CENG786 - Spring 2018
Homework I - Bug Algorithms

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

Abstract

One of the substantial problem of the autonomous mobile robots is **Path-planning** which is trying to find the optimal way between two points. Optimal paths for the autonomous mobile robots can be defined according to different aspects such as energy, shortest path, shortest time, minimize the amount of turning or whatever a specific application requires. In this work, I tried to implement and compare two popular path planning algorithms which are Bug1 and Bug2.

Keywords path planning, mobile robots, bug algorithms

1 Introduction

In many cases of path planning, there exist a global map of the environment when the robot starts to move to the goal point. Potential field based-planning can be used to solve that kind of cases [1]. However, local potential field-based path planners are not complete. It is possible to satisfy certain performance criteria in the presence of uncertainty by using some different algorithms such as Bug algorithms. Bug algorithms handle these uncertainties and limited information by using some primitive sensors to detect the nearest obstacle [2]. Although, Bug1 and Bug2 algorithms use same sensor information to find the appropriate path, they differ from each other. In this work, I gave a brief introduction about the Bug algorithms in Section 1.1-1.2, than I explained how I implemented the algorithms in MatLab environment and my overall structure especially about the FSM(Finite State Machine) which I used and BoundaryFollowing in Section 2. In Section 4, I compared Bug1 and Bug2 for the different obstacles with regards to elapsed time, completeness and soundness. Finally, I summed up everything in Section 5.

1.1 Bug 1 Algorithm

Bug 1 algorithm generally execute two behaviors which are motion-to-goal and boundary following. In motion-to-goal state, the robot moves straight to the goal along the m-line until it either encounters the obstacles or goal. If robot encounters the object, it is circumnavigating it and finds the closest point to the goal from the obstacle surface. Than, it is moving back to this point and it is invoking motion-to-goal behaviour. It is also known as **exhaustive search algorithm** because it looks at all choices before committing any action. You can see Figure -1, for clearer understanding.

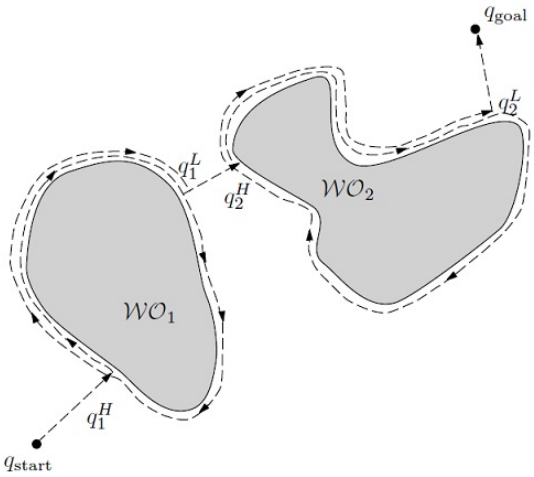

Figure 1: Bug1 Algorithm visualization, the algorithm is explained with detail in section 1.1 [3]

1.2 Bug 2 Algorithm

Bug 2 algorithm appears similar to Bug 1 except it is not circumnavigating the object. Whenever the robot cross the m-line, it towards directly to the goal. most It is also known as **greedy algorithm** because it takes the first thing that looks better. In many cases, Bug 2 outperforms Bug 1 because of its greedy nature. However, Bug 1 generally has a more unsurprising execution. You can see Figure -2, for clearer understanding.

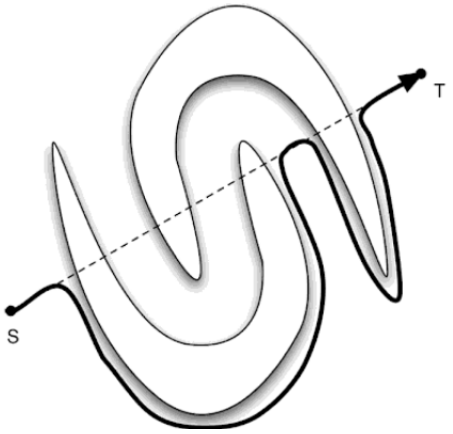

Figure 2: Bug2 Algorithm visualization, the algorithm is explained with detail in section 1.2 [4]

2 Implementation

There is several different methods to implement Bug algorithms. In this work, I created two slightly different FSMs(Finite State Machine) to implement Bug1 and Bug2. These two FSMs are different because their state numbers and transition conditions are not same. Figure 3 and Figure 4 are explaining Bug1's FSM and Bug2's respectively.

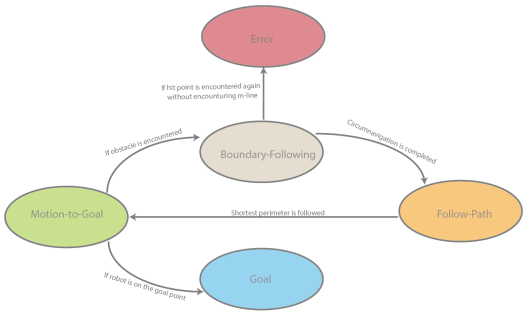

Figure 3: Bug1's FSM is consisted of five states which are Motion-to-Goal, Boundary Following, Follow Path, Goal, Error. In the Motion-to-Goal state, the robot moves straight to the goal along the m-line. Boundary Following and Follow Path states are explained deeply in Part-2.1. Goal and Error states are absorber state of the SM which means ones the robot enters these states can not go back to the other states

As you can see from Figure [3-4], there is some common and different states in Bug1 and Bug2 algorithms. Motion-to-Goal is one of the common state and the robot is basically moving toward the goal as explained in Section 1.1-1.2. However, Bug2 Algorithm does not have FollowPath state and its BoundaryFollowing state is a bit different because the robot which uses Bug2 algorithm does not have to circumnavigate whole obstacle and go back to the closest point to the goal. As we said before, Bug2 algorithm has a greedy nature so the robot will start to toward the goal as soon as it encounters the m-line.

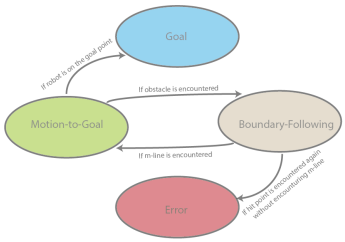

Figure 4: Bug2's FSM is consisted of four states which are Motion-to-Goal, Boundary Following, Goal, Error. In the Motion-to-Goal state, the robot moves straight to the goal along the m-line. Boundary Following state is explained deeply in Part-2.1 and it is almost the same to Bug1's FSM Boundary Following state except its transition condition to the other states. Goal and Error states are absorber state of the SM which means ones the robot enters these states can not go back to the other states

2.1 Boundary Following

Boundary following is one of the most crucial and challenging part of the Bug algorithms because the robot has to circumnavigate the object properly and it has to stay inside a safe distance interval. Some primitive sensors such as range scanner, touch sensor can be used to detect the obstacle but still the appropriate traverse vector has to be defined to have robust boundary following. In this work, the overall boundary following diagram showed in Figure 5. One can see that the robot is using two different sensor rays which are colored as light yellow. These sensor rays have α degree between them and I named the first sensor ray which has l_1 distance to the obstacle as a **lagging ray** and the other one as **leading ray**. I calculated specific vectors according to these rays and later on I derived the traverse vector which can be showed as $\vec{L}_{pos_n pos_{n+1}}$. In simple terms, the traverse vector is derived as sum of three vectors which are $\vec{L}_{pos_n hp_1}$, $\vec{L}_{hp_1 hp_2}$, $\vec{L}_{hp_2 pos_{n+1}}$ (Eq. 3). Basically, the robot moves according to calculated traverse vector and then it is updating its lagging ray angle according to traverse vector. The calculations of the each vectors are explained in Section 2.1.1.

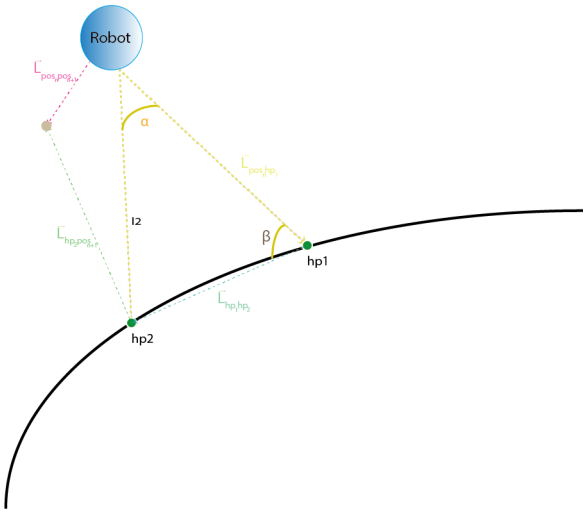

Figure 5: boundfollow bla bla

2.1.1 Vector Calculations

In this section, I tried to explain vector calculations which are done to find appropriate traverse vector. In this part, λ means the robot's lagging ray angle w.r.t. map coordinate system. The distance between two hit points calculated by cosinus theorem (Eq. 1),which is shown as L_2 norm of two point,

$$\Delta l = \|hp_1, hp_2\|_2 = \sqrt{l_1^2 + l_2^2 - 2 * l_1 * l_2 * cos(\alpha)} \quad (1)$$

used for calculating β which is also referred as hit angle (Eq. 2).

$$\arccos(\beta) = \frac{l_1^2 + \Delta l^2 - l_2}{2 * l_1 * \Delta l} \quad (2)$$

$$\vec{L}_{pos_n pos_{n+1}} = \vec{L}_{pos_n hp_1} + \vec{L}_{hp_1 hp_2} + \vec{L}_{hp_2 pos_{n+1}} \quad (3)$$

The descriptor of vectors are,

- $\vec{L}_{pos_n hp_1} = l_1 * [cos(\lambda) sin(\lambda)]$
- $$\begin{cases} angle = \lambda + \beta & \text{CCW} \\ angle = \lambda + \pi - \beta & \text{CW} \end{cases}$$

  $$\vec{L}_{hp_1 hp_2} = \Delta l * [cos(angle) sin(angle)]$$
- $$\begin{cases} angle = \lambda + \beta - 3 * pi/2 & \text{CCW} \\ angle = \lambda + 3 * \pi/2 - \beta & \text{CW} \end{cases}$$

$$\vec{L}_{hp_2 pos_{n+1}} = l * [cos(angle) sin(angle)]$$
$$l = (Sensordistance) - 2 * (Safedistance)$$

2.1.2 Exceptional Cases

I tried to explain BoundaryFollowing state with its equations and explanations. However, there is one exceptional case where the leading ray measures infinity when it encounters obstacle sharp curvatures (Fig. 6). In this situation, the robot starts decrease angle between two rays until it hits the obstacle again and later on it mirrors its position according to new leading ray.

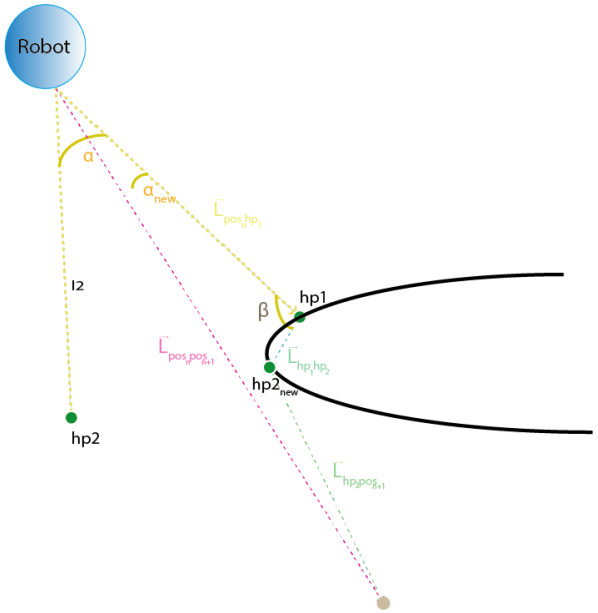

Figure 6: boundfollow bla bla

The descriptor of two vectors ,which are $\vec{L}_{pos_n hp_1}$,$\vec{L}_{hp_1 hp_2}$, are same. The last vector's descriptor is defined as,

$$\begin{cases} angle = \lambda - \alpha_{new} & \text{CCW} \\ angle = \lambda + \alpha_{new} & \text{CW} \end{cases}$$

$$\vec{L}_{hp_2 pos_{n+1}} = l_{new} * [cos(angle) sin(angle)]$$
$$l_{new} = ((Sensordistance) - 2 * (Safedistance))/cos(\alpha_{new})$$

3 Important Properties

Path planning algorithms can be used and implemented in the real world application according to some important parameters such as completeness, soundness and correctness. These properties analyze the solution space of the algorithm and give some information about the algorithm is robust or not, optimal or not. Before analyzing my own Bug algorithms, I will explain some mathematical notation and give some basic information about these properties.

- **Completeness** says that if there exists a path the algorithm will find the path.
- **Soundness** says that if the algorithm will find the path, the whole path has to lay on free workspace.

In this work, Bug1 and Bug2 is sound because it is returning error if there is not any solution. As you can see from Figure **??**, the boundaryFollowing state is returning error if there is not any path. Moreover, in these work, I did not see any contrary situation where the algorithm is incomplete or unsound. The proof of the algorithms are completenes or soundness in all situation is out of scope of this homework.

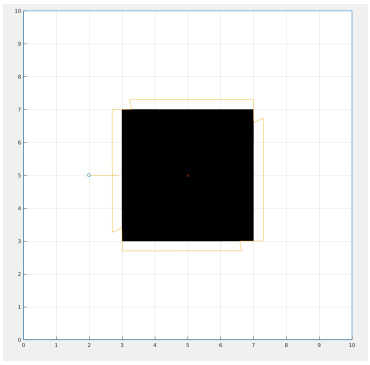

Figure 7: Bug algorithms are returning error if there is not any path

4 Results & Discussions

In this part, I showed the results of the Bug algorithms according to different conditions. Firstly, I compared the bug algorithms' total elapsed time according to different ray angles. Later on, I created two map to compare Bug1 and Bug2. In Figure [11-11], Bug2 is outperforming Bug1 and Bug1 is outperforming Bug2 respectively.

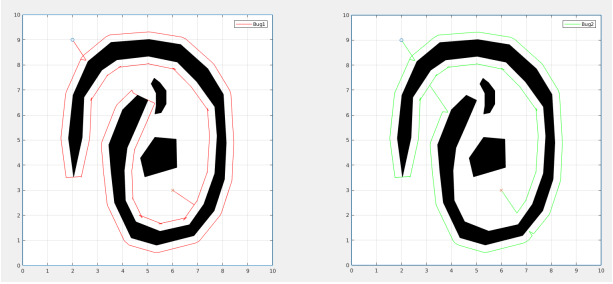

Figure 8: Bug1 and Bug2 comparison in the first map which is created by the instructor, (angle between rays) $\alpha = \pi/45, qstart = \begin{bmatrix} 2 & 9 \end{bmatrix}, qgoal = \begin{bmatrix} 6 & 3 \end{bmatrix}, t_{bug1} = 9.99, t_{bug2} = 10.5$

In Figure [8-9], one can see that when the ray angle is decreased, the robot is trying to recognize the boundary of the obstacles in detail. Hence, the accuracy will increase but the total elapsed time will decrease. Moreover, Bug2 algorithm is outperforming Bug1 due to structure of the obstacles (Greedy algorithms: 1 - Exhaustive Algortihms:0). On the other hand, sometimes the greedy algorithms have some disadvantages when the obstacles is changed.

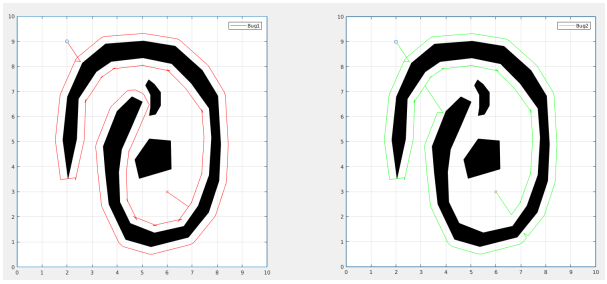

Figure 9: Bug1 and Bug2 comparison in the first map which is created by the instructor, (angle between rays) $\alpha = \pi/180, qstart = \begin{bmatrix} 2 & 9 \end{bmatrix}, qgoal = \begin{bmatrix} 6 & 3 \end{bmatrix}, t_{bug1} = 35.8, t_{bug2} = 34$

As you can see from the Figure 10, Bug2 algorithm is mapping the obstacle again and again due to its greedy nature, but bug1 is mapping only once and going to the goal at one mapping. (Greedy algorithms: 1 - Exhaustive Algortihms:1). If we simplify the obstacle as in Figure 11, the greedy nature will gain the control back and it will finish the path immediately. (Greedy algorithms: 2 - Exhaustive Algortihms:1)

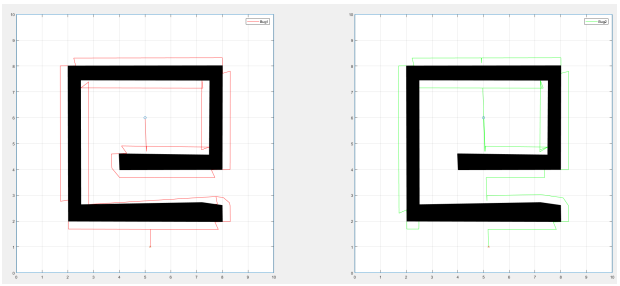

Figure 10: In this map, Bug1 is outperforming Bug2, (angle between rays) $\alpha = \pi/180, qstart = \begin{bmatrix} 5 & 6 \end{bmatrix}, qgoal = \begin{bmatrix} 5.2 & 1 \end{bmatrix}, t_{bug1} = 28.1, t_{bug2} = 28.9$

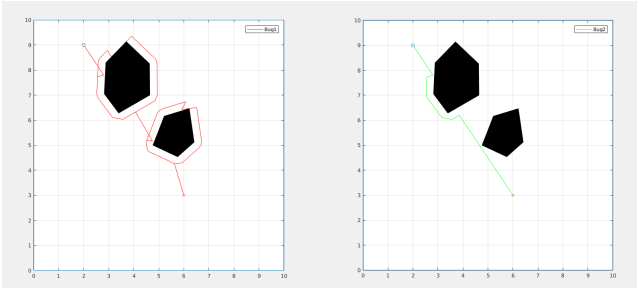

Figure 11: In this map, Bug2 is outperforming Bug1, (angle between rays) $\alpha = \pi/180, qstart = \begin{bmatrix} 1 & 4 \end{bmatrix}, qgoal = \begin{bmatrix} 9 & 9 \end{bmatrix}, t_{bug1} = 6.92, t_{bug2} = 1.88$

5 Conclusion

In this work, I tried to explain Bug1 and Bug2 algorithms and my own implementation of these algorithms with FSM(Finitie State Machine) approach. After the implementation, I compared both of the implementations according to different shaped obstacles, angle between sensor rays and properties which are explained in Section 3. To sum up the whole work, Bug1 and Bug2 represent two essential ways to deal with search problems. Bug1 is performing an exhaustive search which means that is a conservative approach, so it is finding the optimal leaving point from the obstacles after circumnavigating is finished. These is applicable even for complex obstacles but it is too costly in time manner. On the other hand, Bug2 utilizes a crafty approach and it is leaving the obstacle when it encounters the m-line which is the best leaving point for the time being. These kind of algorithms called greedy as we said in Section 1.2. As Choset (2005) [5] said that, when the obstacles are simple, the greedy approach of Bug2 gives a quick payoff, but when the obstacles are complex, the more conservative approach of Bug1 often yields better performance.

References

[1] J.-C. L. J. Barraquand, B. Langlois, "Numerical potential field techniques for robot path planning," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 22, no. 2, pp. 224 – 241.

[2] e. a. Buniyamin, N Ngah, "A simple local path planning algorithm for autonomous mobile robots," *International journal of systems applications, Engineering development*, vol. 5, no. 2.

[3] U. Sinha, "Obstacle avoidance with the bug-1 algorithm."

[4] J. M. Dudek, G., *Computational principles of mobile robotics*. Cambridge University Press, 2011.

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