CS148
Building Intelligent Robots

Week 3 : Sensors and Debugging

Out: 11 Feb 2003

Preliminary Tasks

before 13 Feb 2003, 9am

This week's lab will introduce you to the rest of the standard sensors you will be using on your robot to interact with the world. There is no preliminary reading but as always, the brickOS section of the website will be your friend. Everything in this lab you can also find in the HOWTO, command reference, and code samples pages in the brickOS section of the website. Read it; understand it, live by it.

Lab 3: Sensors and Debugging *13 Feb 2003*

Step 1

The MindStorms light sensors measure light intensity (for the most part). The light sensor can be activated in two modes: *active* and *passive*. In *active* mode, the light sensor also emits a red light, enhancing the range and accuracy of the sensor quite a bit. In *passive* mode, the red light is not activated. The active mode is usually much more accurate and stable. One thing to note is that at close distances to objects (let's say, a playing field of some sort), using active mode will return readings that correspond more to reflectivity rather than intensity. The boundary between reflectivity and intensity seems to be about 4–5 inches away from whatever object you're trying to read.

To set the mode, *active* or *passive*, of a light sensor, use the commands:

- `ds_active(&SENSOR_X);` and `ds_passive(&SENSOR_X)` where X is the number of the sensor (1, 2, or 3).
- To read the light sensor values, access sensor number X as a constant directly using `LIGHT_X`. This value is set approximately every quarter millisecond by brickOS.

Task:

- First, write a program that will display the value of a light sensor connected to the RCX brick (pick an input slot, any slot). Choose a suitable time interval between updates and use the `lcd_int(int)` command to display the value read in by the light sensor.
- You will be given a sheet with three color-coded regions: black, green, and white. Write a second program that will determine the correct color of the regions and output either "black", "green", or "white" to the LCD. Again, mount the light sensor on any input you like.
- As with all sensors, the light sensor is highly vulnerable to changes in the ambient environment and thus experimental values should almost never be used as a good indicator. Augment the program you just wrote with a calibration phase. When the new program starts, it should determine the average value of the black, green, and white areas and then run. Your program should be able to correctly identify the three colors in various ambient light conditions.

Step 2

Touch sensors will almost always be used as binary values. To access the value of a touch sensor as a Boolean, use the constant `TOUCH_X` (where X is the sensor number). This will return a 1 if the sensor is pressed and 0 if the sensor it not.

Task: Write a program that recognizes touches on the bump sensor. Your program should give some indication (beep, LCD print out, spins its motors wildly) that a connected touch sensor has been depressed.

Step 3

Since there are only three inputs on the RCX, we must use them as efficiently as possible. One way to do this is to put a touch and light sensor on the same input. Access the input as you would a light sensor. You will find that when the touch sensor is depressed, the light sensor reading will read a certain range of values outside the norm of a light sensor. In fact, the readings will also vary with the strength of the depression of the touch sensor.

Task: Find out the exact value (or value range) of the light/touch sensor input when the touch sensor is depressed. Use this value range to improve upon your program so that is recognizes all three colors in various conditions and will also display "bump" on the LCD if the touch sensor is depressed.

Step 4

The MindStorms rotation sensors can sense rotation in steps of 1/16th of a revolution. Thus, for most applications, you will probably want to gear up what you're measuring to improve the resolution.

To initialize an input to be read for rotation, call the function `ds_active()`, just as you would do with a light sensor. Use `ds_rotation_set( &SENSOR_X, int position )`. This will set sensor X to an arbitrary position. The default value is 0. `ds_rotation_on( &SENSOR_X )` must then be called to turn on the sensor.

To access the sensors, use `ROTATION_X` (where x is the input number). The number returned is the number sixteenth of revolutions the sensor has turned. The value will increment one way and decrement the other way.

Task: Write a program that will initialize and then output the number of revolutions the sensor has turned.

Project 3: Sensors and Debugging *due 20 Feb 2003, in class*

Specification: The purpose of this project is to become more familiar with the many quirks of the light sensor, as it is the most important—and annoying—sensor you will be using. You must build a robot that will follow a line on the floor. The line will be about $1''$ wide and of a significantly different color than the floor. You cannot assume the line is darker than the floor, or vice-versa. Your robot should have a calibration phase that takes into account this fact and ambient conditions so that your robot could work on any line in any room with any lighting conditions.

When the robot gets to the end of a line, it should not travel along the same line in the opposite direction. Your robot may or may not start on a line after the calibration phase. If it doesn't start on a line, it should go straight forward. If it hasn't found a line within 7 seconds of dead time at any point during its task, it should stop.

Some problems you might run into:

- How many light sensors should you use?
- How should you design the calibration phase?
- How can you shield the light sensor so a minimal amount of light leaks into it? (Note: good shielding is extremely helpful!)
- Should you design your robot so the line takes precedence or the floor takes precedence? In other words, should your robot follow only the line you tell it to follow or any line that is different from the floor? (Both are perfectly acceptable.)

Note: It would a really really good idea to start this project before you go to your own personal Cancun this President's Day weekend. Light sensors introduce a lot of inconsistencies that can't be easily predicted. The only way to catch and fix them is by trial and error. Heed this warning. You do not want to pull a Wednesday night all-nighter in the MSLab.

Paper Handin: Please staple your individual written report and write your group number on the front page. Your report should include a description and drawings of key design elements of your robot. This would also include the number and type of sensors used and how you used them. You should explain your light-following algorithm. Pseudocode is not necessary for this explanation, but recommended. In addition, explain what your robot does for calibration, including how it gathers data and what it does (statistically) once it gathers it. Note whether your robot follows all lines that are different from the floor or just the line it is told to follow. Finally, describe any major failed attempts (bad sensor setup, poor shielding, unsuccessful gearing, etc). If your first design was perfect, please state so! Please electronically hand in your code, as well (`/course/cs148/bin/cs148_handin sensors`). One copy of your code will be sufficient for the group.

Grading: Both your lab write-up and your actual robot will determine your grade.

Your robot will be scored in the following manner:

Follows line	30%
Robust in various ambient conditions	15%
Technical design (does not break, etc.)	5%
Total	**50%**

Your report will be graded as follows:

Design explanations	15%
Line-following algorithm	15%
Calibration	15%
Design process	5%
Total	**50%**