CS190 10/06/06

Robotics Lab5a: NXT-a-Sketch

Introduction

The goal of this lab is to continue learning C and RobotC and will cover using rotation sensors, graphics, and constants.

Both parts of this lab are **due by 11:59pm on Monday 10/16**. You work is strictly governed by the Math/CS SPCA (http://www.mathcs.emory.edu/SPCA/). In particular your work must be exclusively your own and you may not collaborate with any other group or consult any resource other than those indicated below. Violations of this policy will be referred to the Emory Honor Council.

In order to receive full credit for this lab, you must answer each numbered question below and turn in each programming task. You will turn in your programs by placing them in a folder called “lab5” which resides in the “turnin” folder on the designated group member’s webdrive (see http://it.emory.edu/showdoc.cfm?docid=1183&fr=1057). Make sure that this folder is readable only by your group members and “oparekh.” You should also create a “cs190” folder, readable only by your group members, in which you store your files while you are working on them. It is preferred that you electronically turn in your answers to the lab questions in a filed called “answers”; acceptable formats include plain text (“answers.txt”), rich text (“answers.rtf”), and Microsoft Word (“answers.doc”). Please let me know if there is some other format you prefer. If you would rather turn in a hard copy of your answers, please slide them under my door (W424) if I am not in my office.

Part I: Your own view rotation program

In this part you will write a program similar to the built-in “Motor Rotations” program on the NXT brick. The rotation sensor may be read using syntax similar to setting the power level of a motor:

```
int rots;
nMotorEncoder[motorA] = 0; // reset rotation sensor A
...
rots = nMotorEncoder[motorA]; // read rotation sensor A
```

Check out the sample code section of the RobotC site, http://www-education.rec.ri.cmu.edu/robotc/nxt/index.html for examples of simple programs that use the rotation sensors. variables and program flow in C. Next, read the documentation for the nxtDisplayTextLine function at: http://www-education.rec.ri.cmu.edu/robotc/nxt/index.html.

The “text” parameter is a template specifying how the output should appear; it is very similar to the first parameter of printf function that you have seen in the GNU C tutorial. Although the NXT’s function only supports printing the values of integer variables using the %d “place holder,” printf is far more versatile in the types of variable values it can print. See section 16.2.2 of the GNU C tutorial for detailed information.

1. Write a program, **MotorRotations.c,** that displays the current values of the rotation sensor for motors A and B on the first (topmost) and second lines, respectively, of the NXT display.

Make sure “Code Completion” is checked in the “View” menu. This is the (perhaps annoying) feature that suggests completions as you start typing text in the editor. Begin typing “nxtDis,” and you should notice a function called nxtDisplayString. Select this option, then select the text “nxtDisplayString” and right-click on it. After the Cut, Copy, and Paste options you should see that the editor is telling you what it knows about the nxtDisplayString function, namely the number and types of the parameters it takes. Compare this with the corresponding information for nxtDisplayTextLine.

2. Modify your program to use nxtDisplayString instead of nxtDisplayText-

Line. Explain difference between these two functions? If you see a difference but are not sure exactly what it is, then feel free to write a quick test program to try and figure out the exact difference. Which is more appropriate for our purposes?

Now we will "one up" the wimpy built-in Motor Rotations program by displaying both the number of degrees and rotations. Modify your program so that it prints out the number of rotations for motors A and B on the third and fourth lines, respectively. Include on each line a short label so the user knows what the displayed information represents.

3. Turn in this version of the program.

From what we know about integer division in C, we know that that we should expect only the whole number of rotations (with any fractions discarded). Suppose we wanted a more precise measure of the number of rotations. One option is to use the same trick we used for NXT-G (i.e. multiply by 10^k to display the first k digits after the decimal). Although (Robot)C provides a float type for this purpose, the nxt text display functions do not currently support printing the value of floats.

Extra credit: Given a float f, recall (Chapter 5 of the tutorial) that we may use the expression (int)f to cast the value of to an int by throwing away everything after the decimal point. Write a function int fractionalPart(float f) that given a float returns a integer representing the first two digits after the decimal place in f. For example, fractionalPart(1.2345) would return the int value 23. Can you extend this to a function int fractionalPart(float f, int k) that returns the first k digits? Use your function to display a decimal point and the fractional part of the number of rotations in your above program.

Part II: NXT-a-Sketch
In this part we will complete a program that emulates Kevin's **Nxt-a-Sketch**:
http://www.idreamincode.com/mt/archives/cat_nxt_lego_mindstorm.php

Download the **nxtaSketch**.c program for the share/labs/lab5 directory. This is a program that "almost" works. Your job is to fix it. The most useful tool at your disposal is the Step (Into) function of the debugger and the "Global Variables" window.

Notice that various constants have been declared at the beginning of the main task, using the const keyword. Also notice that although we (the RobotC developers and I) were free to name our constants any way we wished, we have chosen to prefix the name of our constants with "k."

4. How does the compiler enforce constancy (hint: write a simple test program that declares a constant and then tries to modify its value)?
5. We could have elected to directly use the value 100 wherever we needed it instead of declaring kDispWidth. What are some advantages of declaring a constant to represent this value (hint: what if we also had another constant whose value was 100 -- there are other very good reasons too)?

We will talk about how Etch-a-Sketches work and how the **NXT-a-Sketch** program should work. Think about how you would write an **NXT-a-Sketch** program, and then read through the program and comments. The nxtSetPixel function is one we haven't used before; however, you can probably guess how it works (select the "nxtSetPixel" text and right-click on it). Write a simple test program to figure out the geometry of the screen by calling nxtSetPixel with various values for the xPos and yPos parameters.
6. Where is the point (0,0) on the screen?
7. What are the maximum x and y coordinates that may be displayed on the screen?
8. What happens when you try to set a pixel that is not on the screen?

There is a bug that prevents anything from being drawn. Your first task is to discover this bug.
9. Explain the nature of the above bug, and explain how you fixed it. What are some simple rules you could follow to avoid bugs like this in the future?

The next bug is an orientation bug. From our discussion, the left wheel, on port A, should move right when rotated clockwise and left when rotated counterclockwise; the right wheel, on port B, should move up when rotated clockwise and down when rotated counterclockwise. How is the buggy version of the program inconsistent with the above?
10. Explain and fix the bug.
11. Change the value of the kRotRange constant from 360 to 720. How does this change the behavior of the program?

Revert to the original value of kRotRange. The final issue is not necessarily a bug, but more of a preference-related issue. The program currently "remembers" the position of the pointer if it moves off screen. For instance if the pointer is just at the right edge of the screen, and wheel A is turned clockwise two full rotations, then the pointer will disappear off the screen, and it will take two full counterclockwise rotations for it to appear again. Modify the program so that the pointer is not allowed to leave the screen.

In effect you may implement this behavior by ignoring the appropriate rotation sensor when the pointer is at an edge. An issue is, however, that the rotation sensor will be modified even if you are ignoring its value. Note that you **cannot** currently set the rotation sensor to an arbitrary value with code like

```
nMotorEncoder[motorA] = 50;
// will be reset to 0 even though 50 was specified
```

I suggest you use extra variables to keep track of the rotation value. You can use the integer remainder operator to limit the range of a value:

```
a = b % 100;
// a will always been between 0 and 99 no matter the value of b
```

Please come talk to me if you need help.
12. Turn in your final **nxtASketch.c** program.