Proceedings of the 2023 International Conference on Machine Learning and Automation
DOI: 10.54254/2755-2721/33/20230278

A comprehensive review of bug algorithms in path planning

Keming Liu

Department of Mechanical Engineering, The Hong Kong Polytechnic University, Hong Kong, China

22107218d@connect.polyu.hk

Abstract. The bug algorithm family is the well-known navigation algorithm for robots in unknown environments. However, different types of bug algorithms have their own priorities and weaknesses based on different environmental conditions. According to the characteristics of the bug navigation algorithms, the bug algorithm family is mainly divided into 4 parts. In this paper, four types of bug algorithms (original bug algorithm, M-line Bug, Angel Bug and Range Bug) are presented along with typical samples of each type. Each type of algorithm will be thoroughly explained and exemplified, highlighting its unique characteristics and the suitable environmental conditions for each algorithm. By presenting the basic logic behind each algorithm and examining the environmental conditions they are best suited for, this paper aims to provide researchers with a comprehensive understanding of the bug algorithm family. Ultimately, this knowledge will contribute to the advancement of navigation capabilities in robots operating in unknown and challenging environments.

Keywords: bug algorithm, mobile robot, navigation algorithm, path planning.

1. Introduction

The revolution in the field of artificial intelligence and robotics has made it possible to do high-risk work in some unreachable regions using intelligent mobile robots. Different kinds of robots permit human beings access to different hazardous regions, including accidents like fire, earthquake, nuclear radiation and deep-sea [1]. Rescue missions are hard to launch in these regions as the high-level unpredictable hazard. However, these regions also presented a number of difficulties, such as planning routes, avoiding obstacles and localizing the position.

It is challenging in the field of robot navigation, as it is a vital function in building a robot with the ability to auto-move. Robot navigation environments can be classified into three types: completely unknown, partially unknown and completely known environment. In a completely known environment, it is easy for robots to create the binary map and use A* algorithms to solve it [2]. Moreover, in a completely or partially known environment, the functions of navigation and obstacle avoidance become necessary to build a safe path. The bug algorithm (BA) can form the safety path to the target and take a quick decision to avoid the obstacles by using sensorial information.

The name of the bug algorithm is suggested a biological origin. However, BA is a path-planning technique evolved from the existing maze solving algorithms like Dijkstra and A*[3,4]. The main principle behind the BA is that they are not aware of information of the environment, only the relative

DOI: 10.54254/2755-2721/33/20230278

position of the target. Only when the algorithm come into contact with obstacles and walls, it reacts locally, allowing the robot to track the edge and move towards the target.

This paper provides a summary review of mainstream algorithms in the bug family. Based on the characteristics of path planning algorithms, they are classified into origin Bug algorithms, M-line-bugs, angel-bugs and range-bugs. Then, the algorithms are further classified, the principles of each algorithm and their own limitations are introduced, and the advantages and disadvantages of each algorithm are summarized. Finally, according to the current research status of mobile robot path planning algorithms, this paper makes a further outlook and analyses the development trend of these algorithms.

2. Classification of bug algorithms

The term bug algorithm was first coming into exist at the last 80s, evolving from the path planning algorithms. However, the latter algorithms have to apply in a completely known environment, including the local coordinates of the start and end points as well as positions of all obstacles in the location. The bug algorithm explored the navigation problem in an environment without the information about the obstacles and their positions. While the bug algorithm will be better suited to perform the task in a typical no-crossing-obstacle environment [5].

Bug algorithms require very little memory to solve the navigation problem as only a minimal number of waypoints are stored, but not a full map of the environment. The bug algorithm makes three hypothetic to the mobile robot. First, the robot is assumed to be a point object. Second, the localization ability is perfect. Third, sensors are perfect [6]. These assumptions make Bug algorithms in an ideal environment. Therefore, Bug algorithms may not be able to apply to the real robot directly but can be considered as the supervisory methods to solve the 2D navigation problem.

Bug algorithms plan the path to avoid obstacles and navigate the robot as a bug. Bug algorithms let the robot reach the destination if it lies in the given region. This algorithm is not just a goal-orientated process. It includes two behaviours, moving to the goal and avoiding the obstacles. The robot moves to approach the goal and starts to follow the edge of the obstacles when it meets during the process. After avoiding the obstacle, the robot restarts to approach the goal [7]. Furthermore, there are several kinds of bug algorithms exist. Based on their development and added feature, those methods are classified into four types: origin Bug algorithm, M-line-bugs, angel-bugs and range-bugs, while each type of bug algorithm suits certain conditions with superiority [6]. However, it may be less efficient in other conditions, shown on figure 1 [8].

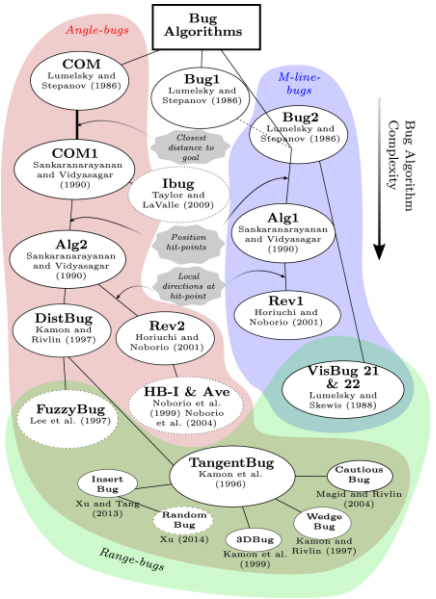

Figure 1. The classification of the bug family [8].

DOI: 10.54254/2755-2721/33/20230278

2.1. Bug 1 algorithm

The bug-1 algorithm is the earliest navigation algorithm in the field of obstacle avoidance. And it is one of the most efficient and easy methods in the family of techniques for obstacle avoidance [9]. This algorithm encompasses two distinct behaviours: navigation towards the destination and obstacle avoidance. The robotic being initiates moving on a linear trajectory towards the target point. Once the robot has detected an obstacle, it starts a process of tracking the edge of the obstacle. The initial location at which the robot commences tracking is defined as the tracking point. During the process of tracking the boundary, the robot computes the distance between its own position and endpoint. If hit to the tracking point again, the robot restarts its movement towards the leaving point, which is determined to have the shortest local distance between the endpoint and the surrounding path [10]. Once successfully navigating around the obstacle, the robot proceeds to calculate a straight path from the leaving point (x1, y1) to the desired destination (x2, y2) by employing the straight-line equation [2]. The values for the slope, denoted as m, and the y-intercept, denoted as c, are provided in equations (1) and (2) correspondingly.

$$m = tan^{-1}\left(\frac{y_2 - y_1}{x_2 - x_1}\right) \tag{1}$$

$$C = y_1 - m \cdot x_1 \tag{2}$$

However, this bug algorithm does not have capacity to find the shorter path. Also, it wastes lots of time on surrounding the obstacles. The researchers realized that the Bug1 algorithm produces longer trajectories by default [11]. The community seems to agree, as no one has made any extensions or variants of the Bug1 algorithm, and thus no similar Bug algorithm has appeared since then [12].

2.2. M-line bugs

The Bug 2 algorithm is considered to be one of the simplest algorithms to implement, as it does not require the use of range sensors [13]. The Bug 2 algorithm encompasses two distinct behaviours, including travelling towards a goal and following the edges. The algorithm begins by establishing the M-line using the local coordinates of both the initial point and the target. The M-line is a linear segment that connects those two specified points. Following the establishment of the M-line, the algorithm initiates the moving-to-goal mode. The mobile robot is capable of travelling along the M-line. When the robot encounters an obstruction, the algorithm transitions into edge-following mode. The term used to refer to that concept is commonly known as the hit-point. In the edge-following mode, the robot possesses the capability to determine the direction in which the edge-following operation is executed. In the event that the destination is situated within the first and second quadrants, the robot will execute a left turn. In contrast, the robot exhibits a rightward turn when the destination is situated inside the third and fourth quadrants. In this operational state, the autonomous robot follows the perimeter of the barrier until it reaches the point of leaving, at which point it intersects with the M-line again. Once the robot reaches the leaving point, the algorithm transitions back into Moving-to-Goal mode and terminates when it reaches the final destination. In the event that the robot encounters the hit spot once more, the programme ceases execution, as it determines that the destination is unreachable [14].

2.3. Angle bugs

Dist-Bug is a mobile robot navigation algorithm that utilizes range sensors. The Dist-Bug algorithm utilises distance-based bug strategies to navigate data, enabling the robot to freely move towards the objective while ensuring global convergence before abandoning obstacle bounds. The leaving conditions are assessed directly based on the sensor readings, hence facilitating the implementation of the algorithm. The Dist-bug method, like other algorithms in the Bug family, encompasses two distinct behaviours: approaching the destination and tracing the edge of obstructions. At the onset, the robotic entity proceeds on a linear trajectory towards the target. The algorithm will terminate if the target is achieved. Alternatively, the algorithm transitions into a mode where it tracks the boundary of the obstacle after collision with the robot. The location at which the robot initially makes contact with the obstacle is referred to as the hit-point. During the boundary following process, the robot identifies the

DOI: 10.54254/2755-2721/33/20230278

shortest distance to the target. Additionally, the robot is capable of measuring and storing the distance between its current position and the closest obstacle in the direction of its intended destination. The robot restarts "Towards Destination" mode when the leaving conditions occur. The mode is changed when one of the following conditions is fulfilled: (1) If there are no obstacles detected in the path towards the objective and the target can be reached by a direct route. (2) When the distance to the destination is less than that of the preceding step. Once any of the specified leaving conditions are met, the robot starts the process of navigating around obstacles and resumes its trajectory towards the intended destination. However, If the robot hits the hit-point again, the algorithm terminates, as it determines that the target is inaccessible [15].

2.4. Range-bugs

2.4.1. Fuzzy bugs. Compared with other navigation systems, Fuzzy navigation systems are easier to apply as it can handle an infinite number of navigational situations with limited fuzzy rules by applying a fuzzy logic controller. The Fuzzy bug algorithm is basically built on three fuzzy rules: towards the destination, obstacle avoidance and tracking. In toward destination mode, the robot directly moves to the target. The obstacle avoidance mode guides the robot to avoid obstacles. The tracking matrix guides the robot along a path parallel to the obstacle [16]. In this "Toward Destination" mode, the robot can be oriented towards its destination by changing its turning angle based on the local coordinate of the target point. If the local coordinate of target is in the first or second quadrants, the robot will turn left. If the local coordinate of target is in the third or fourth quadrants, the robot will turn right. If a long distance exists between the current position and target, the turning angle becomes smaller. If a short distance exists between the current position and target, the turning angle becomes larger. In the event that the goal is situated in a positive direction, the robot will proceed to move directly towards it. In the event that the objective is situated in the negative direction, the robot will execute a turning operation until it aligns itself with the destination.

In this “Obstacle Avoidance” mode, the robot uses two front sensors and side sensors to determine the position of the obstacle. If left sensor and front sensors detect an obstacle, the robot turns right. In the event that right sensor and front sensors detect an obstacle, the robot will proceed to turn left. In the event that only the forward sensors identify an obstacle, the robot will proceed to make a random decision to turn either left or right. The robot exhibits a right turning behaviour, when met an obstacle in the left front region. The robot exhibits a left turning behaviour, when met with an obstacle in the right front region. The robot exhibits random left or right turning behaviour, when met with an obstacle directly in its front.

In the “tracking” mode, the robot uses side sensors to measure the distance to the obstacle surface and track the tangent line to the surface of the obstacle. If the robot encounters an obstacle, it steers tangentially parallel to the surface of the obstacle. If the robot follows the tangent line, it maintains a constant distance from the obstacle.

2.4.2. Tangent bug algorithm. The tangent bug algorithm is a sophisticated navigation system that relies on range sensors to calculate local shortest pathways. similar to the other algorithms in the Range Bug family, the tangent bug algorithm is comprised of three distinct behaviours: approaching the target, avoiding obstacles, and tracing the perimeter of an obstacle. The algorithm initiates with the robot consistently commencing in the Towards Destination mode. The robotic system navigates in a straight path towards the target. In the event that the robot finds an obstacle, the algorithm transitions into Obstacle Avoiding mode. Conversely, if no obstacles are encountered, the algorithm will stop running when it reaches the specified position. In the Obstacle Avoiding mode, the robot employs a systematic approach to detect and navigate around obstacles with the aim of achieving a smooth traversal towards the target goal. In this mode, the algorithm determines whether the robot is directly oriented towards the destination at any given moment by utilising local orientation. When the robot moves directly towards its destination, the algorithm restarts the Towards Destination mode. Furthermore, the Tracking mode

DOI: 10.54254/2755-2721/33/20230278

is initiated. The algorithm designates the initial position at which the Tracking mode commences as the tracking point. In tracking mode, the robotic system will employ a tracking mechanism to track the perimeter of the obstacle until it reaches the leaving point, at which point the robot will go directly towards the destination location. Once the robot reaches the designated leaving point, the algorithm will transition into Towards Destination mode. In the event that the robot returns to the tracking point, the programme will terminate and determine that the goal is unreachable [17-19]. The flowchart can be seen in Figure 2.

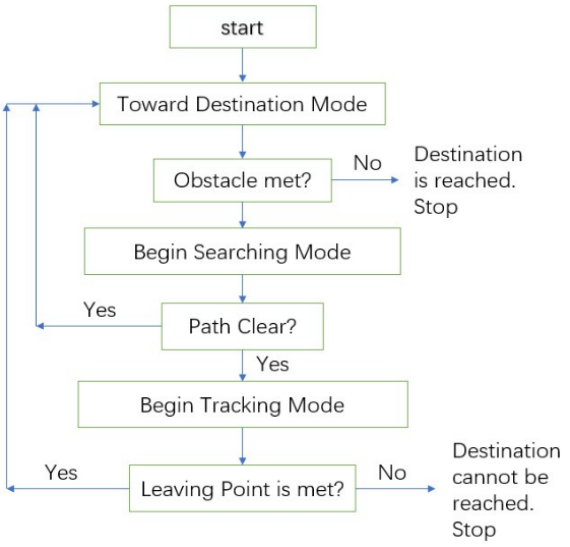

Figure 2. Flowchart of tangent bug.

3. Discussion

In Bug algorithms family, the precise location and shape of the obstacle are not information that bug algorithms require. However, it is important to obtain the data pertaining to the local coordinates of the target as well as their present location. In M-line bugs (Bug2), robots are able to memorize the M-line and the position of the target, as well as recognise whether or not they have arrived at the target. Angle bugs (Dist-Bug) need to analyse and memorize the distance and azimuth between target and their own location. BAs also need to memorize hit points to define the reachability of the target. Furthermore, the effectiveness of any algorithm is significantly influenced by the surrounding setting. The Bug 1 algorithm, while effective in preventing the robot from revisiting points on obstacle boundaries more than twice, has a drawback in environments consisting solely of convex obstacles. In such scenarios, the resulting path generated by the algorithm tends to be excessively long, as the robot is compelled to circumnavigate the complete circle of each visited obstacle. However, in environments with only convex obstacles, Bug2 generates much shorter paths. This is because the robot only bypasses a portion of each obstacle boundary, no local loops are generated. However, in environments where localized loops are generated, Bug2 generates very long paths, whereas Bug1 generates relatively short paths. Tangent bug algorithm generates the shortest path in spatially wide environments, relying on the range sensors. Also, in environments which contain lots of small obstacles, tangent bug algorithm can move directly towards the target while other algorithms are still on the edge following process.

4. Conclusion

This paper aims to provide a comprehensive analysis of four different types of bug algorithms: the original bug algorithm, M-line Bug, Angle Bug, and Range Bug. Each algorithm will be presented in detail along with typical samples to illustrate their implementation. While the concepts of these bug algorithms may seem well suited for lightweight robotic applications, it is worth noting that many of the existing variants rely heavily on global positioning systems and perfect on-board sensor systems. This

DOI: 10.54254/2755-2721/33/20230278

creates a limitation in practical implementations where such reliable and accurate positioning systems may not be available or feasible. However, the rapid development of artificial intelligence technology has opened up new possibilities for the improvement and advancement of bug algorithms. Through the integration of multiple technologies, there is an opportunity to improve the performance and applicability of path-planning algorithms, thereby expanding the range of applications for mobile robots. The integration of artificial intelligence technologies such as machine learning and computer vision has the potential to address the limitations of traditional bug algorithms. These advances can enable mobile robots to navigate efficiently and reliably in complex and dynamic environments. In addition, the combination of machine learning techniques could allow robots to adapt and learn from their surroundings, improving their decision-making capabilities and overall performance. By utilising recent advances in AI and multi-technology integration, the application of vulnerability algorithms can be extended to a variety of domains such as search and rescue operations, agriculture, exploration, and surveillance. These algorithms can enable robots to navigate autonomously and effectively, contributing to increased productivity, safety, and efficiency.

References

[1] Kobayashi Y, Kanai S, Kikumoto C, and Sakoda K 2022 Design and Fabricate of Reconnaissance Robots for Nuclear Power Plants that Underwent Accidents. Journal of Robotics and Mechatronics, 34(3), 523–526.

[2] Nosrati M S, Karimi R and Hasanvand H A 2012 Investigation of the * (Star) Search Algorithms: Characteristics, Methods and Approaches - TI Journals. *World Applied Programming*.

[3] Dijkstra E 1959 A note on two problems in connexion with graphs. Numerische Mathematik, 1(1), 269–271.

[4] Hart P E, Nilsson N J and Raphael B 1968 A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics, 4(2), 100–107.

[5] Mishra S and Bande P 2008 Maze Solving Algorithms for Micro Mouse. 2008 IEEE International Conference on Signal Image Technology and Internet Based Systems, 86–93.

[6] Maravall D, de Lope J and Fuentes J P n.d. Visual Bug Algorithm for Simultaneous Robot Homing and Obstacle Avoidance Using Visual Topological Maps in an Unmanned Ground Vehicle. Bioinspired Computation in Artificial Systems, 301–310. https://doi.org/10.1007/978-3-319-18833-1_32

[7] Shaikh F K, Chowdhry B S, Zeadally S, Hussain D M A, Memon A A and Uqaili M A 2014 IBA: Intelligent Bug Algorithm A Novel Strategy to Navigate Mobile Robots Autonomously. In Communication Technologies, Information Security and Sustainable Development (Vol. 414, pp. 291–299). Switzerland: Springer International Publishing AG.

[8] McGuire K N, de Croon G C H E and Tuyls K 2019 A comparative study of bug algorithms for robot navigation. Robotics and Autonomous Systems, 121, 103261–.

[9] Ng J and Bräunl T 2007 Performance comparison of Bug navigation algorithms. Journal of Intelligent & Robotic Systems, 50(1), 73–84.

[10] Oroko J A and Nyakoe G N 2022 Obstacle avoidance and path planning schemes for autonomous navigation of a mobile robot: a review. In Proceedings of the Sustainable Research and Innovation Conference (pp. 314-318).

[11] Sankaranarayanan A and Vidyasagar M 1990 A new path planning algorithm for moving a point object amidst unknown obstacles in a plane. Proceedings., IEEE International Conference on Robotics and Automation, 1930–1936 vol.3.

[12] Lumelsky V and Stepanov A 1986 Dynamic path planning for a mobile automaton with limited information on the environment. IEEE Transactions on Automatic Control, 31(11), 1058–1063.

[13] Al-Haddad A, Sudirman R, Omar C, Koo Yin Hui and bin Jimin M R 2012 Wheelchair motion control guide using eye gaze and blinks based on bug 2 algorithm. 2012 8th International Conference on Information Science and Digital Content Technology (ICIDT2012), 2, 438–443.

DOI: 10.54254/2755-2721/33/20230278

[14] Menegatti E, Michael N, Berns K and Yamaguchi H 2015 A Minimalistic Quadrotor Navigation Strategy for Indoor Multi-floor Scenarios. In Intelligent Autonomous Systems 13 (Vol. 302, pp. 1561–1570). Springer International Publishing AG.
[15] Kamon I and Rivlin E 1997 Sensory-based motion planning with global proofs. IEEE Transactions on Robotics and Automation, 13(6), 814–822.
[16] Lee S, Adams T M and Ryoo B 1997 A fuzzy navigation system for mobile construction robots. Automation in Construction, 6(2), 97–107.
[17] Al-Haddad A A, Sudirman R and Omar C 2011 Guiding Wheelchair Motion Based on EOG Signals Using Tangent Bug Algorithm. 2011 Third International Conference on Computational Intelligence, Modelling & Simulation, 40–45.
[18] Tomita M and Yamamoto M 2008 A Navigation Algorithm for Avoidance of Moving and Stationary Obstacles for Mobile Robot. Nihon Kikai Gakkai ronbunshū. C, 74(748), 2976–2984.
[19] Kamon I, Rimon E and Rivlin E 1998 TangentBug: A Range-Sensor-Based Navigation Algorithm. The International Journal of Robotics Research, 17(9), 934–953.