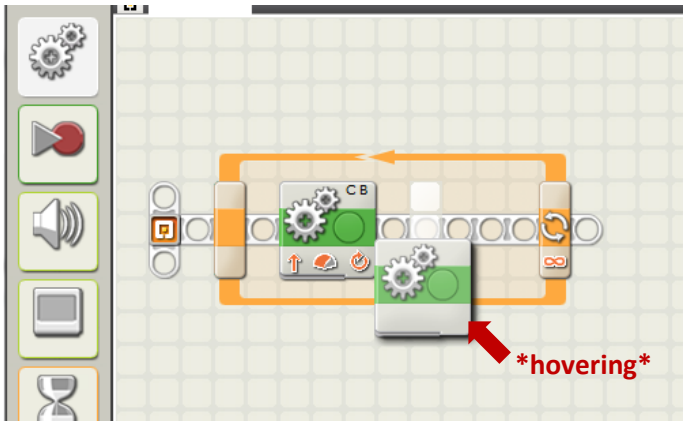


A loop can have many "stopping" conditions; the one above will continue repeating forever (hence the infinity symbol), but we can use other conditions like "sensor value" (loop until a sensor reaches some value) and "count" (loop a specified number of times). There's even a "time" setting that will stop looping after *x* seconds pass.

Let's use a loop to make our robot drive in a square. (You already know the proper duration for a 90° turn from Exercise 0, right?) Pull a Loop Block to the Sequence Beam:

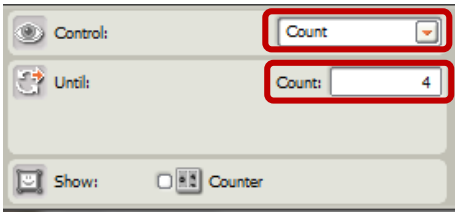


Now, we need to place two move blocks *inside* of this one; while you're clicking/dragging the Move Block inside of the loop, hover over the middle of the loop and it will expand to fit:



Configure the first move block to make the robot traverse the "edge" of the square, and then make the second move block turn 90°.

We're almost done! If you save, download, and run the program now, your robot will try to drive in a square forever. We're only interested in making a single square, so change the loop's control to "Count" and make it loop four times:



Exercise 2: Count Lines

Using a loop and the Move/Wait/Stop design pattern, see if you can get your robot to stop after passing a certain number of lines. (Hint: you'll have to modify the pattern just a little bit—you're not just waiting for one thing.)

Exercise 3: Follow the Hand

Using a loop and appropriate sensors, have your robot attempt to stay 5 inches away from your hand as you either move further away or stay still. (We'll get it to move backwards using another block later.) That is, any time you move your hand away, it drives forward until it sees your hand again.

Competition:

With what you've learned today, what are some ideas you have to approach the competition? Remember, the goal is to complete as many laps in 2 minutes as possible, and you don't want to get stuck on a wall. Start thinking about some approaches; feel free to test them on one of the competition boards. If several teams reach this point, we'll do a walkthrough for line-following.

Questions for Consideration

1. How does a red surface differ from a white surface to the light sensor? Why?
2. Are there particular situations in which the ultrasonic sensor consistently returns inaccurate "bad" readings? Why?