Real Time Navigation with the Lee Algorithm
Jaeyun Stella Seo
Polaris, 13-0535

Real-Time Navigation with the Lee Algorithm

I. Abstract

To find an unobstructed route in an unplanned autonomous robotic game, a real-time navigation system is needed. Our team improved the Lee algorithm to create a navigation system that implements the concept of dynamic blockages. After testing, this navigation program has proved to be beneficial.

II. Introduction

When a robot is programmed to move from Point A to Point B, programmers generally use a hard, nonflexible program like "go forward five units, turn right, go forward ten units." This rigid code does not consider unexpected obstacles on the field. Some tournaments allow human interaction with the robot, so the team can pull the robot out of an unexpected situation. However, Botball is an entirely autonomous game, and human assistance is not allowed. This means that a robot must have an intelligent navigation system, like a human. Our team has created a fluid, versatile navigation program that can move the robot from Point A to Point B, even if unexpected situations arise.

III. Fundamental Concepts

The basics of our navigation program come from the Lee algorithm [1]. The Lee algorithm finds the shortest path between two predetermined points. Based on this algorithm, we have created a navigation system that utilizes propagation and back-tracing in a game field with blockages.

The Lee Algorithm is an exhaustive search algorithm. It will find a path between two points on a coordinate grid, so long as such a path exists [1]. Through wave propagation, each coordinate is assigned a value. The starting point has a value of 0. The surrounding points are assigned values of 1. The points around the former points are assigned values of 2, and so on. See **Figure 2** for an example.

The playing field is divided into a 16x16 grid map, where each grid is 6x6 inches. The starting position, target position, known hard-blockages, and unknown dynamic blockages are labeled with A, B, H, and D respectively. In programming, we use negative values for these labels to distinguish them from the positive values in wave propagation. **Figure 1** shows an example of an initial grid.

H	H	H	H	H						H	H	H	H	H	H
										B					
									D						
H	H	H	H											H	H
	A													H	
														H	

Figure 1: This is an initial marking of a map before wave propagation. A is the starting point. B is the target point. H represents a hard blockage. D is an example where a dynamic blockage may be encountered.

The program begins propagation, utilizing the Lee algorithm. Each coordinate point north, east, south, and west of the starting point is assigned a value of one greater than the original point. The value represents distance from the starting point. This continues until the entire grid is filled. Note that the propagation does not override any grids with A, B, H, or D. **Figure 2** shows an example of the wave propagation.

4	**3**	**2**	**3**	**B**
3	2	1	2	3
2	1	**A**	1	2
3	2	1	2	3
4	3	2	3	4

Figure 2: This map shows how wave propagation works. It begins with grid A in the middle and works outward until every grid is assigned a value.

Once the map is filled, the program begins back-tracing from the predetermined target point B by finding the lowest value in the four grids surrounding it. A grid is chosen, and its value is changed to P. From this new grid, the program then checks its surroundings again for the next lowest number. If there are multiple grids with the same lowest value, priority is given to the grid that maintains the same direction. As the program decides its next grid, it replaces the current grid with the label P to indicate the path chosen. This process continues until the shortest path is traced to the starting grid A, as shown in **Figure 3**.

4	**3**	**P**	**P**	**B**
3	2	**P**	2	3
2	1	**A**	1	2
3	2	1	2	3
4	3	2	3	4

Figure 3: This diagram illustrates back-tracing. It starts from the target at B in the top-right corner and back-traces until it reaches the starting grid at A. The chain of grids marked P provides the path from A to B.

IV. Our Innovations

Our main innovation is the addition of dynamic blockages to the navigation map. It is a real-time navigation system, because we can add blockages as we encounter them.

As the robot moves around the game field, it keeps track of its exact coordinate on the grid. It also records the movement of each turn made. As the robot moves along its path, it may encounter unexpected obstacles, such as the other team's robot(s). If this happens, the robot knows the exact location of the unexpected obstacle. Upon detecting it with bumpers or touch sensors, the robot stops to avoid pushing the dynamic blockage into multiple grids. At the same time, it places a dynamic blockage in this grid location.

In the program, it replaces the current grid with a label D. The program repeats the wave propagation with the modified map. It finds a new path to its original destination, given the new blockage. This is very effective, because the robot is not disabled for the rest of the game simply because of an unexpected obstacle. The program is repeated if another dynamic blockage is encountered. This makes the robot tolerant of blockages in unexpected situations.

V. Experimental Results

Figure 4 is an example of an initial map. **Figure 5** is an example of a map after completing propagation. **Figure 6** is an example of a map that has created its route.

15	0	0	0	0	0	0	**H**	**H**	**H**	**H**	**H**	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	**B**	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**H**	**H**
2	**H**	**H**	**H**	0	0	0	0	0	0	0	0	0	0	0	**H**	0
1	0	**A**	0	0	0	0	0	0	0	0	0	0	0	0	**H**	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**H**	0
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Figure 4: In this initial map (outlined in the red box), the starting position is located at (1,1) and is marked by A. The target position is located at (10, 6) and is marked by B. The numbers outside the red box are coordinate references.

15	19	18	17	16	17	18	**H**	**H**	**H**	**H**	**H**	24	25	26	27	28
14	18	17	16	15	16	17	18	19	20	21	22	23	24	25	26	27
13	17	16	15	14	15	16	17	18	19	20	21	22	23	24	25	26
12	16	15	14	13	14	15	16	17	18	19	20	21	22	23	24	25
11	15	14	13	12	13	14	15	16	17	18	19	20	21	22	23	24
10	14	13	12	11	12	13	14	15	16	17	18	19	20	21	22	23
9	13	12	11	10	11	12	13	14	15	16	17	18	19	20	21	22
8	12	11	10	9	10	11	12	13	14	15	16	17	18	19	20	21
7	11	10	9	8	9	10	11	12	13	14	15	16	17	18	19	20
6	10	9	8	7	8	9	10	11	12	13	**B**	15	16	17	18	19
5	9	8	7	6	7	8	9	10	11	12	13	14	15	16	17	18
4	8	7	6	5	6	7	8	9	10	11	12	13	14	15	16	17
3	7	6	5	4	5	6	7	8	9	10	11	12	13	14	**H**	**H**
2	**H**	**H**	**H**	3	4	5	6	7	8	9	10	11	12	13	**H**	0
1	1	**A**	1	2	3	4	5	6	7	8	9	10	11	12	**H**	0
0	2	1	2	3	4	5	6	7	8	9	10	11	12	13	**H**	0
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Figure 5: The possible paths are marked by increasing numbers until the target point (ex. 1 to 2 to 3 to 4 until B is detected). Obstructed grids are marked with H.

15	19	18	17	16	17	18	**H**	**H**	**H**	**H**	**H**	24	25	26	27	28
14	18	17	16	15	16	17	18	19	20	21	22	23	24	25	26	27
13	17	16	15	14	15	16	17	18	19	20	21	22	23	24	25	26
12	16	15	14	13	14	15	16	17	18	19	20	21	22	23	24	25
11	15	14	13	12	13	14	15	16	17	18	19	20	21	22	23	24
10	14	13	12	11	12	13	14	15	16	17	18	19	20	21	22	23
9	13	12	11	10	11	12	13	14	15	16	17	18	19	20	21	22
8	12	11	10	9	10	11	12	13	14	15	16	17	18	19	20	21
7	11	10	9	8	9	10	11	12	13	14	15	16	17	18	19	20
6	10	9	8	**P**	**P**	**P**	**P**	**P**	**P**	**P**	**B**	15	16	17	18	19
5	9	8	7	**P**	7	8	9	10	11	12	13	14	15	16	17	18
4	8	7	6	**P**	6	7	8	9	10	11	12	13	14	15	16	17
3	7	6	5	**P**	5	6	7	8	9	10	11	12	13	14	**H**	**H**
2	**H**	**H**	**H**	**P**	4	5	6	7	8	9	10	11	12	13	**H**	0
1	1	**A**	**P**	**P**	3	4	5	6	7	8	9	10	11	12	**H**	0
0	2	1	2	3	4	5	6	7	8	9	10	11	12	13	**H**	0
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Figure 6: The robot will follow the path marked by P, the shortest path. New blockages are automatically added if the robot is interrupted during its run and the robot will automatically recalculate the path, as explained in Section IV.

VI. Further Works

The real-time navigation program has aided us greatly, but some improvements can still be made. There are a two known issues.

Even though the navigation program always finds the shortest path, it is not always the "best" path. **Figure 7** exhibits a such a situation. The shortest path, represented in yellow, is only five units long. The longer path represented in red is seven units long, but is actually more effective. The yellow path requires the robot to turn three times before reaching its destination. These turns cost time and are not necessarily perfect 90 degree turns, resulting in slight error. The red path, though longer, is more effective because it requires only two turns. We continue to develop our program to find the best path by assigning a cost value to turns. This allows us to make the trade-off between path length and turns, ending up with the best path.

P'	P'	P'	P'
P'		H	B
P'	H	P	P
A	P	P	H

Figure 7: This is an example in which the shortest path is not the best path.

Another known problem involves mechanical tolerance. When the robot moves, it gains momentum. If the robot stops abruptly, the momentum carries it forward more than planned. This additional displacement can create problems later as it accumulates, eventually causing navigation failure. For example, if a robot were programmed to move forward and stop at a grid, it may move forward 1/5 grid more. If this were repeated five times in the same direction, the

robot would have moved one full grid extra. The robot is no longer in the correct grid, so the rest of the program becomes irrelevant. If this additional movement were consistent, we could compensate for it in the program. Unfortunately, we observed that it is not. To alleviate this problem, we will change our move function to accelerate and decelerate gradually. This will result in less jerky stops and less momentum, allowing the robot to remain in its desired grid.

The next improvement to the navigation system is the application of the ET sensor, which is a range distance sensor. It detects distance to an object by sending and measuring the reflection of a modulated frequency IR beam. We use the ET sensor to scan for the presence of blockages, the size of said blockages, and distance from the blockage. With the ET sensor, we can detect dynamic blockages without colliding into them and potentially disturbing the robot's position. Using this information, we could place dynamic blockages on our map and find an effective path more quickly.

We also hope to apply this navigation system into the third dimension of height. We have added a third layer in our navigation program, but we have not yet implemented it. We hope this would improve accuracy when using an arm to grab objects that are not on the ground.

VII. Conclusion

Using the Lee algorithm, we have designed a program that aids in robot navigation. Besides the basic navigation function of finding the shortest path on a map with preset blockages, we have improved it to be more humanlike so that it can respond to unexpected situations. When the robot bumps into an object, it sets its current location as a dynamic blockage and finds a new route to its destination. This has been experimentally proven to be very effective. In the future, we would like to improve the program to find the "best" path instead of the shortest, mediate start-stop problems, use the ET sensor, and apply this precise-movement program to the third dimension.

References

[1] Zhou, Hai. "Maze Router: Lee Algorithm." Lecture. *Hai Zhou's Personal Webpage*. Web. 20 May 2013. <http://users.eecs.northwestern.edu/~haizhou/357/lec6.pdf>.