Date: 12/29/2014
To: Dr. Carlotta A. Berry
From: Alexandre van der Ven de Freitas and Peter Richard Olejnik A.V.D.V.F, P.R.O.
Robot: Capek
Subject: Lab3 - Random wander, Avoid obstacle

Introduction

The purpose of this lab is to program the robot to respond to inputs of the environment provided by 4 IR sensors, which depending of the group of sensors triggered, produces different behaviors initially set by the user.

The Robot's Program

The program performs the following tasks: sets the robot into the aggressive and shy kid modes, sets the robot into random wander mode and random wander mode that avoids obstacles and go to point mode in which avoids obstacles. To be able to perform this list of actions our code is divided into: the main file which is called lab3demo.c, which contains the functions main, shy() and aggressive() for the kids behavior when they are selected. The main function basically manages the menu selection from the LCD display and call the local functions as well as foreign from the .h files included. aggressive() basically reads the front IR sensor and keeps moving forward while the distance of the robot to a front obstacle is 5 inches or more, else the robot stops. shy() sets a 5 inches limit distance where the robot remains stationary. If a moving obstacle approaches one or more of the IR sensors and the distance from the obstacle is smaller than 5 inches, the robot moves away. The movements are described by the function Movement_Selector_Executor() located in Movement_Selector.h. An important file which contains functions that supports the previous mentioned function and most of the functions of Capek's code is Capek_Movements.h. Another important file that contains all the IR sensor functions is IRlib.h. Both functions Random_Wanderer() and Shy_Random_Wanderer(), which contains the random wanderer that acts according to the IR sensors' environment inputs, are located in Random_Wanderer.h. goToPoint.h contains the go2Point() and go2Angle() functions where go2Point() calls go2Angle() to be used to select the proper direction the robot should move. On go2Point() when the robot sees an obstacle, it stops the step counter turns 90 degress to the left and moves straight for a certain amount of pre-defined steps then again turning -90 degrees to return to its previous direction moving forward again using the step counter. After the robot does not detect the obstacle from one of the side IR sensors, it stops the step counter, turns -90 degrees, moves the same amount of given steps and then moves 90 degrees again moving forward counting the number of steps again until reaching the desired point. For more details on how the robot works see Figure 1.

Testing the Program

The program was tested and developed in a linear manner where each one of the goals were coded and then tested until satisfied. Both the aggressive and shy behaviors were coded, where the success was immediate. The same success rate came with the development of the

random wanderer and the random wanderer that responds to the environment. The longest and hardest task occurred on the development of the go to point with obstacle avoidance. The biggest difficulty was caused by the fact we didn't have an encoder which would make it possible to count the distance covered, also the psychological human factor where things got a little over-complicated when they shouldn't be. The solution, as previously mentioned, was a routine whenever the robot is moving in the direction towards the goal point it decrements the amount of steps to the distance to be covered and maintaining the amount of steps the same when the robot is adjusting its direction during obstacle avoidance.

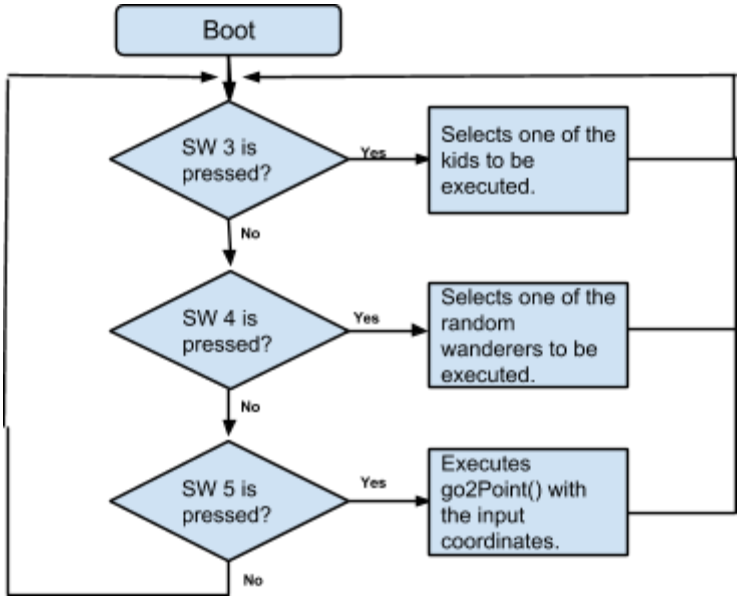


Figure 1 - Robot program flowchart.

Sensor Data

When first inspecting our sensors, we collected data on the output value, versus what the actual values were. These are our results (all in inches):

Target	Front IR	Right IR	Left IR	Back IR
1	1.4011	1.46	1.514	1.518
2	2.11	1.494	1.544	1.504
3	2.11	2.35	2.111	2.111
4	3.01	3.024	2.812	2.954
5	3.874	4.216	3.621	3.709
6	4.497	4.924	4.49	4.451
7	5.129	5.813	5.403	5.101
8	5.919	6.827	6.56	5.981
9	6.788	7.74	7.418	6.874
10	7.524	8.234	8.049	7.746
11	8.043	9.501	9.01	8.501
12	8.933	10.13	9.67	9.091

Discussion of Questions

1) To implement our random wanderer, we utilized the rand() function twice, once for each wheel. The results where then input as the robot's wheels speed. This would be repeated every 0.5 seconds. This resulted in rather unpredictable movements on the robots parts.

   The obstacle avoidance behaviors generally functioned on the principle that the robot had a default action which it performed. However, if it detected an object within range, the robot would then go and perform another action. This action varied depending on what sort of behavior we were looking to have the robot perform.

2) The modularity of our program is most evident in our generous usage of functions. Depending on what actions we were trying to perform, the set and order of functions we would call and the conditions calling them varied.

3) We only utilized the IR sensors. The usage of the contact sensors, while it would have given redundancy, was not needed in the end.

4) To cover the entire room, you could create a systematic program which would go to every point in the room.

5) When first trying to implement our obstacle avoidance, we tried to create a smooth behavior to avoid the obstacle. That didn't work out too well. What we found was that to create something in a reasonable amount of time, we needed to make more deliberate avoidance actions, at the cost of slick looking movements.

6) Improvement of the obstacle avoidance could come from a few sources. From a hardware side, by increasing the the number of sensors in more than the four places currently. From a software side, the avoidance movement could have a smoother flow to it.

7) Any obstacle that was outside the sensors range could not be detected. For example, while testing, we had couches in the room we were working in; just high enough for the sensor to miss but low enough for the robot to hit.

8) We didn't experience any situation where the robot had difficulty detecting an obstacle (when it could have). All the surfaces we worked with reflected the IR quite well.

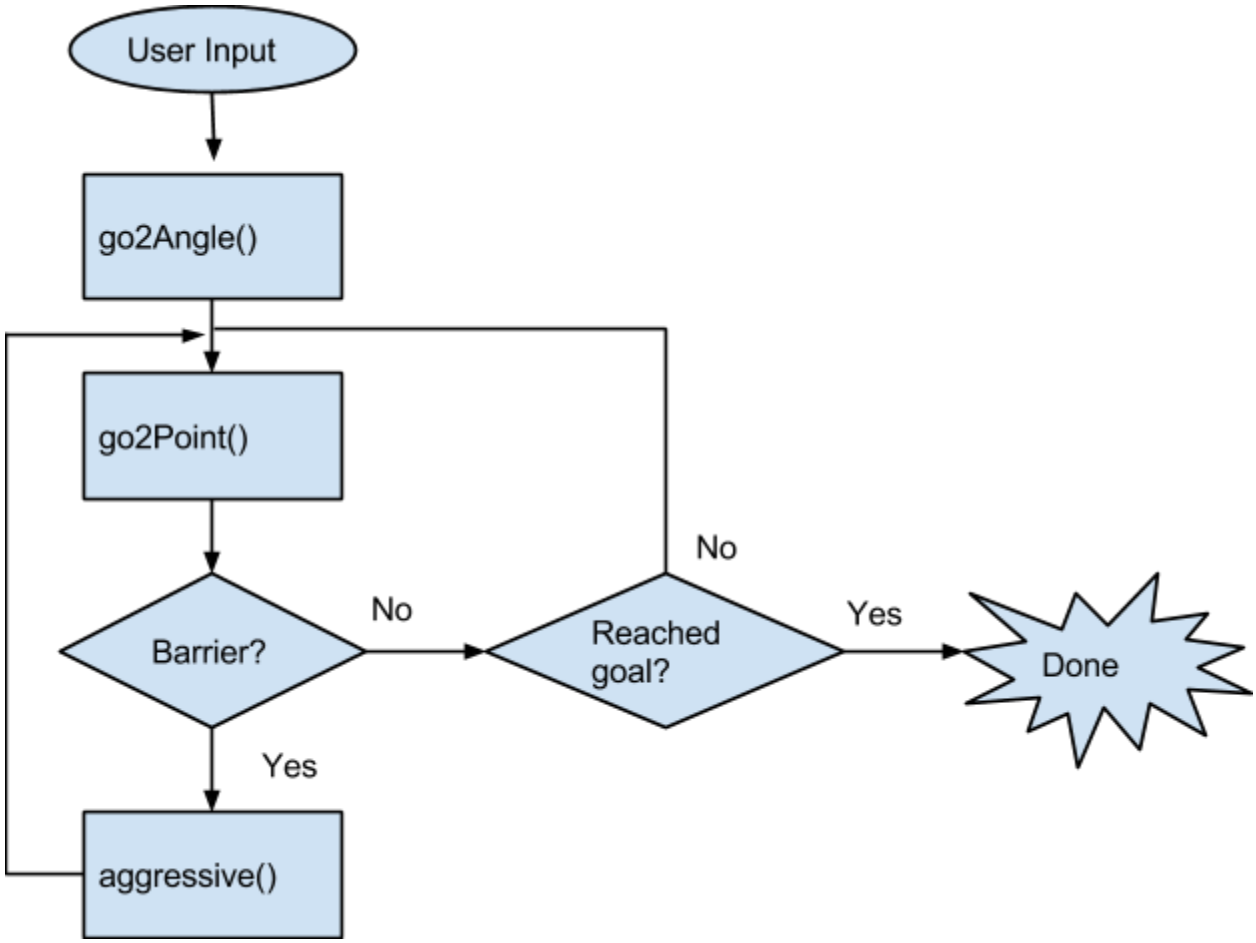
9) To keep track of the robots state, a generous usage of the LCD was used.

10) The stuck situations we encountered mainly came from our first attempt to have the go to goal and avoidance behavior together. This was the result of our attempt to have the movement be fluid, very universal, and not very condition specific. Due to

limitations on time and moral, this did not work out. What this resulted in were far more deliberate in looks movements.

11) To keep track of our position in the go to goal as the avoid obstacles took place, what resulted was the deliberate usage of the number of steps in the movements in the mobot. By specifying the number of steps, we could know exactly where the robot was in the world.

12) The answer to this question can be exemplified in the following diagram:



Conclusions

This lab took much longer than expected. This was in part to a large number of new things for us. New hardware for one. Also new was that both of us were coding at the same time, where previously only one of us was doing the actual coding.

What also burnt a lot of our time was that on the go to avoidance behavior we tried to have a very general and fluid movement of the robots avoidance. However, this did not work out well for us in the end, costing us a good six or so hours.

We are looking forward to the wall following, as we are confident we have the basework in place to have successful results.