Vinayak Vedantam
02/17/2015

Sensor Characteristics

I. Intro

This assignment contained 3 distinct micro-assignments. The first of which was to mount a light sensor onto the servo and write a program that would scan for light and leave the light sensor pointed in the direction of the most light. Hence, if i were to position a flashlight to the right of the robot, the light sensor would scan around the robot and point directly at the flashlight. The range of this program was supposed to be at least a few feet.

The second part of the assignment asked us to mount and calibrate a potentiometer which would accurately measure the angle of the light sensor. This addition would allow us to run tests to measure the accuracy of the different enclosures on our light sensor.

The third part of the assignment utilized our knowledge from the first 2 assignments and required us to move the robot to drive toward the greatest source of light around it. This would incorporate the active directionality sensing function from the first part of the assignment. If the robot were unable to find a light source, it should have wandered randomly until the light source came into view.

A. *Active Directionality Sensing*

II. Mechanical Design

The initial mounting of the light sensor on the robot was fairly easy. We mounted a servo motor upon the back of the robot and attached a light sensor on a small rod above the motor. The servo motor and light sensor were directly connected to the microcontroller unit and thus could operate while the robot was stationary. The servo motor is especially useful as it can store and move to specific locations.

III. Algorithm

The basic algorithm for this section was easy, but vitally important, as it was the basis for every other section of the lab. The readings obtained from the servo motor and the values used to code its functions were especially important. Servo motors operate under the range of [-127, 127]. In addition, these motors move more than 180 degrees. Our algorithm, thus, had to check the value of the light sensor at some incremental value from -127 to 127 and record the servo motor location for the point of greatest light. We started by initializing two int variables, min and minl. min was instantiated at 1100 and served as our record for the brightest light encountered, while minl was instantiated at -127 and served as our record for the location of the brightest light. The program then checked to see what the value of the light returned at the current location, and if it was brighter than the current records (which was always the case for the first loop since min and minl were instantiated to the minimum values), the values for light and location were saved in min and minl respectively. After running through the loop, the program set the servo motor to 10 units higher than its current value and ran the tests again. If the new light was brighter, then new values for light and location were recorded. Else, the program added 10 more units to the servo and checked again until the servo value reached 127. At this point, the servo accepted the value stored in minl and turned towards the location where the brightest light was recorded.

IV. Performance evaluation

When we began this lab, I misunderstood the instruction. I believed that the entire robot was supposed to move and face the light source. Thus, I attempted to move the robot to scan for and face the direction of the light. This was far more difficult, as the right and left motors on the robot were unable to store and recall location data like the servo motor. However soon after realizing my mistake, I switched to the proper technique. Our program also encountered to instances where it simply would not load onto the robot. The first time, our problem lay in the 'for' loop, wherein the incrementing statement used '+' instead of '+=' and thus failed to work properly in RobotC. Our next issue was resolved by implementing a wait period after moving to the source of the light. Without this wait period, the robot would simply face the light and then end the program and reset the servo motor to its default value (0). However, our robot usually waited a third of the instructed waiting period. We were unable to figure out why this happened, but were able to fix the issue by simply increasing our desired waiting period by a factor of 3. Our robot only ran into one hardware issue, which was due to an unfortunately small distance between the screws used to install the light sensor and the servo motor. Occasionally, the screw heads would get stuck on one another and the servo motor would be unable to move completely to position minl. To fix this, we called the movement to minl again after a 1 second waiting period.

B. *Accuracy Measurement*

II. Mechanical Design

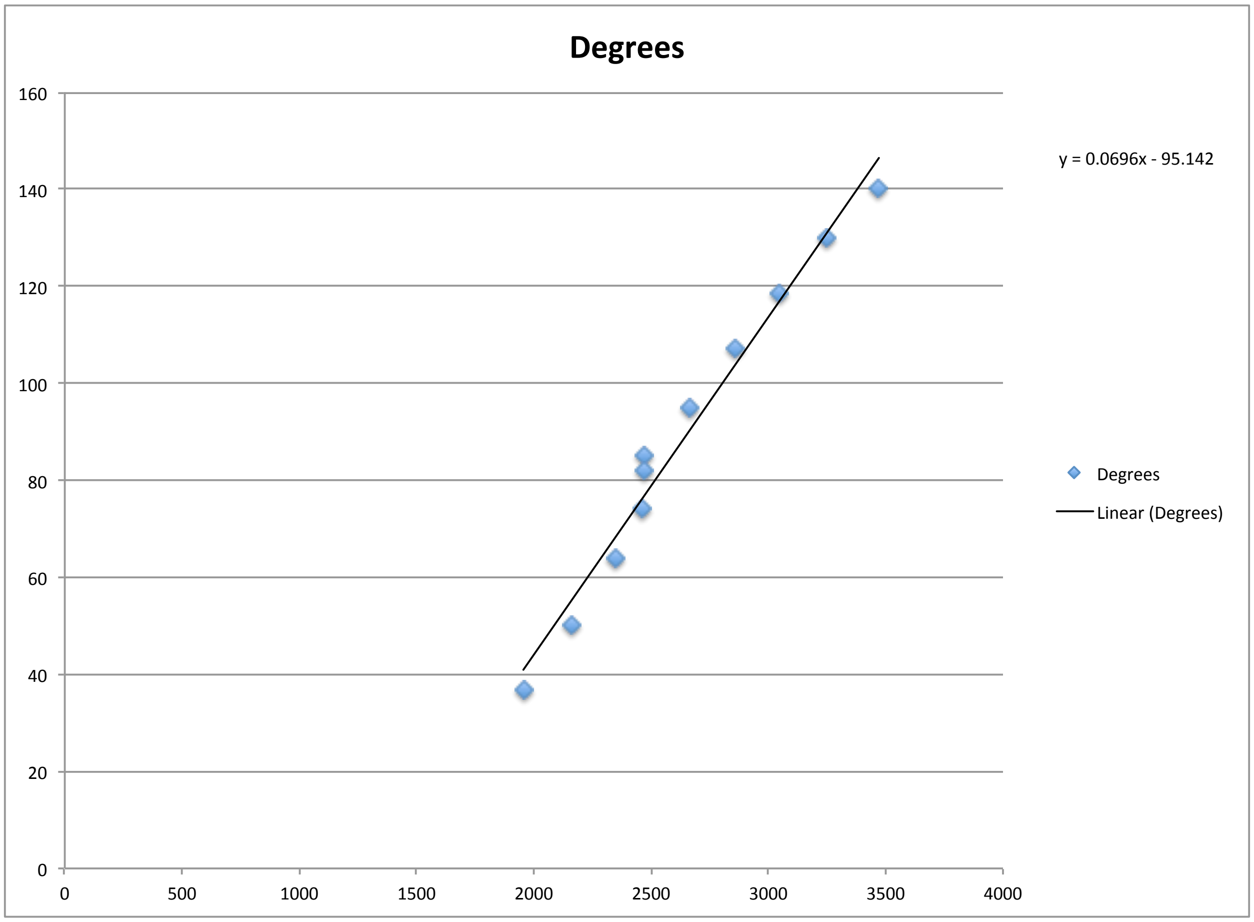
The accuracy measurements required mounting a potentiometer onto the the robot. The actual device sat above the servo motor and below the light sensor. It was a very tight fit. We changed our design from part one and put 3 threaded beams to allow for more places where the potentiometer could be mounted.

In order to calibrate the potentiometer, we added a protractor to the robot as shown. This protractor was flush with the light sensor and corresponded directly with both the potentiometer and the servo motor.

Two different enclosures were also mounted onto the light sensor in order to increase the aperture size and decrease the amount of outside light allowed to fall on the sensor. This should have theoretically increased the accuracy of our measurements, and we were very pleased with the results. The trick with the enclosures was to make sure that they did not sag while attached to the light sensor, and thus could block outside light, but not direct light. We used heavy amounts of duct tape to support the enclosures and keep their opening directly in line with the light source.

III. Algorithm

The calibration of the robot required careful measurement of the corresponding degrees and potentiometer readings at each programmed servo motor point. Below is a table and graph of the data. Notice that the potentiometer readings do not scale exactly to the degrees. After plotting the data, we found a line of best fit with the potentiometer readings on the x-axis and the degrees on the y-axis.



Servo	Pot	Degrees
-100	3471	140
-80	3250	130
-60	3043	118.5
-40	2860	107
-20	2662	95
0	2467	85
20	2466	82
40	2462	74

60	2350	64
80	2162	50
100	1955	37

We calibrated the potentiometer and receiving a line of best fit:

$$y=0.0696x - 95.142$$

This equation gives us an empirical formula for calculating pot results from degrees. After finding this equation, we moved on to part b, accuracy measurement.

This step required running our Active Directionality Sensing program at least 18 times. We took 3 measurements for each of our enclosure and light settings. Each set of tests were executed at 45, 90, and 135 degrees. The first measurements were completed with the light source resting approximately a foot away from the light source. The second measurements were completed with the light source resting approximately 2 to 3 feet away. Measurements were taken for the robot without a light enclosure, with the 1 inch enclosure, and with the 2 inch enclosure. The results in potentiometer readings is shown below:

	Close	**Far**	**1 in - Close**	**1 in - Far**	**2 in - Close**	**2 in - Far**
45 degrees	1754	1750	1750	1735	1775	1800
90 degrees	2600	2440	2560	2330	2515	2720
135 degrees	3550	3800	3260	3245	3320	3339

After this data was collected, we compared the actual retrieved values with the empirical values gained from our line of best fit to measure the accuracy of each of the enclosures. Negative errors indicate that the real pot value was higher than the empirical value:

	Empirical	Error - Close	Error - Far	Error - 1 in Close	Error - 1 in Far	Error - 2 in Close	Error - 2 in Far
45 degrees	2013.5	12.8%	13.09%	13.09%	13.83%	11.84%	10.60%
90 degrees	2660.1	2.26%	8.27%	3.76%	12.41%	5.45%	-2.25%
135 degrees	3306.6	-7.36%	-14.92%	1.41%	1.86%	-0.40%	-0.98%

IV. Performance Evaluation

We initial completed our calibration incorrectly and used theoretical pot values in comparison with the pot values retrieved from 2b and 2c. However, we were able to go part to 2a and retrieve empirical data without tampering with the machine. Thus, our values for 2b and 2c were not wasted.

Our tests showed conclusively that our 2 inch enclosure provided the least amount of error. This result made sense since the 2 inch enclosure would let in the least amount of outside light, and would thus allow the light sensor to take more accurate readings during each loop of the program. The main trade off between the each enclosure was that the larger enclosures provided increasingly accurate results, but were only able to find light directly in front of the robot. If our light source were at a 45 degree angle above the robot, the 2 inch enclosure would not be able to find it, whereas the light source without an enclosure might be far more accurate. For the purpose of this lab and our demonstration, the 2 inch enclosure provided us with the lowest error as shown in the table above. We had to make sure that the enclosure was mounted properly at all times, however, since a small shift could greatly impact the amount of light received by the sensor.

C. Photoaxis

II. Mechanical design

We only made 1 change to the robot for part 3 of this assignment. Since our light sensor is mounted onto the back of the robot, We added two threaded beams to the back of our robot to protect against accidental collisions.

III. Algorithm

We decided that the first step to moving our robot towards the light source was to understand how to turn our robot. We decided upon settings one set of wheels to max speed while leaving the other set of wheels stationary. This allowed our robot to turn. In addition, we decided that we would like to have our robot adjust itself until it was directly facing the light source before moving forward.

This part of the assignment simply added upon the active directionality sensing program we defined in the first part of this assignment. We implemented simple if conditionals. If our servo value was between -20 and 20, we considered the light source to be directly ahead of the robot and propelled the robot forward at full speed. If the value of the servo was negative but not greater than -20, we moved the left motor (since our robot is now moving backwards, this motor functions as the right motor) at full speed for a time defined by 375/ 45. This number came to me as a brainwave on the last day of lab. Instead of using half a dozen if statements to define the amount of time the robot turns, I could use multiplication to scale the time the robot turns to the number of degrees. At first, I tried the time for a 90 degree turn divided by the total motion of the servo motor on one side (127). However, this was inaccurate since the servo motor did not actually turn 127 units to one side, and the time for a 45 degree turn was not exactly half that of a 90 degree turn. Thus, I tried dividing the time required for a 45 degree turn by the degrees turned (45).

375/45 = 4.6875.

This value proved to scale well. For negative values, this value had to be negative, since minI was also negative, and the robot could not wait for a negative amount of time. After turning for the scaled amount of time, the robot would stop moving and rescan for the light source.

The else case of this procedure was executed if the servo value was positive but greater than 20. The right motor (functioning as the left motor) operated at full speed for 4.6875 * minI seconds and then stopped.

IV. Performance Evaluation

Our program worked very well. Often the robot would make 2 or 3 small adjustments after facing the direction of the light source, but I believe this was caused by our limited range defined for the first if case. The light source presumably registered just outside the range (-20, 20), and our scaled wait time was probably too large for such small turns. Thus, the robot took a little while longer to face the target and advance. However, the program worked every time regardless of the initial direction of the robot.

We faced considerable confusion with the directions during the first 2 parts of the assignment. While the 3rd part was far more straightforward, we encountered a very different issue. The robot fell off the desk during one of our tests and the clutch on the right motor became worn on the outside. Thus the metal rod connecting it to the gears and wheels did not fit properly and often caused the wheels to remain stationary while the motor ran. In order to fix this issue, I had to completely disassemble the right side of the robot and replace the clutch. Upon replacement, however, the robot performed admirably.

V. Conclusion

This lab was very extensive and very strenuous. This was due, in great measure, to our continual misunderstandings concerning the directions. Many of our tests had to be repeated, and this slowed us down immensely. This being said, the lab was very rewarding. I cemented my understanding of imports and how the robot microcontroller communicated with RobotC. In addition, I found myself writing in C as easily as I would write in any one of my other programming languages. If I had to change anything about this lab, I would use examples to make the directions for the accuracy measurement less confusing. Overall, lab 3 was very useful and made for a great learning experience.

1.
```
#pragma config(Sensor, in1,    lit,           sensorReflection)
#pragma config(Sensor, in2,    lit2,           sensorReflection)
#pragma config(Sensor, in3,    pot,            sensorPotentiometer)
#pragma config(Sensor, dgtl7,  sonarSensor,    sensorSONAR_mm)
#pragma config(Motor,  port1,           rightMotor,    tmotorVex393, openLoop)
#pragma config(Motor,  port6,           servoMotor,    tmotorServoStandard, openLoop)
#pragma config(Motor,  port10,          leftMotor,     tmotorVex393, openLoop)
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//

task main() {
	int sensor_value;
	int min = 1100;
	int minI = -127;
	while (true) {

	for(int i = -127;i < 127; i += 10)
		{
		motor[servoMotor] = i;
		wait1Msec(500);
		sensor_value = SensorValue[lit2];
		if (sensor_value < min) {
			min = sensor_value;
			minI = i;
		}
	}

	wait1Msec(1000);
	motor[servoMotor] = minI;
	wait1Msec(1000);
	motor[servoMotor] = minI;
	wait1Msec(1000);
```

3.
```
#pragma config(Sensor, in1,    lit,           sensorReflection)
#pragma config(Sensor, in2,    lit2,           sensorReflection)
#pragma config(Sensor, in3,    pot,            sensorPotentiometer)
#pragma config(Sensor, dgtl7,  sonarSensor,    sensorSONAR_mm)
#pragma config(Motor,  port1,           rightMotor,    tmotorVex393, openLoop)
#pragma config(Motor,  port6,           servoMotor,    tmotorServoStandard, openLoop)
#pragma config(Motor,  port10,          leftMotor,     tmotorVex393, openLoop)
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//

task main() {
	int sensor_value;
	int min = 1100;
int minI = -127;
while (true) {

for(int i = -127;i < 127; i += 10)
        {
        motor[servoMotor] = i;
        wait1Msec(500);
        sensor_value = SensorValue[lit2];
        if (sensor_value < min) {
                min = sensor_value;
                minI = i;
        }
 }

wait1Msec(1000);
motor[servoMotor] = minI;
wait1Msec(1000);
motor[servoMotor] = minI;
wait1Msec(1000);

if (-20 < minI && minI < 20) {
        wait1Msec(1000);
        motor[leftMotor] = -127;
        motor[rightMotor] = 127;
        wait1Msec(3000);
        motor[leftMotor] = 0;
        motor[rightMotor] = 0;
        wait1Msec(1000);

} else if (-127 < minI && minI < -20) {
        wait1Msec(1000);
        motor[leftMotor] = -127;
        motor[rightMotor] = 0;
        wait1Msec(-4.6875 * minI);
        writeDebugStreamLine("%f", -4.6875 * minI);
        motor[leftMotor] = 0;
        motor[rightMotor] = 0;
        wait1Msec(1000);

} else {
        wait1Msec(1000);
        motor[leftMotor] = 0;
        motor[rightMotor] = 127;
        wait1Msec(4.6875 * minI);
        motor[leftMotor] = 0;
        motor[rightMotor] = 0;
```

```
				wait1Msec(1000);
				}
		}
}
```