TO: Dr. Carlotta Berry
FROM: Arbib, Ellie Honious and Rahul Yarlagadda
DATE: 7 January 2018
SUBJECT: ECE425: Lab 2-Obstacle Avoidance and Random Wander

The purpose of the "Lab 2- Obstacle Avoidance and Random Wander" was to meet one of the most crucial characteristics of an autonomous robot- to navigate through some environment, safely. In order to reach this goal, a plan needs to be implemented, attached is the pseudo code in state diagram and transition table form. In order to reach these plans and goals, the following functions were created: robot motion, update state, update sonar, and update IR. The update functions read and average the analog sensor values then use the appropriate calibration equation to make the reading more accurate. The update state function was used to define the state that the robot was in and was later used to turn on different colored LEDs to establish a visual and live representation of which state the robot was in. The robot motion function was used to achieve the required motions as defined by the various layers from within the lab.

Calibration

Of course, planning is not the only preliminary work that is necessary when creating an autonomous robot. There was also extensive sensor calibration. There are 4 IR and 2 sonar sensors on the robot, and each one needs to be tested and calibrated. Attached are the tables and graphs that were used to establish each calibration equations. In order to calibrate each sensor, each sensor was placed from 1 to 20 inches at 1 inch intervals and the average analog value from the sensor was recorded. The analog values were converted to inches and adjustments were made, to create reasonable equations, to match as closely to the actual distance as possible. Error was also calculated, using root sum mean error (RSME). The equation used to calibrate the IR sensors was:

$$y = \frac{m}{Analog + b} - k,$$

where *m*, *b*, and *k*, were adjusted in order to create the best fit and the analog value was the value that was read directly from the sensor. The equation used to calibrate the sonar sensor was:

$$y = m \cdot Analog + b,$$

where, once again, *m* and *b* were adjusted and the analog value was read directly from the sensor. After the calibration adjustment all six sensors were very similar to the actual measured distance, all of the errors were below 1%, at about .98-.99%. It was however determined, that the ranges for the IR sensors are around 5-13 inches. While the ranges for the sonar sensors are around 5-20 inches.

Physical Robot

1. What was the general plan you used to implement the random wander and obstacle avoidance behaviors?

For random wander, the robot would generate a random angle. The robot would then turn to that angle, and drive forward a set amount of distance (½ of a wheel revolution). After the robot had driven forward that set distance, it would generate a new random angle, and repeat.

For obstacle avoidance, we would still randomly wander if there were no obstacles detected. However, if the robot saw an obstacle, the robot would generate an angle opposite the obstacle instead of a random angle. If multiple obstacles were detected, it would use the angle of the sum of vectors going away from each obstacle. Then, it would spin to that desired angle, and drive forward.

2. How did you create a modular program and integrate the two layers into the overall program?

We have each layer defined in a separate function, with a global layer variable stating the top layer that is currently active. An if-statement in the main loop of the program then only executes the functions that are either at the current level or at a level below the current level.

3. Did you use the contact and IR sensors to create redundant sensing on the robot's front half.

We only used the IR sensors on the robot's front half for sensing obstacles. The contact sensors do not exist on this robot platform, and we have not gotten the sonar sensors reading reliably to add redundancy.

4. How could you create a smart wander routine to entirely cover a room?

We could constantly travel in one direction (for example, the x axis) and move until we detect an obstacle. We would then move around that obstacle and keep traveling in that direction. Once we encounter an obstacle that we can't seem to avoid without a drastic change in the other axis, we could then travel back down the x axis at a slightly offset y position. Then, we repeat, moving back and forth across the room, moving slightly more in the y after every traversal.

5. What kind of errors did you encounter with the obstacle avoidance behavior?

Often, just because of the position of the sensors, the robot would maneuver to avoid an obstacle but not move quite far enough. In this situation, the robot would have moved far enough over that the sensor would no longer detect the obstacle, but the wheel would catch on the obstacle.

6. How could you improve the obstacle avoidance behavior?

By adding more sensors to the robot (such as the sonar sensors) we could increase the robot's field of vision, and decrease the number of situations where the robot would physically hit a obstacle in a position where none of the sensors could detect the obstacle.

7. Were there any obstacles that the robot could not detect?

Obstacles that were directly in front of the wheels were hard to detect and were often just run into. In an attempt to correct this blind spot, we tried increasing the correctional motions when the robot initially saw the object because it was more often the corner of the box that would be unseen, after the robot had maneuvered around its side.

8. Were there any situations when the range sensors did not give you reliable data?

For most of our sensors, we found that any deadbands were before 5 inches and after 20. This actually turned out to be fine for us because we needed the robot to react by 5 inches in order to account for its length. Any smaller threshold and the back end of the robot usually hits the object it is avoiding when making the 90° turn away from it.

9. How did you keep track of the robot's states in the program?

We kept track of the current robot's state in a global variable. Additionally, a separate variable was used to keep track of the state of every obstacle sensor, and keep track of whether a obstacle was present or not. Therefore, when the state of a sensor changed, we could also update the overall robot state.

10. Did the robot encounter any "stuck" situations? How did you account for those?

We did not actually encounter any "stuck" situations while working with the robot. However, they are possible, especially if an obstacle were to suddenly appear within the 5 inch dead space of a sensor. This could be accounted for with the use of encoders, if the robot continues with the same readings and state for so many seconds, an "abort" function could be created. This would make the robot move in the opposite direction and re-attempt random wander.

11. How did you keep track of the goal position and robot states as it integrated avoid-obstacle and go-to-goal behaviors?

We assumed that when the robot was turned on, it was at (0,0) and facing an angle of 0 degrees. Every time a random angle was generated during random wander, these angles were considered to be absolute angles, so the robot would spin by the difference of its current angle and the desired angle. The same logic was used even when the angle was generated to avoid an obstacle or to go towards a goal, instead of randomly generated. Since the robot always drove forward the same distance based on the stepper motors, we could use atan2 to figure out the x and y translation of the robot, and therefore know it’s position relative to the origin. Therefore, the robot also always knew where the goal position was in relation to itself.

12. What should the subsumption architecture look like for the addition of the go-to-goal and avoid-obstacle behaviors?

For the subsumption architecture, the go-to-goal functionality would be a higher layer than the obstacle avoidance behavior. When this layer is enabled, we would want to make sure that the robot’s movements always trends towards the goal, but also avoids any local obstacles as well. Therefore, in the subsumption architecture, we would want to sum the result of the avoid-obstacle behavior and the go-to-goal behavior. Also, if there is a goal, we would want to suppress the result of the random-wander behavior, so that the final output of the architecture is just the sum of the go-to-goal and the avoid-obstacle behaviors.

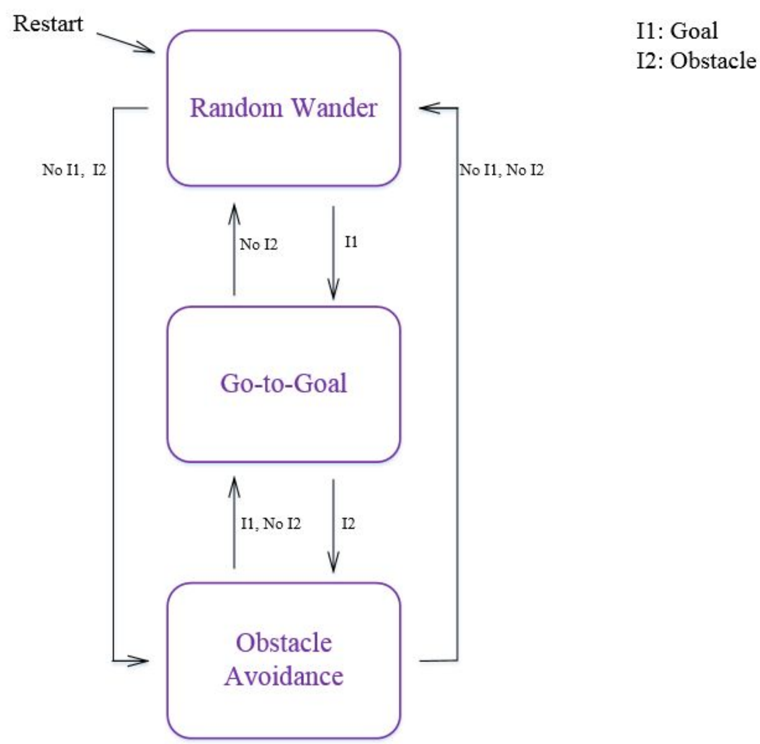
Code Advancements

With this lab, we found a new IDE to use, called “sloeber.” This IDE is an eclipse extension that has allowed us to create header files and independent files to read and run the IR and Sonar sensors, drive the robot and define the pins. This will help us stay organized with our code, help simplify debugging and let us call specific files to achieve desired tasks. We also started using GitHub and SourceTree so that we can independently work on the code and easily share updates with each other.

Appendix:

Pseudo code:

<u>State Diagram:</u>



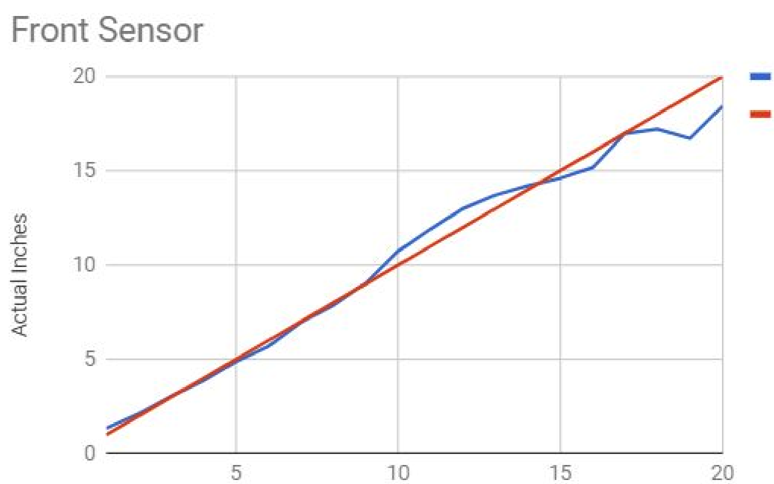
<u>Transition Table:</u>

State Transition Table (Robot Perspective)			
Robot Random Wander and Obstacle Avoidance			
States	Input	Output	Next Step
Random Wander	Goal		Go-to-Goal
	No Goal, Obstacle		Obstacle Avoidance
Go-to-Goal	No Goal	Goal Achieved	Random Wander
	Obstacle	Goal Not Achieved	Obstacle Avoidance
Obstacle Avoidance	Goal, No Obstacle	Obstacle Avoided	Go-to-Goal
	No Goal, No Obstacle	Obstacle Avoided	Random Wander

Calibration Data and Graphs:

Front IR Sensor

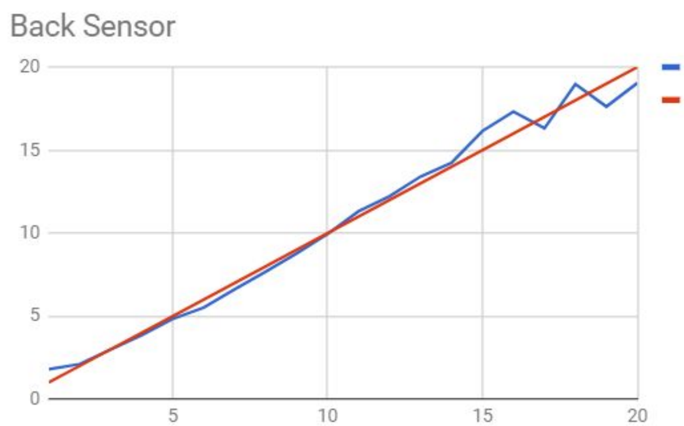
FRONT IR			
Actual Inches	Analog Values	Measured Inches	Percent Error
1	674.717	1.348	34.780
2	470.960	2.118	5.890
3	342.970	3.046	1.533
4	275.293	3.864	3.394
5	220.040	4.877	2.455
6	188.960	5.685	5.254
7	153.889	6.947	0.762
8	135.030	7.864	1.696
9	116.242	9.035	0.389
10	95.980	10.730	7.301
11	85.192	11.904	8.219
12	76.818	13.000	8.329
13	72.040	13.716	5.506
14	69.000	14.213	1.519
15	66.758	14.602	2.654
16	63.677	15.172	5.178
17	55.242	16.976	0.140
18	54.273	17.211	4.385
19	56.354	16.715	12.026
20	49.606	18.433	7.834
		RSME:	**0.9845**
m	1280		
b	18		
k	0.5		



Equation: $y = \frac{1280}{Analog + 18} - 0.5$

Back IR Sensor

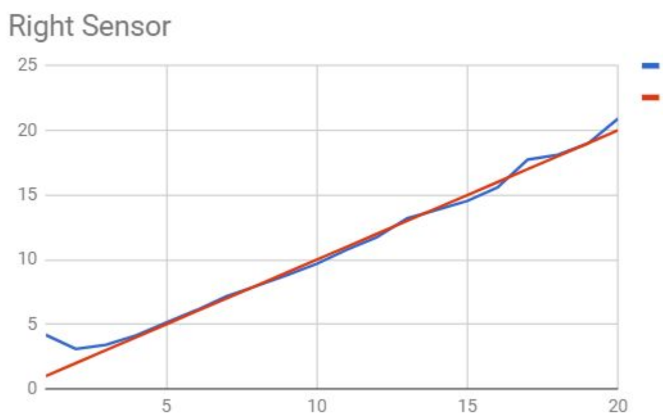
BACK IR			
Actual Inches	Analog Values	Measured Inches	Percent Error
1	593.980	1.803	80.334
2	505.245	2.110	5.517
3	351.041	2.997	0.102
4	269.816	3.849	3.784
5	211.408	4.837	3.258
6	183.367	5.517	8.042
7	150.429	6.609	5.579
8	127.429	7.669	4.133
9	109.265	8.781	2.429
10	94.653	9.941	0.590
11	81.184	11.319	2.898
12	74.041	12.217	1.806
13	66.061	13.405	3.113
14	61.245	14.240	1.717
15	52.122	16.147	7.649
16	47.531	17.314	8.216
17	51.429	16.314	4.038
18	42.000	18.966	5.364
19	46.429	17.620	7.262
20	41.755	19.046	4.770
		RSME:	**0.9867**
m	1100		
b	16		
k	0		



Equation: $y = \frac{1100}{Analog + 16} - 0.0$

Right IR Sensor

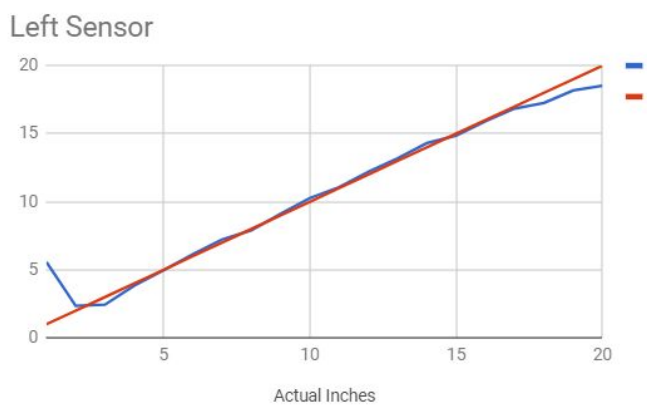
RIGHT IR			
Actual Inches	Analog Values	Measured Inches	Percent Error
1	500.449	4.181	318.052
2	664.061	3.095	54.747
3	604.653	3.417	13.905
4	505.041	4.140	3.494
5	413.061	5.144	2.886
6	354.204	6.090	1.498
7	306.408	7.158	2.263
8	278.776	7.966	0.419
9	256.041	8.782	2.420
10	235.306	9.687	3.133
11	215.102	10.767	2.114
12	200.102	11.740	2.169
13	181.837	13.190	1.463
14	174.592	13.870	0.929
15	168.082	14.543	3.044
16	159.102	15.587	2.580
17	143.980	17.731	4.297
18	141.673	18.110	0.613
19	136.694	18.988	0.061
20	127.286	20.904	4.518
		RSME:	**0.9828**
m	1950		
b	-34		
k	0		



Equation: $y = \frac{1950}{Analog - 34} - 0.0$

Left IR Sensor

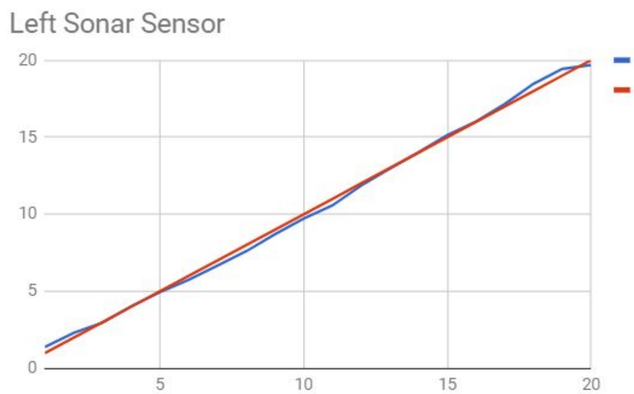
Left IR			
Actual Inches	Analog Values	Measured Inches	Percent Error
1	374.102	5.574	457.381
2	666.347	2.358	17.913
3	653.122	2.444	18.545
4	491.837	3.838	4.039
5	408.327	4.971	0.571
6	346.816	6.134	2.236
7	303.041	7.230	3.280
8	280.939	7.903	1.213
9	247.714	9.123	1.365
10	222.510	10.269	2.694
11	207.612	11.066	0.596
12	189.245	12.202	1.679
13	175.449	13.194	1.491
14	161.939	14.310	2.213
15	155.959	14.858	0.948
16	145.408	15.920	0.498
17	137.163	16.849	0.891
18	133.796	17.256	4.134
19	126.694	18.176	4.339
20	124.102	18.534	7.332
		RSME:	**0.9679**
m	3000		
b	22		
k	2		



Equation: $y = \frac{3000}{Analog + 22} - 2.0$

Left Sonar Sensor

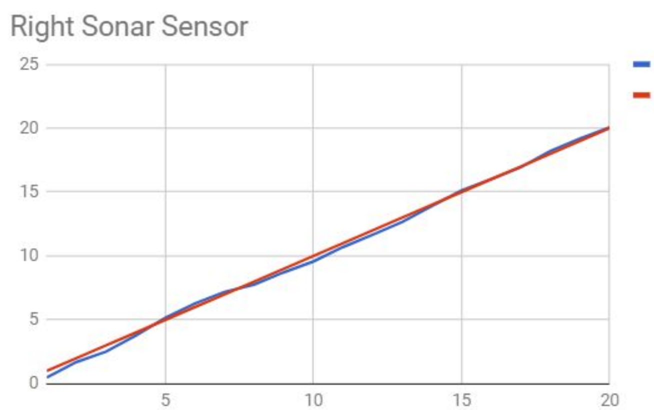
Left Sonar			
Actual Inches	Analog Values	Measured Inches	Percent Error
1	332.640	1.395	39.501
2	459.840	2.311	15.542
3	552.000	2.974	0.853
4	699.080	4.033	0.834
5	827.360	4.957	0.860
6	938.480	5.757	4.049
7	1065.920	6.675	4.648
8	1194.320	7.599	5.011
9	1348.360	8.708	3.242
10	1489.880	9.727	2.729
11	1607.354	10.573	3.882
12	1786.320	11.862	1.154
13	1939.880	12.967	0.253
14	2087.440	14.030	0.211
15	2242.680	15.147	0.982
16	2362.720	16.012	0.072
17	2520.880	17.150	0.884
18	2703.760	18.467	2.595
19	2838.360	19.436	2.296
20	2872.040	19.679	1.607
		RSME:	**0.9978**
m	0.0072		
b	-1		



Equation: $y = 0.0072 \cdot Analog - 1$

Right Sonar Sensor

Right Sonar			
Actual Inches	Analog Values	Measured Inches	Percent Error
1	211.840	0.483	51.712
2	386.360	1.705	14.774
3	502.440	2.517	16.097
4	680.360	3.763	5.937
5	879.680	5.158	3.155
6	1041.040	6.287	4.788
7	1169.960	7.190	2.710
8	1252.720	7.769	2.887
9	1388.800	8.722	3.093
10	1510.560	9.574	4.261
11	1671.200	10.698	2.742
12	1809.800	11.669	2.762
13	1952.760	12.669	2.544
14	2131.600	13.921	0.563
15	2302.640	15.118	0.790
16	2431.520	16.021	0.129
17	2563.000	16.941	0.347
18	2744.320	18.210	1.168
19	2883.440	19.184	0.969
20	3008.880	20.062	0.311
		RSME:	**0.9986**
m	0.007		
b	-1		



Equation: $y = 0.007 \cdot Analog - 1$