A Study of Motion Planning Algorithms for Mobile Robots

Savita Kshirsagar, Kamini Shukla

Department of CSE , Priyadarshini Institute of Engg. & Tech., Affiliated to Nagpur University Nagpur , India
Department of MCA , Priyadarshini Institute of Engg. & Tech., Affiliated to Nagpur University Nagpur , India

savitap_kshirsagar@rediffmail.com kamini_shukla@yahoo.co.in

Abstract---**This paper introduces a study of motion planning algorithms for mobile robots in which we addresses the problem of designing provable path planning algorithm in the frame work of the model with incomplete information. We introduces two such algorithms are called Bug algorithms that uses less sensing information than any other within the family of bug algorithm. This paper present the first approach to visibility-based potential application areas include surveillance, high risk military operations, video game design search-and-rescue efforts, firefighting, and law enforcement. Research findings can be applied not only to robotics but to planning routes on circuit boards, directing digital actors in computer graphics, robot-assisted surgery and medicine, and in novel areas such as drug design and protein folding.**

Keywords--- **Bug Algorithm, Path Planning, Bug 2**

I. INTRODUCTION

The current research on robot path planning can be classified into two large categories depending on which of the two following basic models is being used. In the first model, called path planning with complete information, perfect information about the obstacles is assumed. In the second model, called path planning with incomplete information, an element of uncertainty about the environment is present. Another important distinction can be made between the provable and heuristic approaches. In these terms, this paper addresses the problem of designing provable path-planning algorithms in the framework of the model with incomplete information.

The robot is unable to access precise information regarding position coordinates, angular coordinates, time, but is nevertheless able to navigate itself to a goal among unknown piecewise-analytic obstacles in the plane. The only sensor providing real values is an intensity sensor, which measures the signal strength emanating from the goal. The signal intensity function may or may not be symmetric; the main requirement is that the level sets are concentric images of simple closed curves, i.e. topological circles. Convergence analysis and distance bounds are established for the presented approach.

Navigation in an unknown environment is a classical robotics problem. Typically, the robot must gather information about the obstacles in the environment, its position coordinates, orientation, and much more. If limited sensors deny the robot access to this information, one may wonder if it can complete any task of significance. Various portions of the radio wave spectrum are sensed by numerous devices, including submarines, wireless heart monitors, radios, televisions, mobile phones, and anything with Bluetooth. The main question in this project is: Can we get a robot navigate to the source of a transmitter among unknown obstacles while only being able to sense the signal intensity and estimate its local gradient? Yes we can.

Bug algorithms are special type of path finding algorithms. These algorithms are classical and widely used for sensor-based path finder. There are different types of algorithms: Bug 1[1], Bug 2[1], Tangent Bug [2], Dist Bug [5], Wedge Bug [8], and Rover Bug [6], all of them usually called "Bug algorithms".

The purpose of this paper is to generate a collision-free path by using the boundary-following and the motion-to-goal behaviors. The Bug's family has three assumptions about

the mobile robot: i) the robot is a point, ii) it has a perfect localization, and iii) its sensors are precise.

A Bug 1 Algorithm

Perhaps the most straight forward path planning approach is to move toward the goal, unless an obstacle is encountered, in which case, circumnavigate the obstacle until motion toward the goal is once again allowable. Essentially, the Bug l algorithm formalizes the "common sense" idea of moving toward the goal and going around obstacles. The robot is assumed to be a point with perfect positioning with a contact sensor that can detect an obstacle boundary if the point robot "touches" it. The robot can also measure the distance $d(x, y)$ between any two points x and y. Finally, assume that the workspace is bounded. Let $B_r(x)$ denote a ball of radius r centered on x, i.e., $B_r(x) = \{y \in R^2 \mid d(x,y) < r\}$. The fact that the workspace is bounded implies that for all $x \in W$, there exists an r such that $W \subset B_r(x)$.

The start and goal are labeled Start and Goal, respectively. Let Start= Start and the m-line be the line segment that connects Start to Goal. Initially, $i = 0$. The Bug l algorithm exhibits two behaviors: motion-to-goal and boundary-following. During motion-to-goal, the robot moves along the m-line toward Goal until it either encounters the goal or an obstacle. If the robot encounters an obstacle, let H1 be the point where the robot first encounters an obstacle and call this point a hit point. The robot then circumnavigates the obstacle until it returns to H1. Then, the robot determines the closest point to the goal on the perimeter of the obstacle and traverses to this point. This point is called a leave point and is labeled L1. From L1, the robot heads straight toward the goal again, i.e., it reinvokes the motion-to-goal behavior. If the line that connects L1 and the goal intersects the current obstacle, then there is no path to the goal; note that this intersection would occur immediately "after" leaving L1. Otherwise, the index i is incremented and this procedure is then repeated for Li and Hi until the goal is reached or the planner determines that the robot cannot reach the goal (figures 2.1, 2.2). Finally, if the line to the goal "grazes" an obstacle, the robot need not invoke a boundary following behavior, but rather continues onward toward the goal.

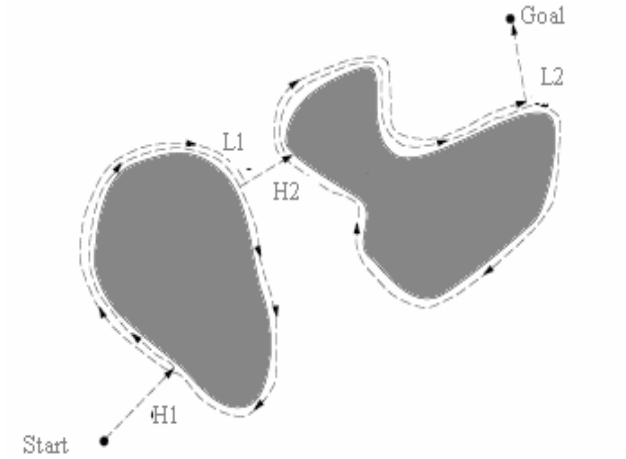

Figure 2.1- The Bug 1 Algorithm finds the goal successfully.

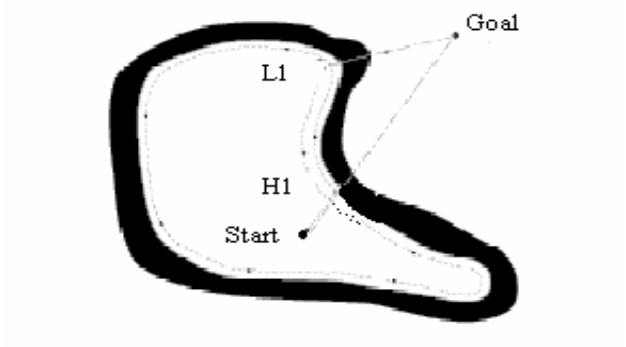

Figure 2.2-The Bug 1 Algorithm cannot find the goal.

Bug 1 Algorithm is the simplest algorithm among Bug Algorithms. It goes toward goal and if it faces obstacles then it follows boundary of the obstacle as measuring distance between current position and goal. After getting back to initial position, it finds the shortest points and goes that point. It does this process until getting to goal.

This algorithm is quite exhausted because robot should all perimeters of obstacles and in worst case 1.5 times of all perimeters of the obstacles robot faces. This makes robot exhausted and

sometimes it can fall in infinite loop. Also it has more memory to store all the value of perimeters of obstacles. Each value of perimeter compare with the value of Goal point.

The Bug1 algorithm is the most naive of the three algorithms in a sense that it only uses the range data to cut corners and move around the obstacle until if finds a point closest to the target. The robot then moves ahead from this closest point towards the goal point. The procedure Bug l is to be executed at any point of a continuous path. The goal is to generate a path from the Start to the Goal. When meeting an ith obstacle, a hit point Hi i = 1, 2 , ... When leaving the ith obstacle, to continue its travel toward the Goal, It defines a leave point Li initially, i= 1; L_0 = Start.

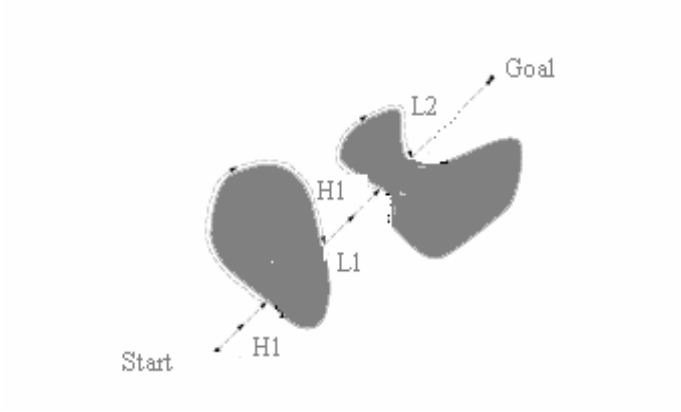

Figure 2.3- The Bug 2 Algorithm finds the goal successfully.

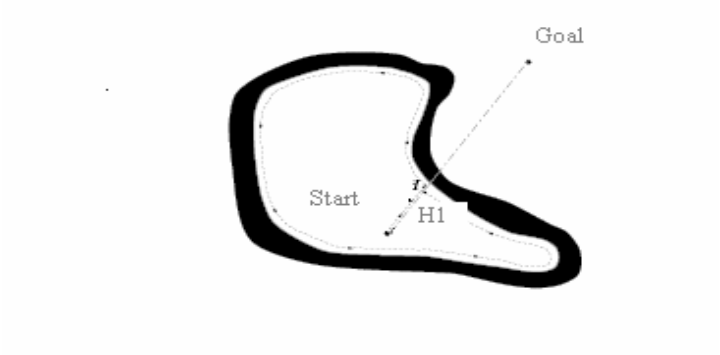

Figure 2.4- The Bug 2 Algorithm can not find goal.

II Bug 2 Algorithms

Like its Bug 1 sibling, the Bug 2 algorithm exhibits two behaviors: motion-to-goal and boundary-following. During motion-to-goal, the robot moves toward the goal on the m-line; however, in Bug 2 the m-line connects Start and Goal, and thus remains fixed. The boundary-following behavior is invoked if the robot encounters an obstacle, but this behavior is different from that of Bug l. For Bug 2, the robot circumnavigates the obstacle until it reaches a new point on the m-line closer to the goal than the initial point of contact with the obstacle. At this time, the robot proceeds toward the goal, repeating this process if it encounters an object. If the robot re-encounters the original departure point from the m-line, then the robot concludes there is no path to the goal (figures 2.3, 2.4).

Let $x \in Wi \subset R^2$ be the current position of the robot, $i = 1$, and Start be the start location.

At first glance, it seems that Bug 2 is a more effective algorithm than Bug l because the robot does not have to entirely circumnavigate the obstacles; however, this is not always the case. This can be seen by comparing the lengths of the paths found by the two algorithms. For Bug l, when the ith obstacle is encountered, the robot completely circumnavigates the boundary, and then returns to the leave point. In the worst case, the robot must traverse half the perimeter, pi, of the obstacle to reach this leave point. Moreover, in the worst case, the robot encounters all n obstacles. If there are no obstacles, the robot must traverse a distance of length d (Start, Goal).

A casual examination of (2.1) and (2.2) shows that Bug 2 can be arbitrarily longer than Bug

1. This can be achieved by constructing an obstacle whose boundary has many intersections with the m-line. Thus, as the "complexity" of the obstacle increases, it becomes increasingly likely that Bug l could outperform Bug 2 (figure 2.4).

In fact, Bug l and Bug 2 illustrate two basic approaches to search problems. For each obstacle that it encounters, Bug l performs an exhaustive search to find the optimal leave point. This requires that Bug l traverse the entire perimeter of the obstacle, but having done so, it is certain to have found the optimal leave point. In contrast, Bug 2 uses and opportunistic approach. When Bug 2 finds a leave point that is better than any it has seen before, it commits to that leave point. Such an algorithm is also called greedy, since it opts for the first promising option that is found. When the obstacles are simple, the greedy approach of Bug 2 gives a quick payoff, but the obstacles are complex, the more conservative approach of Bug 1 often yields better performance.

The Bug 2 Algorithm is improved algorithm in some sense because it does not have to follow all perimeters of the obstacles it faces.

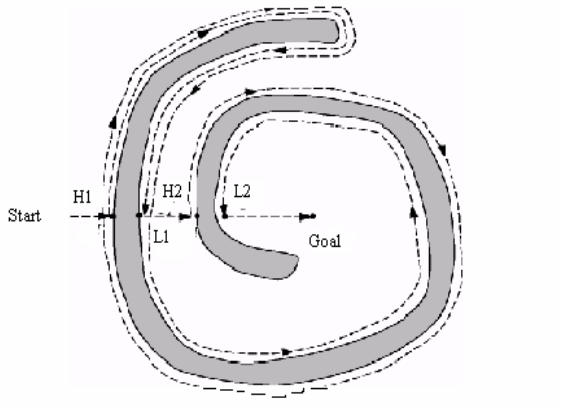

Figure 2.5- Bug 2 algorithm with complex obstacle .

It leaves the obstacles when it meets intersection between line from start point to goal point and obstacles if possible. In that case we do not have that much of memory than Bug 1, but all these depend on the obstacles present in that environment. If obstacle is more complex or bug select wrong direction then this technique cause serious problem. As I mentioned, if bug selects wrong direction in escaping spiral obstacle, it has to take a walk even much more than Bug 1 algorithm. Example shows how it works and situation when exhausted and when failed. In Bug 2 algorithms.

A desirable path to the goal point Goal, called the M-line is introduced as a straight-line segment that connects the start point and the goal point. An elementary operation of defining the next intermediate target point is executed by the robot at every moment given its current position and the range data within the current field of vision. Then the robot makes a little step in the direction of goal point and the process repeats. A local direction is a once and for all determined direction facing the obstacle; it can either be clockwise or anti-clockwise. For the sake of clarity I have always assumed it to be anti-clockwise. Instead of like the Bug 1 the robot takes advantage of opportunities that look more promising

II. CONCLUSIONS

The Bug 1 Algorithm is very much simple to implement .But it take more time and distance than other algorithms. Also it has more memory than other algorithms to store each point or pixel on the screen ,which is used to calculate distance from Start to Goal point, because in this algorithm robot traveled all the points covered by perimeter of the obstacle , also find out next nearest point towards Goal.

The Bug 2 Algorithm is improved algorithm in some sense because it does not have to follow all perimeters of the obstacles it faces. It leaves the obstacles when it meets intersection between line from start point to goal point and obstacles if possible. So it leaves the points as close as goal but sometimes this technique cause serious problem. As I mentioned, if bug

selects wrong direction in escaping spiral obstacle, it has to take a walk even much more than Bug 1 algorithm. Example shows how it works and situation when exhausted and when failed. Bug 2 and Bug l are the least complicated. To implement the Bug Algorithm, 2 algorithms such as Bug 1, Bug 2 algorithms are implemented in java.

In order to implement these Bug Algorithms, a main GUI is developed in java, where user can select individual algorithms. There are only two algorithms to implement but in these two we have also included left and right side of the robot, i.e. Bug 1R means robot travel form right side. Same we implement Bug 1L, Bug 2R, and Bug 2L. User can select any one of the algorithm with any one of the obstacle. There are four different shapes of obstacles used in this implementation. Also Special Case obstacle especially for Bug 1 and Bug 2 Algorithms to show the limitation of Bug 2. Special Case obstacle is a complex obstacle.

Thus we will also develop the Tangent Bug, Dist Bug, Wedge Bug, and Rover Bug. The Tangent Bug deals with finite distance sensing. Tangent Bug is useful in unbounded environments. In addition, it produces "locally optimal "solutions, that is, the resultant paths are the shortest length possible given the use of solely local information. "Wedge Bug" algorithm to address the shortcoming of Tangent Bug, as a step towards a more practical path planner for flight micro rovers. Wedge Bug is complete, correct, and relies solely upon the robot's sensors.

References:

[1] Howie Choset, Kevin Lynch, Seth Hutchinson, Geore Kantor, Wolfram Burgard, Lydia Kavraki and Sebastain Thrun. Principal of Robot Motion: Theory, Algorithms, and Implementation. September 14, 2007.

[2] I. Kamon, E. Rimon, and E. Rivlin. Tangent bug: A range-sensor based navigation algorithm. Int. Journal of Robotics Research, 17(9):934-953, 1998.

[3] .V. Lumelsky and T. Skewis, Incorporating range sensing in the robot navigation function, IEEE Transactions on Systems Man and Cybernetics, vol. 20, pp. 1058-1068, 1990.

[4] .V. Lumelsky and Stepanov, Path-planning strategies for a point mobile automaton amidst unknown obstacles of arbitrary shape, in Autonomous Robots Vehicles, I.J. Cox, G.T. Wilfong (Eds), New York, Springer, pp. 1058-1068, 1990.

[5]. I. Kamon, E. Rimon, and E. Rivlin, "A New Range-Sensor Based Globally Convergent Navigation Algorithm for Mobile Robots," CIS-Center of Intelligent Systems 9517, Computer Science Dept., Technion, Israel, 1995.

[6]. S. L. Laubach J. W. Burdick , "An Autonomous Sensor-Based Path-Planner for Planetary Micro rovers" Jet Propulsion Laboratory Department of Mechanical Engineering, California Institute of Technology California Institute of Technology Pasadena, CA 91109 Pasadena, CA 91125 [209] K. Kanazawa, D. Koller,

[7].V. Lumelsky and T. Skewis, "Incorporating Range Sensing in the Robot Navigation Function", IEEE transactions on systems, man and cybernetics.

[8]. I. Kamon and E. Rivlin, "Sensor based motion planning with global proofs", IEEE transactions on robotics and automation.

[9] M.H. Lee and H.R. Nicholls, "Tactile sensing for mechatronics- a state of-the-art surgery," Mechatronics, vol. 9, no. 1, pp. 1-31, Feb. 1999.

[10] Mark H. Lee, "Tactile sensing: new directions, new challenges," International Journal of Robotics Research, vol. 19, no. 7, pp. 636-643,July 2000.