SOFTWARE ALGORITHMS DOCUMENT

Project: ECSE 221 DPM 'Capture the Flag' Robot

Task: To construct an autonomous robot that navigates to its opponents specified area in search for a specific coloured block through a 12'x12' enclosed area filled with randomly placed obstacles and then to deposit the block on a specific square area 1'x1' in under the constraint time.

Document Version Number: 1.0

Date: 2015/11/15

Authors: Asher Wright

TABLE OF CONTENTS

1. OBJECTIVE OF DOCUMENT

The objective of this document is to outline the various algorithms that were considered for completing each subtask of the project. The four subtasks are ultrasonic localization, light sensor localization, navigation and obstacle avoidance, and block searching.

2. TASK 1: ULTRASONIC LOCALIZATION

Algorithm 1,2: Falling edge or Rising edge

These two algorithms were used in the course labs. Falling edge works by rotating the robot until it doesn't see a wall, then until it sees a wall. This angle is called angle A. It then rotates until it doesn't see a wall. This angle is called angle B. It can then use simple geometry to figure out its initial orientation. Rising edge is the same, but it faces away from the corner.

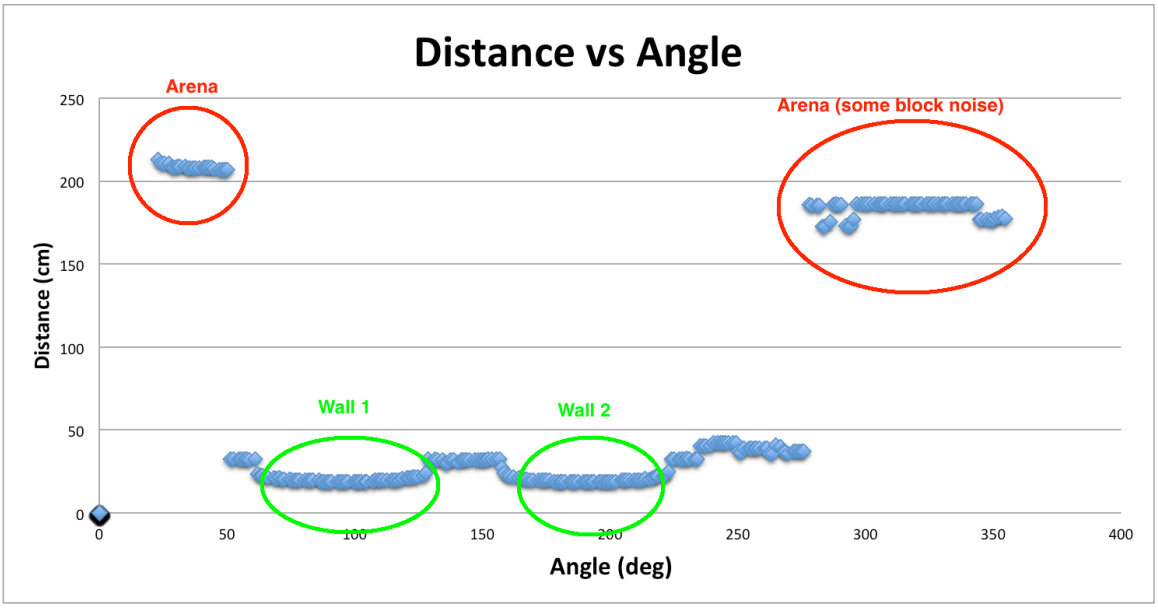
The issue with these algorithms is they do not perform as well when there are blocks in the arena. In this case, the robot will think that the block is the beginning of the wall. One can hard-code minimum distances to prevent this, but it can get messy.

Algorithm 3: 360-degree profile

The algorithm works by taking a profile of the location around the robot. The robot rotates 360 degrees very quickly. During the rotation, it records all of the distance measurements it takes (every 25ms). After rotation, the robot has a profile of what is around it. It can then effectively determine what part of the profile corresponds to the wall, and what part corresponds to the arena. A brief explanation is given below:

The robot will look for the absolute minimum distance value. It calls this index one of the walls. To find the other wall, it looks for a local-minimum that has a local-maximum between it and the first index. It then calls this index the other wall. To figure out which index corresponds to which wall, it finds the difference between the first wall's index and the second wall's index. It does this both by circling around the array, and by subtracting the two directly. If circling around is less, then the second wall is along the X-axis. Otherwise, the first wall is along the X-axis.

A sample data profile looks like this **(see US-360-Profile.xlxs)**:



Preliminary tests of this method showed its promise (faster and more accurate with blocks), but we will need to test it thoroughly and record these tests before using it instead of falling edge.

3. TASK 2: LIGHT SENSOR LOCALIZATION

Algorithm 1: Two light sensors

With two light sensors, the algorithm is fast and simple. The two sensors are placed on the left and right side of the robot. The robot drives forwards until it hits the black line with one sensor, and then rotates until it hits the line with the other sensor. It then does this again after rotating 90 degrees (for the y=0 black line). It also uses these black lines to update its position.

The biggest drawback to this method is the use of two sensors. We chose to have two ultrasonic sensors instead, and thus used algorithm 2.

Algorithm 2: One light sensor

With only one light sensor, we use an identical algorithm to that used in the lab. That is, the robot spins around 360 degrees, and records the angles at which it hits the four black lines. Using these angles, it can calculate the robots displacement and orientation.

4. TASK 2: NAVIGATION AND OBSTACLE AVOIDANCE

Algorithm 1: Navigation with P-controller wall follower

The robot drives towards the end position. If it sees a wall, it starts up the wall follower. The wall follower uses the 90-degree sensor and uses the logic from the P-controller wall follower. That is, depending on how far the robot is, and how far it wants to be, it corrects it with a speed proportional to the difference. The wall follower is stopped when the robot is closer to the end than the block. This was found to be pretty effective. However, the P-controller logic was optimized for a sensor at 45-degrees, and this caused some problems (collisions) with the use of the 90-degree sensor.

Algorithm 2: Navigation with right angle wall follower

Since we have sensors at 0-degrees and 90-degrees, we are able to create a more effective wall follower algorithm. This one rotates until the 0-degree sensor sees no wall. It then drives forward until the 90-degree sensor sees no wall. It then rotates until the 90-degree sensor sees a wall. It then drives forward until the 90-degree sensor sees no wall. This is a very simple algorithm to make small cuts around obstacles in the way. However, it is not as smooth as the p-controller.

Both of these algorithms need to be better tested before the team can decide on one to use.

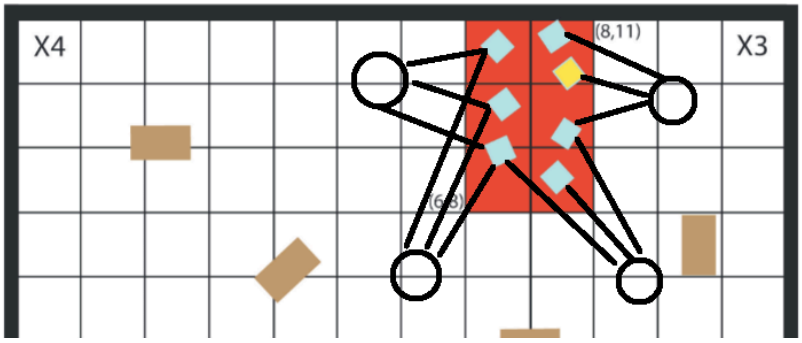
5. TASK 3: BLOCK SEARCHING

Algorithm 1: Two-point Triangulation Map and Go-to

This first algorithm works by mapping out the locations of the blocks, and then going to visit them one by one (in an optimized order). The robot travels to one position outside of the block-zone (where all the blocks are located), and takes a distance profile of the zone. It then travels to another position and does the same thing. Using the ultrasonic data from these two points, it can (ideally) determine where the blocks are located on that side of the zone. It could take distance profiles at more points if it does not find the block after the first two.

In theory, this is a good method. However, it is highly susceptible to noise and inaccuracies in the ultrasonic sensor. If one of the distances was off by 5cm at 50cm away, then the block could be totally missed (or the robot would think it was in a different position).
Additionally, this method would require a lot of planning and coding. This would likely go over-budget.

Here is a schematic of it mapping the blocks with four distance-profiles.



Algorithm 2: Perimeter search

This algorithm is much more simple. It starts the robot at one of the corners of the block-zone. It then drives around the perimeter of the zone, using the 90-degree ultrasonic sensor to detect blocks. It only detects blocks that are within 1 tile (to prevent double-counting blocks). After detecting a block, it rotates, drives forward to check if it is a flag, and then either picks it up or drives back and continues.

This algorithm takes advantage of the 90-degree sensor. There are block layouts that make it somewhat inefficient, as there could be no blocks in the first half that it searches. However, it still works in cases like this, and is more accurate than the first algorithm.