CENG786 - Spring 2018
Homework II - Sensor Based Coverage

JeanPiere Demir
Middle East Technical University
jeanpiere.demir@gmail.com

Abstract

One of the biggest problem for the mobile robots is the information about the environment and if there is not any remote sensor such as camera or any other robot, this information can not be obtained no other than mobile robot itself. For example, industrial, mine finder or home cleaning mobile robot can be very good candidate for this situation. If robot encounters with the environment for the first time, it has to cover/map the whole environment for current/further missions by using own sensing capability. This type of mapping algorithms are called **sensor-based coverage** and basically sensor-based coverage uses mobile robot own sensor information to map/cover the whole environment.

Keywords sensor based coverage, mobile robots, trapezoidal decomposition

1 Introduction

For robotic platforms where the robot has to traverse the fixed environment again and the again, the information of environment is crucial and has to be known for mobile robot current/further missions. Mine detection, floor cleaning robot can be given as a example for that kind of platforms. When robot encounters with the environment for the first time, the overall task can be specified as a complete mapping/coverage of the environment [1],[2]. Moreover, path planning of environment covering is simply a path which leads a robot to scan over all points in environment free space [3],[4].

In this work, we are using trapezoidal decomposition for sensor based coverage and our robot is 2D circular robot. I gave brief information about the sensor based coverage with trapezoidal decomposition and cycle algorithm in Section 1.1„ then I explained how I implemented this method in computer environment in Section 2. In Section 3, I explained the result of my implementation with regards to different obstacles, different starting point and clarified my implementation deficiencies. Finally, I summed up everything in Section 4.

1.1 Sensor-Based Coverage

Sensor-based coverage can be done by using several different methods. In this work, I used one of the popular method to map the environment which is exact cell decomposition. Exact cell decomposition represents the free space by bunch of cells which are non-overlapping sliced environment regions [5].

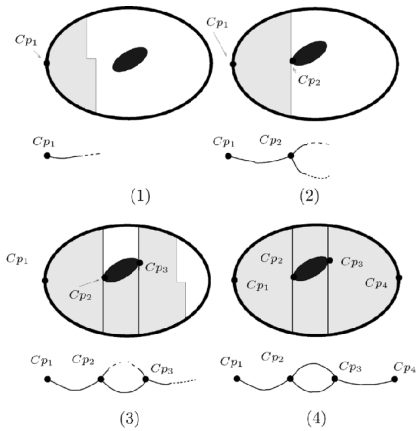

Figure 1: Sensor-based coverage has different phases and this figure tries to illustrate these phases. To add more,there used morse decomposition as a decomposition method. The most important point of this figure is that as robot covers the environment the graph is growing [2]

Moreover, these cells have some important properties such as,

- The interior of each cell intersects no other cell
- Union of all of the cells fills the free space
- Adjacent cells share a common boundary

1.1.1 Trapezoidal Decomposition

Trapezoidal decomposition also known as vertical decomposition is a decomposition for Polygonal World environments. It decomposes the environment according to vertices and every vertex creates two adjacent cells which unions are the free space. The whole map can be covered properly by developing an algorithm which uses this property [6].

1.1.2 Cycle Algorithm

While decomposing the environment, there exist some narrow passages and these passages can not be covered sometimes. Cycle algorithm ensures that the robot will find every critical point while executing the coverage [2],[4]. In my implementation, I could not apply cycle algorithm because I could not figure out any way to combine my implementation with cycle algorithm.

2 Implementation

In this section, I explained the detail of my own implementation. For the sake of simplicity, I divided the script to three parts which are Motion-to-Boundary, Border-Boundary Following and Obstacle-Boundary Following. In my implementation, finding the critical point and constructing the graph are embedded to these three parts. So, I started with explaining how I found critical points and constructed the graph. Thereafter, I explained Motion-to-Boundary part and lastly I explained Boundary-Following parts.

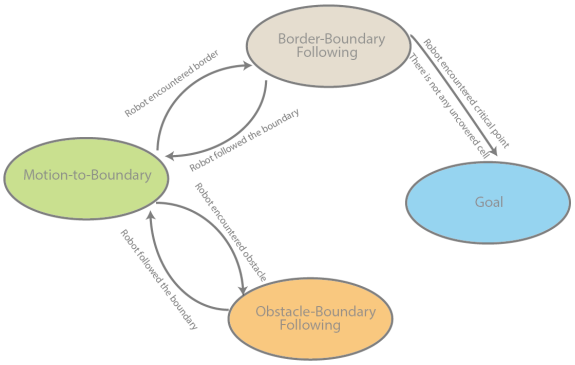

Figure 2: I tried to implement Sensor-based coverage as a Finite State Machine. This FSM is consisted of Motion-to-Boundary, Border-Boundary Following and Obstacle-Boundary Following states and they are explained in 2.3,2.4 and 2.5 respectively. The arrows try to explain transition conditions and if transition conditions occurs where the machine will pass as a next state. Next state is pointed by the arrow.

2.1 Critical-Point Finder

In my opinion, finding the critical point was one of the challenging part of this work. I could not generalize finding the every critical point, so I divided and embedded finding the critical points to other parts which are explained in Section 2. The first critical point of the obstacle is found in Motion-to-Boundary part because the robot can easily detect the obstacle starting point by checking normal vector of its motion according to scanning direction. To find other critical points, the robot sends there different sensor ray whenever it encounters the boundaries. I called these three different sensor as leading, center, lagging ray and the center ray has equal angle difference to lagging and leading rays. The illustration of finding critical points are showed in Figure 3-4.

The robot calculates three different points where the boundary and sensor rays encounter, hp_{lag},hp_{lead} and hp_{cent}. Thereafter, it obtains two different vectors according to this points. The first vector($\vec{line_1}$) is passing through leading and center point, second vector($\vec{line_2}$) is passing through center and lagging point. After that, as you can see from equation 1 the robot finds the dot product of these two vectors,

$$result = \vec{line_1} \cdot \vec{line_2} \tag{1}$$

and checks the $result$ value, to understand either it is on critical point or not. As you know, dot product means geometrically the product of the Euclidean magnitudes of the

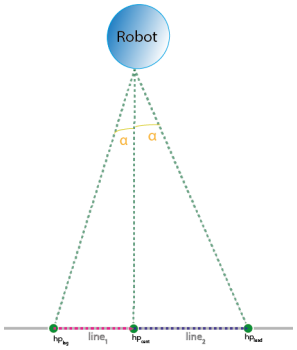

Figure 3: Whenever the robot encounters the boundary, it sends three different rays to the boundary. As you can see, if three calculated points are composing kind of straight line that means the current position is not any critical point.

two vectors and the cosine of the angle between them or simply,

$$\vec{A} \cdot \vec{B} = \left|\vec{A}\right|\left|\vec{B}\right| cos(\angle(\vec{A}, \vec{B})). \tag{2}$$

The conditions are explained as showed in below conditional equation.

$$\begin{cases} result \approx \left|\vec{line_1}\right|\left|\vec{line_2}\right| & do\ nothing \\ otherwise & add\ pos_{curr}\ to\ C \end{cases} \tag{3}$$

Where C is critical point array and it stores the (x,y) coordination of critical points. If first condition holds that means the robot is on straight line because angle between the vectors is zero, Fig. 3. If second condition holds that means the robot is on uneven line and current position is a critical point, Fig. 4

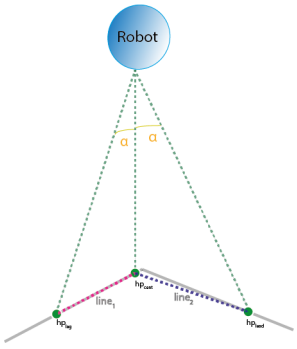

Figure 4: The robot calculates three different points in consequence of sending three different sensor ray. As you can see, $\vec{line_1}$ is composed of hp_{lag} and hp_{cent}, $\vec{line_2}$ is composed of hp_{cent} and hp_{lead}. If the angle between these two vectors is not zero, that means the current position is a critical point.

2.2 Constructing the Graph

After finding the critical points, constructing the graph is becoming a bit trivial. The array of graph is created separately from critical point array but they are constructed at the same time. Whenever the robot encounters new critical point it adds one to number of critical points and this number is becoming the new node identity number. Later on, the robot checks which node is last visited and it creates an edge between last visited node and current node. The illustration of one of the graph can be seen in Figure 5.

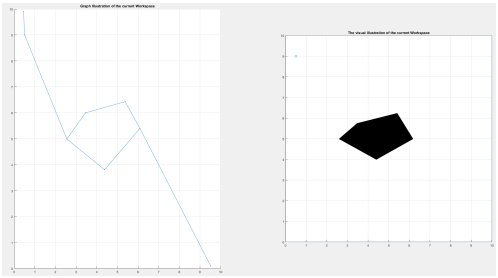

Figure 5: Graph nodes are corresponding to their actual position, and as you can see both figures is almost same. The robot started from [x,y] = [0.5,9]

2.3 Motion-to-Boundary

In this part, the robot moves in the up-down direction and behaves as a swipe line. Whenever it encounters to the boundary it passes to the other states. If the robot has not encountered the obstacle yet, the robot checks normal vector of its motion according to scanning direction to find the starting point of the obstacle and only alters between this state and Border-Boundary Following state. When the robot encounters the obstacle for the first time, it changes this behavior to check it is encountered to the obstacle or border by checking min/max vertical boundary values of the environment which are found by the robot. Moreover, the robot starts to alter between this state, Border-Boundary Following and Obstacle-Boundary Following states. The illustration of this part can be seen in Figure 2.3.

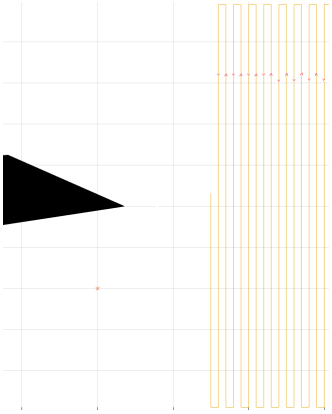

Figure 6: Motion-to-Boundary is consists of two different motion direction, up and down. The arrows show the motion direction.

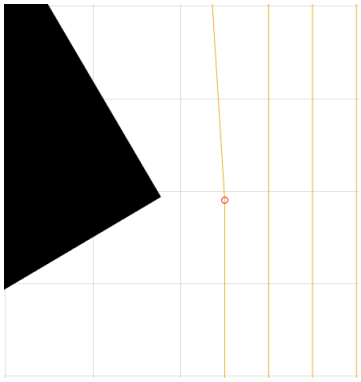

Figure 7: Red circle corresponds to first critical point of obstacle. As you can see, the encountered position stored as a critical point because when the robot.

2.4 Border-Boundary Following

In this part, the robot follows the border boundary according to scanning direction and later on switches back to the Motion-to-Boundary state. The robot checks through the moving direction and if the robot senses something in the moving direction, it decides that this point is one of the border critical point and it adds this point to the critical point and graph array. In other words, the robot is only considering and finding critical points of left-right hand side border boundary. Thereafter, it moves to previous node and continue to scanning from the down side of the obstacle because in my implementation the robot scans up-prior. The illustration of this part can be seen in Figure 8.

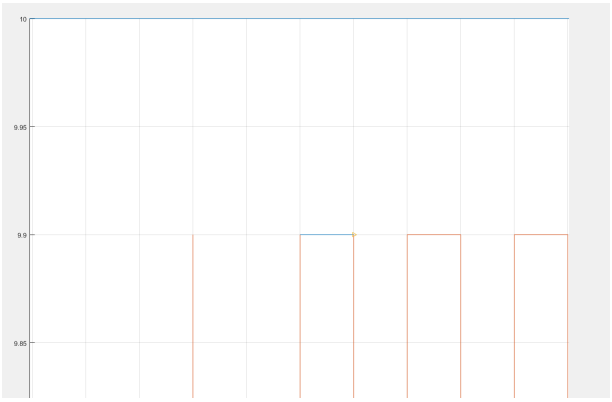

Figure 8: The blue line is showing us the followed path which is executed by the robot while it is in the Border-Boundary Following state.

2.5 Obstacle-Boundary Following

The behavior of the robot in this state/part is almost close to Border-Boundary Following behavior which is explained in Section 2.4. The main difference between them is that this part is considering all critical points on the obstacle. I separated border and obstacle following because I want to simplify the implementation and the robot can understand easier that if the robot finishes the scanning of the obstacle or not. The illustration of this part can be seen in Figure 2.5

3 Results & Discussions

In this part, I showed you results of my implementation for different obstacles and starting position. In the figure 10, I tried to give you the illustration of behavior of the robot for different states. Moreover, I explained deficiencies of my implementation

As you can see from Figure 5 and 11, the robot can finds the graph even the obstacles have different number of edges.

Sometimes the robot can have different starting positions inside the environment. So, I started the robot from the right side of the environment and showed you the result in Figure 12. As you can see, the implementation is working well for this situation and the robot can cover the environment precisely. Moreover, I tested the implementation when the

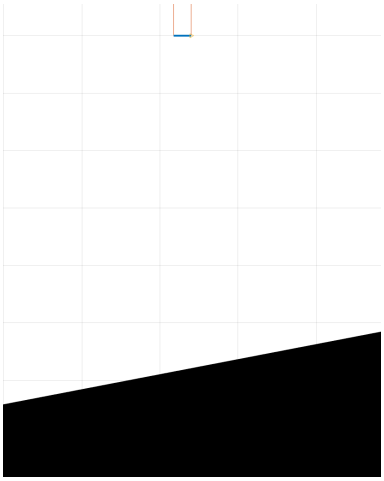

Figure 9: The blue line is showing us the followed path which is executed by the robot while it is in the Obstacle-Boundary Following state and as you can see, the robot maintains a safe following distance to the obstacle.

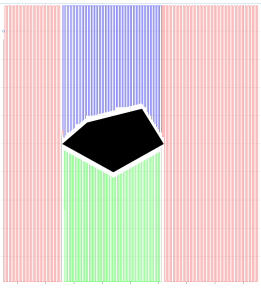

Figure 10: Motion of the robot is showed with different colors. Red,blue and green lines indicating respectively behavior of the robot in Motion-to-Boundary state, Obstacle-Boundary Following state when the obstacle is scanned from up direction, Obstacle-Boundary Following state when the obstacle is scanned from down direction.

environment is consisted of two side-by-side obstacles. The robot covers the environment accurately and it connects two obstacles properly. The illustration of this case can be seen in Figure 13.

3.1 Deficiencies

There exist some deficiencies in my implementation because in the start I did not implement uncovered cell/node finding based implementation. I directly focused on critical points finding and later on I combined with recognizing cell is covered or not. Next time, I will be much more prepared for this kind of problem. The deficiencies are,

- The robot can not cover concave critical points because geometrically I distinguished concave/convex critical points but my implementation can not recognize if there left uncovered cell or not for concave obstacles. The illustration can be seen in Figure 14.
- The robot can not cover the passages between obstacles if obstacles are one on top of the other. The main reason of this problem is again not recognizing the uncovered cells precisely. The illustration can be seen in Figure 15.

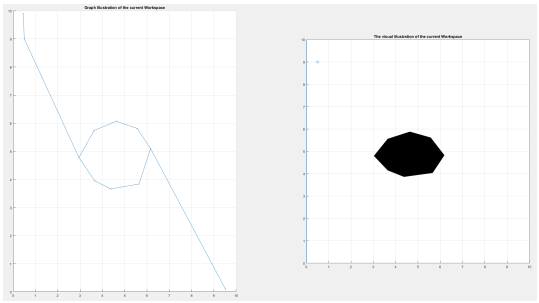

Figure 11: There exist a single octagonal obstacle in the environment. The robot is finding the graph victoriously.The robot started from [x,y] = [0.5,9]

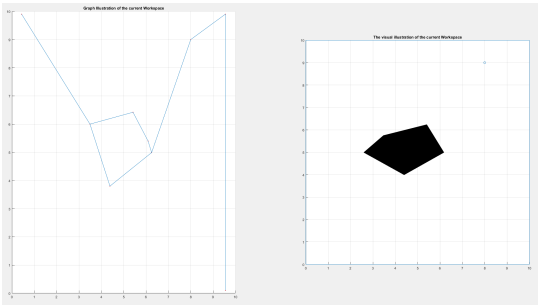

Figure 12: There exist same obstacle which is showed in Figure 5 but the robot is starting from different point. Even the starting position changes the robot is finding the graph accurately. The robot started from [x,y] = [8,9]

4 Conclusion

In this work, I gave a brief information about Sensor-based coverage and its implementation area. Later on, I explained my implementation of sensor-based coverage with FSM(Finite State Machine) like approach. One of the most challenging part of the implementation was finding critical points, so I explained this part in a separate Section 2.1. Thereafter, I tested the implementation from different aspects by changing the robot starting point and the environment. After doing some tests, I recognized that there is some deficiencies where the created graph is not complete for specific conditions, so I explained them also in Section 3.1. To sum up, sensor-based coverage is one of the important method for cases where the information about the environment is too little and it can be used for very different robots such as mine sweeping, house cleaning robot. My main achievement was that before implementing graph based problem, I have to work more and construct the code more graph/nodes creation oriented.

References

[1] Z. Butler, A. Rizzi, and R. Hollis, "Contact sensor-based coverage of rectilinear environments," *Proceedings of the 1999 IEEE International Symposium on Intelligent Control Intelligent Systems and Semiotics*, pp. 266–271, 1999.

[2] H. Choset, *Principles of Robot Motion*. MIT Press, 2016.

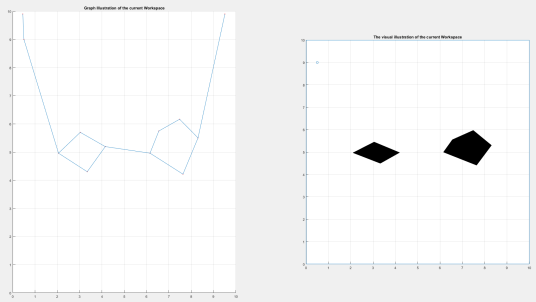

Figure 13: There exist two different obstacles in the environment and they are positioned side-by-side. Even there is multiple obstacles, if they positioned side-by-side, the robot will find the graph victoriously.The robot started from [x,y] = [0.5,9]

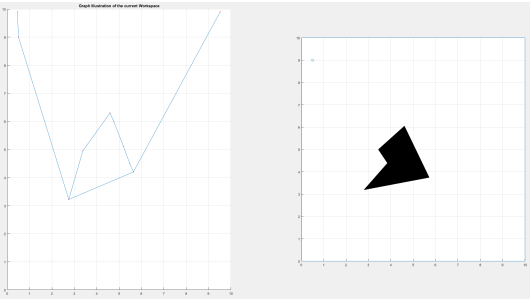

Figure 14: The environment is consisted of concave obstacle. The implementation can not cover the concave critical point due to lack of finding cell is covered or not. If the goal point is close to concave critical point, the robot can not find a path. On the contrary, the algorithm is complete. The robot started from [x,y] = [0.5,9]

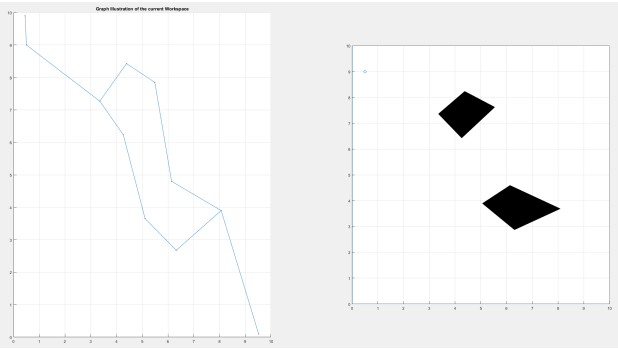

Figure 15: The environment is consisted of two obstacles. One of the obstacle is almost on top of the other obstacle. Due to implementation is lack of finding cell is covered or not, it can not cover the environment precisely. If the goal is not between the passages, the robot can find a path according to its starting position, else the algorithm is becoming incomplete because it can not return path even there exist a path.The robot started from [x,y] = [0.5,9]

[3] E. U. Acar, H. Choset, and J. Y. Lee, "Sensor-based coverage with extended range detectors," *IEEE Transactions on Robotics*, vol. 22, no. 1, pp. 189–198, 2006.

[4] E. U. Acar and H. Choset, "Sensor-based coverage of unknown environments: Incremental construction of Morse decompositions," *International Journal of Robotics Research*, vol. 21, no. 4, pp. 345–366, 2002.

[5] B. M. A. K. Burgard W., Stachniss C., "Introduction to mobile robotics - robot motion planning," July 2011.

[6] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006. Available at http://planning.cs.uiuc.edu/.