16th IEEE International Conference on Control Applications
Part of IEEE Multi-conference on Systems and Control
Singapore, 1-3 October 2007

K-Bug, A NEW BUG APPROACH FOR MOBILE ROBOT'S PATH PLANNING

Ricardo A. Langer, Leandro S. Coelho and Gustavo H. C. Oliveira
Industrial and Systems Engineering Graduate Program - PPGEPS
Pontifical Catholic University of Paraná
Curitiba/PR/Brazil - Zip Code 80215-901
Email: ricardo_langer@yahoo.com.br, leandro.coelho@pucpr.br and gustavo.oliveira@pucpr.br

***Abstract*— This work presents the K-Bug algorithm, a new method for path planning of mobile robots belonging to the Bug family. The main idea of the algorithm may be used to improve the performance of existing methods of path planning that use local information, or as an entirely new method, if global information is available. It's also presented a short comparison of the methods found in literature, proving its efficiency, low computational cost and high robustness, even in complex environments.**

I. Introduction

Nowadays, mobile robotics applications are found in a wide range of areas, such as: manufacturing, transport, goods storage, medicine, military applications, computer games, spatial exploration, etc. [1] [2] [3] [4] [5]. Although different control approaches have been tried for mobile robots, they still have not caused an impact in domestic applications. This fact is due to its lack of a robust, reliable and flexible control system which operates in a dynamical, irregular and human populated environment. With so many different applications, there is an increasing need to find methods capable of realizing path planning for more and more complex environments, in a fast and reliable way. The problem of mobile robots path planning is defined in [6] as follows: "for a given robot and an environment description, plan a route between two specific locations, which must be clear of obstacles and attend all the optimizations criteria". The study of path planning can be divided in two main categories: local and global. In the first case, the planning is based on the information given by sensors (ultrasonic, infrared, laser, cameras) installed on the robot, witch provide details about the unknown environment [7].

In the global planning case, the environment's model is precisely defined, and the navigation is performed with the information known a priori ([8] [9]). This approach improves the convergence towards the destination point, but have a high computations cost to build and maintain the environment models.

Genetic algorithms, due to its generality, has been used also to solve the path planning problem. However, its lack of specialization results in basically three inconveniences: (i) the results are not optimum (ii) high computation cost and (iii) too many parameters must be chosen and tuned by the user ([10] [11] [12]).

The artificial potential fields method is specialized in path planning. It has a less amount of parameters needing adjustments and with less computations cost than the genetic algorithms. Besides this two advantages, it can be pointed out its ability to function in dynamical environments, which are environments with obstacles that can change their position and shape during the movement of the robot. The disadvantage of the method is a problem with local minima, which is particularly difficult to solve [13] [14] [15].

There is a class of algorithms that conduct a enumerative search in the environment through heuristic functions. In 1968, the A* (A-Star) algorithm was introduced by Hart et al. [16] , which is considered until now the most efficient method of heuristic search. Its efficiency lies in the fact that the best path can always be found. Amongst its disadvantages is the need to decompose the environment in a way to make the search possible, increasing even more its already high computational cost [16] [17] [18].

A class of algorithms that plan the trajectory of mobile robots considering local and global information are the Bug Algorithms. One of the first created within this context is called Bug2 [19]. This algorithm operates switching between two simple behaviors: (i) the movement to-wards the objective and (ii) the movement around an obstacle.

Lumelsky and Skewis [20] propose an improvement in the Bug2 with the VisBug, which is an enhancement to the condition that the robot uses to stop contouring an obstacle and resume the movement to the goal, the so called *leaving condition*. Such improvement generates short cuts in the path.

In this context, Kamon and Rivlin [21] created the DistBug which is characterized by another alteration in the leaving condition. Under certain special conditions the convergence of the DistBug can be proved.

In the present paper, a new method for path planning is presented, called K-Bug. Although this method is inspired in the bug algorithms, it changes the paradigm of contouring all the obstacles to the idea of contouring only the essential to reach the destiny point. A greater amount of information about the environment is used, generating

1-4244-0443-6/07/$20.00 ©2007 IEEE.

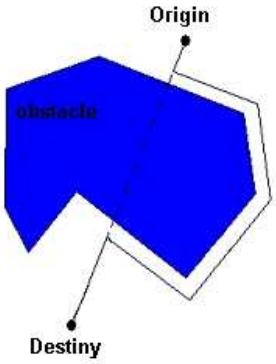

Fig. 1. Trajectory generated by the Bug2 algorithm proposed by Lumelsky and Stepanov. The moment when the path stop contouring the obstacle and goes towards to the destiny is called leaving condition

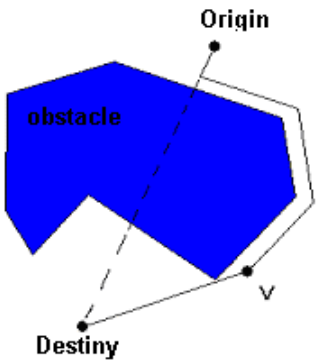

Fig. 2. Trajectory generated by the VisBug algorithm proposed by Lumelsky and Skewis.

better paths when compared to the traditional algorithms.

The rest of the paper is divided as follows. In section 2 are shown particularities of the Bug2, VisBug and DistBug algorithms. On section 3 the K-Bug is presented. On section 4 experimental assays are shown and it is done a comparison between the K-Bug and the classical algorithms, including the A*. Finally, in section 5 the conclusions are addressed.

II. The Bug Algorithms

All Bug type algorithms are based on the fact that the shortest distance between two points is a straight line. Therefore, the efforts to find the path for a mobile robot are such that to make it as close as possible to the straight line that cross the start and the end point. When this cannot be obtained as, for example, an environment with obstacles, the algorithms tries first to contour the obstacles and then it resumes the path by following the Origin-Destiny line.

In the Bug2 algorithm, the robot starts moving directly toward the destination. When an obstacle is found, the robot begins contouring it, only resuming to move to the destiny point when a certain leaving condition is met. In this method the leaving condition consists on finding the Origin-Destiny line while contouring an obstacle. The figure 1 shows an example of a trajectory generated by the algorithm.

It is easy to see that, even for simple obstacles, the method cited is not able to find the shortest path. Amongst its advantages are the low computational cost and the guarantee of finding at least one plausible path.

The main difference between the Bug2 and the VisBug relies in the leaving condition. While the Bug2's leaving condition requires the robot reaching the Origin-Destiny line, in the VisBug algorithm, the robot should be able only to see the line to trigger the condition. The figure 2 shows an example of a trajectory generated by the VisBug in order to contour an obstacle. From the point V, the Origin-Destiny line is visible to the robot, thus its direction is changed towards to the destiny point.

The DistBug algorithm incorporates, basically, two contributions in relation to the previous methods: (i) a more aggressive leaving condition without in-creasing the computational cost and (ii) a method to determine which side of the obstacle should be con-toured.

The leaving condition is taken to assure that the next point of collision is closer to the goal than the previous point of collision. In order to make this work correctly, the robot must continuously measure the free distance F, that is, the unobstructed distance to any obstacle toward the destination point T. After a collision is detected the robot initiates the contouring mode. The shortest distance to the destiny point T measured since the last collision, $dmin(T)$, is also stored by the robot. Taking $d(X,T)$ for the distance from the point X to the destiny T, the leaving condition is triggered when one of the following occurs [21]:

If the equation (1) is met, it means that the destiny point is visible. If the equation (2) is true, it means that it is more advantageous to go toward the destiny point than to keep contouring the obstacle. The parameter STEP is constant, and according to the author's suggestion, equal to the minimal distance between two objects in the environment.

$$d(X,T) - F \leq 0 \tag{1}$$

$$d(X,T) - F \leq d_{min} - \text{STEP} \tag{2}$$

The second point related to the DistBug algorithm is a method to determine which side of the obstacle should be contoured, through local sensing, together with a search manager. These procedures often contribute for generating shorter paths and can be used not only with the DistBug, but also with any other bug algorithm. Without these procedures, considering environments containing only convex obstacles, the performance of the DistBug is the same as the VisBug. The differences appear when, during the

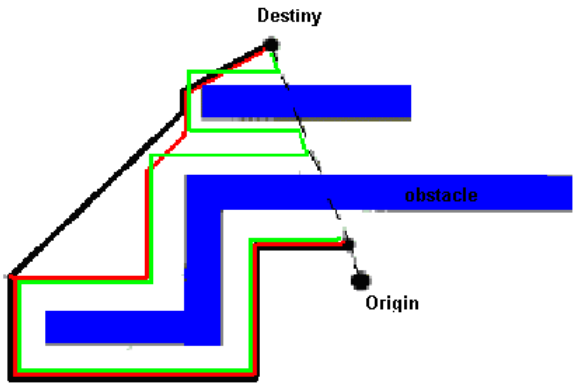

Fig. 3. Comparison between a trajectory generated by the Bug2 (green), VisBug (red) and DistBug (black)algorithms.

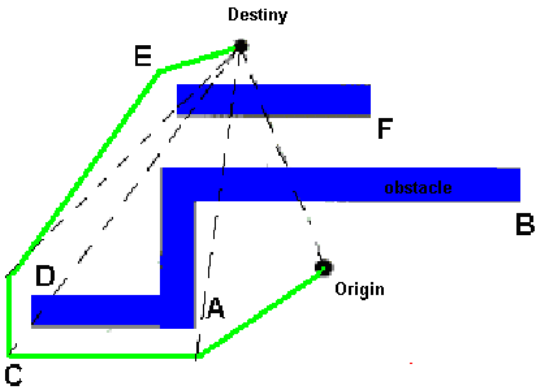

Fig. 4. Trajectory generated by the K-Bug algorithm.

contour mode, it is difficult to visualize the Origin-Destiny line. In such cases, the DistBug generates shorter paths than the VisBug algorithm as shown in figure 3.

The DistBug algorithm is an example of how simple rules can generate a complex path using sensor based techniques. The algorithm is reactive in the way that it depends on local sensing to make local decisions.

Kamon e Rivlin (1997) also demonstrated that the DistBug may oscillate and produce paths with several cycles before converge to a trajectory to the destiny point. This is expected in all the methods that privilege the information gain through local sensing.

In any of the discussed algorithms, if the robot makes a full circle around the destiny point it means that the goal is inaccessible.

III. The K-Bug algorithm

The K-Bug algorithm and the methods described in the previous section are based on the same concept, however, the K-Bug is not limited to the origin-destiny line. The major contribution of this new method relies on the fact that the algorithm leaves the origin-destiny line as soon as a new collision is found by local sensing. When such event happens, the robot (or the trajectory planner) has two choices: contour the obstacle by its left side or by its right side. The decision is taken verifying, in each side, the further visible point from the robot, that still belongs to the obstacle. Between this two options, the robot picks the side which has the closest visible point. Such point will be the next waypoint.

An example of this procedure is illustrated by figure 4. Let us assume that the trajectory planner, following the origin-destiny line, detects an obstacle. In this situation, the two obstacle points with the longest distance from the collision point, but are still visible by the robot, are the vertices A and B. Since the vertex A is closer than B, this is the one where the robot drives to. When the robot reaches it, a new collision is again detected in the destiny's direction (from point A). In this new position the two furthest points are vertices B and C. Vertex C is picked to be the next waypoint and so on.

By following this procedure, the K-Bug algorithm focuses on contouring only the necessary points in order to avoid the obstacle, leaping from one extremity to the other. However, in order for this method to work properly, the same requirements as the other methods are needed, such as a perfect sensing.

A. Assuming complete knowledge about the environment

There are some applications where all information about the environment is known *a priori*, then it can be considered in the path planning procedure. The information about the obstacles geometry and position can are added to the algorithm in order to determine, in a more efficient way, the intermediate points that should be inserted to the trajectory. Such procedure could solve the convergence problems inherent to the methods that consider only local information.

The *modus operandi* consists in, starting from the origin-destiny line, to search for a point where a collision occur. Then, the aim is to contour the obstacle it by inserting waypoints in the trajectory. The waypoint are selected among the vertex of the obstacle that caused the collisions. The computed waypoint is considered the next origin and the entire process is repeated until there are no more collisions left.

With all environment information available, the line *od* (Origin-Destiny) is plotted. This line will cross the obstacles, passing through them by cutting its edges. These are the collisions that must be avoided. Now, considering a straight line and a obstacle defined as a irregular polygon, it is well known that collisions will, in the same obstacle, always come up in pairs, an entry and an exit point. To select a collision entry point, it is sufficient to pick the one closer to the origin point. This collision entry point is then called C_1.

The second closest collision from the Origin point is an exit collision. Such point is then called C_2. To find out by

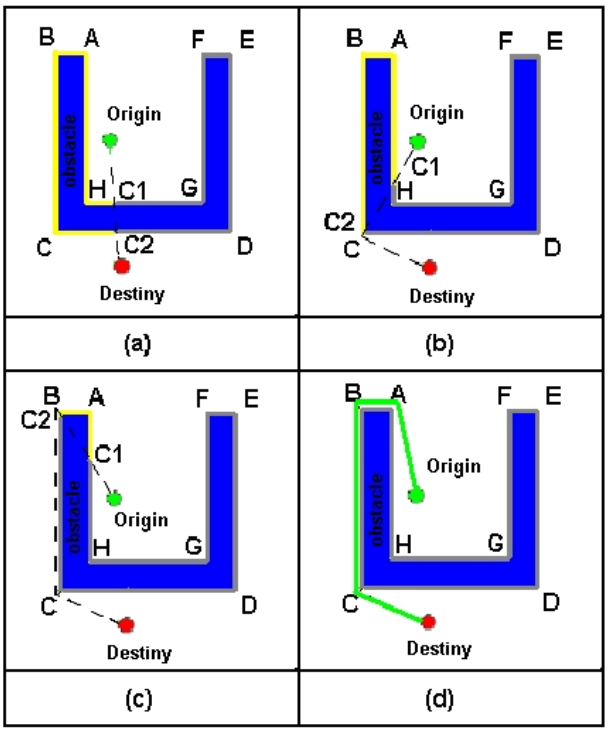

Fig. 5. Trajectory generated by the K-Bug algorithm step-by-step.

which side the obstacle will be contoured, the perimeter of the obstacle is measured from C_1 to C_2 and from C_2 to C_1. The side with smallest perimeter will be selected.

The selected side (or section) between C_1 and C_2 contains a finite number of vertices. The vertex of this section having the biggest distant from rm will be inserted as the next waypoint in the trajectory between the origin and the destiny points. This new point is then called V_1. Now, the path has two segments, the one given by Origin-V_1 and the V_1-Destiny segment. The entire procedure is repeated recursively for the segment V_1-Destiny until all the collisions are avoided.

The procedure conducted to find out which vertex (x_i, y_i) has the biggest distance from the line rm is presented in following. It is done by calculating the Euclidian distance formula between a point (the vertex) and a line $(y = \alpha x + \beta)$, in the $\mathbb{R}^2$, as shown by equation 3.

$$d = \frac{|x_i \times \alpha + \beta - y_i|}{\sqrt{\alpha^2 + 1}} \tag{3}$$

A trajectory generation example is shown in figure 5. The obstacle has a "U" shape. This shape is known by causing some instability problems and local minima in common methods [21].

Initially, the trajectory is firstly composed by the Origin-Destiny segment as shown in figure 5(a). Two collision points can be found, they are C_1 (between obstacle's vertices G and H) and C_2 (between obstacle's vertices G and H). Two sections are clearly identified, sections

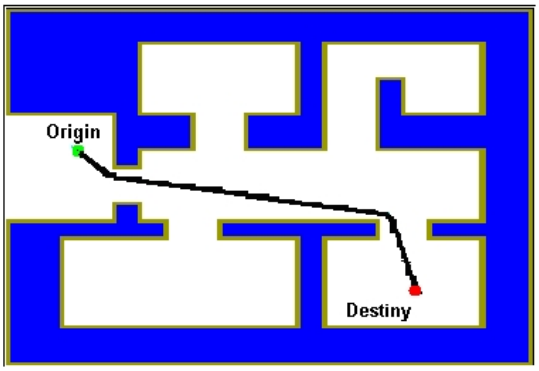

Fig. 6. Trajectories generated by the K-Bug and the A* algorithms are identical for this environment.

$C_1 - H - A - B - C - C_2$ and $C_1 - G - F - E - D - C_2$. The first is the one with a smaller length. Among the vertices included in the section with smaller length, the one which is the most far from the line containing the segment Origin-Destiny is the vertex C. This will be added to the trajectory as a waypoint, as shown in figure 5(b).

The process is repeated, but now considering the segment Origin-C instead of Origin-Destiny. According to the same criteria, vertex B is selected and inserted between the Origin point and the C vertex, as shown in figure 5(c). Finally, the vertex A is inserted in the trajectory, linking the Origin point to the Destiny, resulting in a free of collisions path, as shown in figure 5(d).

It may happen a case where the obstacle has more than one pair of collisions. In such cases, the one closest to the obstacle middle, according to its perimeter's length, will be one first processed.

IV. Simulation Results

Using the total path length as the heuristic rule for the A* algorithm, it is a mathematical certainty that the shortest path is always generated, assuming that it exists. Therefore, in the subsequent figures are presented comparisons between the paths generated by the A* and the K-Bug algorithms, with the intention to analyze their similarity. In order to make viable the A* processing, the environments were decomposed using visibility graphs [22]. Both of the algorithms were coded using National Instruments' LabView™ software.

Figure 6 exhibits an environment that resemblances an office containing several rooms. The trajectories produced by the A* and the K-Bug, plotted in black, are coincident.

For the environment shown in figure 7 the trajectories are similar but not equal.

The figure 8 shows an office like environment. All the obstacles in the room, such as chairs, desks and walls, were mapped as irregular polygons. Although the origin and destiny points are close to each other, two waypoints ware placed on the other side of the room to test a more sophisticated path. Because the K-Bug is naturally a recursive

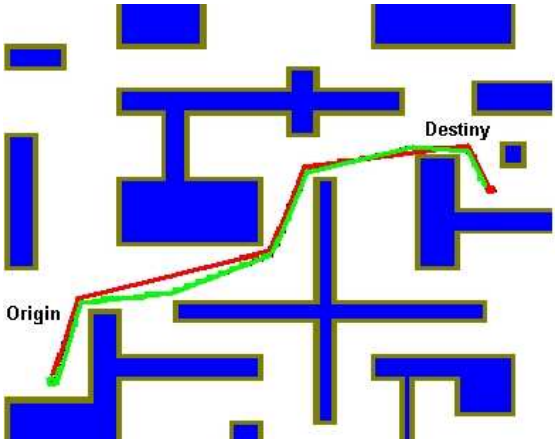

Fig. 7. Trajectories generated by the K-Bug (green) and the A* (red) algorithms are similar.

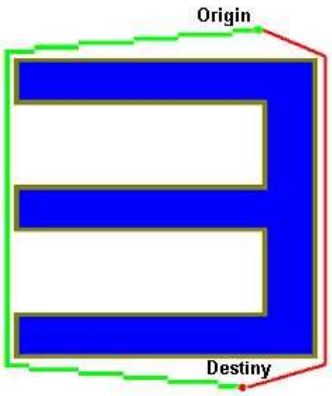

Fig. 9. Path generated by the K-Bug (green) is not optimum, as it can be he A* (red).

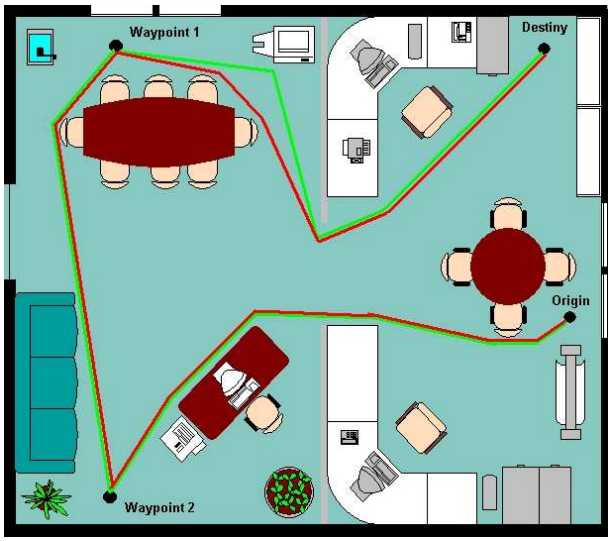

Fig. 8. Office like environment. Trajectories generated by the K-Bug (green) and the A* (red) algorithms are similar.

algorithm, there is no alteration on its behavior concerning the intermediary waypoints. The two preset waypoints are treated exactly the same way as the waypoints inserted by the algorithm as described in the previous section. The A* algorithm had to be processed separately for each one of the path segments.

An example of a situation that K-Bug chooses the wrong side of an obstacle to contour is in figure 9. Due to the fact that, taking into consideration any of the three pair of collisions, during the algorithm's processing, the obstacle's perimeter's length is larger on its right side. Therefore, the K-Bug chooses the left side for contouring.

V. Conclusion

In this paper, three path planning algorithms are studied and the K-Bug algorithm is presented. This new method may be used to improve the performance of local search algorithms or as a global path planning method. The K-Bug has proven itself as an efficient and robust algorithm, even when compared to other consecrated path planning methods in the literature.

Taking into consideration the global information, the algorithm proposed has revealed itself sufficiently robust to find plausible trajectories even in environments that usually cause problems of local minima for the classical methods (*i.e.* genetic algorithms and potential fields).

In environments of low complexity, precisely, with few obstacles and not many bifurcations, the A* may be faster than the K-Bug. For environments with the same complexity, the K-Bug takes the advantage of visiting only the obstacle's vertices that generate collisions. Considering the environments presented on figures 6, 7 and 8 the K-Bug performed respectively 16%, 25% and 21% faster than the A* algorithm .

Amongst the advantages of the K-Bug algorithms when compared to the classical methods are: (*i*) Few iterations to find the goal. (*ii*) There is no need to decompose the environment before the process is initiated. (*iii*) There is no need to process the obstacles that don't produce collisions. (*iv*) The algorithm can evolve trajectories in order to visit pre-settled points on the map.

The K-Bug does not need to decompose the entire environment, as it is needed in the enumerative search methods such as the A*. Operations often used to decompose the environment like cell decomposition, visibility graphs and Voronoi diagrams are so time consuming that, in several cases, surpass the processing time cost of the search algorithm itself.

The algorithm's performance is directly influenced by the side to contour criteria. In this paper the criteria used was the obstacle's perimeter. As future research it could be used the obstacle's mass center. The obstacle's side which the mass center is closer to the Origin-Destiny line should be the side contoured.

Finally, the algorithm was not designed to operate in

dynamical environments, which are environments with obstacles that its position or/and shape change during the robot movement. Therefore, the method's utilization is viable only if it is possible to process the entire algorithm at each environment alteration, and the robot's speed is greater then the obstacle's movement.

References

[1] L. Sciavicco and B. Siciliano, *Modelling and Control of Robot Manipulators*. Springer, 2000.

[2] Y. Takahashi, T. Komeda, and H. Koyama, "Development of assistive mobile robot system: Amos." in *Advanced Robotics*, vol. 18, no. 5, 2004.

[3] J. Mclurkin, "Using cooperative robots for explosive ordnance disposal," Massachussets Institute of Technology. Cambridge, MA USA: Artificial Intelligence Laboratory, 1995.

[4] M. L. Littman, T. Dean, and L. P. Kaelbling, "Markov games as a framework for multi-agent reinforcement learning," in *Eleventh International Conference on Machine Learning*. San Francisco, CA: Morgan Kaufmann, 1994, pp. 157–163.

[5] NASA, "Mars pathfinder project," California Institute of Technology, Jet Propulsion Laboratory. Pasadena, CA, Tech. Rep., 1997.

[6] J. Z. M. Xiao, L. Zhang, and T. Krzysztof, "Adaptive evolutionary planner/navigator for mobile robots," in *IEEE Trans. on Evolutionary Compututation*, vol. 1, no. 1, 1997, pp. 18–28.

[7] H. R. Beom and H. S. Cho, "A sensor-based navigation for a mobile robot using fuzzy logic and reinforcement learning," in *IEEE Trans. Syst. Man Cyber*, vol. 25, no. 3, 1995, pp. 464–477.

[8] T. Aoki and M. Matsuno, "Motion planning for multiple obstacles avoidance of autonomous mobile robot using hierarchical fuzzy rules," in *Proc. IEEE Int. Conf. on Multisensor Fusion and Integ. for Intel. Systems*, Las Vegas, 1994, pp. 265–271.

[9] G. Foux, M. Heymann, and A. Bruckstein, "Two-dimensional robot navigation among un-known stationary polygonal obstacles," in *IEEE Trans. Robot. Automat*, vol. 9, 1993, pp. 96–102.

[10] D. Goldberg, "Genetic and evolutionary algorithms come of age," in *Communications of the ACM*, vol. 37, no. 3, 1994, pp. 113–119.

[11] ——, "Genetic algorithms in search optimization and machine learning," in *Communications of the ACM*, 1989.

[12] T. Bäck, U. Hammel, and H. Schwefel, "Evolu-tionary computation: Comments on the history and current state," in *IEEE Transactions on Evolu-tionary Computation*, vol. 1, no. 1, 1997, pp. 3–17.

[13] J. Agirrebeitia, R. Avile, I. F. Bustos, and G. Ajuria, "A new APF strategy for path planning in environments with obstacles - esta referencia estÁ parecendo uma tese. estÁ citada errada?" University of the Basque Country. Alameda de Urquijo: Mechanical Engineering Department, 2005.

[14] C. Lin, C. Chen, and C. Chen, "Path planning and dynamic control of a redundant robot manipulator for conveyor tracking," *International Journal of Systems Science*, vol. 30, no. 5, pp. 491–503, 1999.

[15] P. Vadakkepat, C. K. Tan, and W. Ming-Liang, "Evolutionary artificial potential fields and their application in real time robot path planning," in *Congress of Evolutionary Computation*, San Diego - CA, 2000, pp. 256–263.

[16] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basic for the heuristic determination of minimum cost paths," *IEEE Trans. Systems Science and Cybernetics*, vol. 4, no. 100-107, 1968.

[17] E. Rich, *Inteligência Artificial*. São Paulo: McGraw-Hill, 1988.

[18] E. A. Bender, "Mathematical methods in arti-ficial intelligente," in *IEEE Computer Society Press*, 1996.

[19] V. J. Lumelsky and A. A. Stepanov, "Path-planning strategies for a point mobile automa-ton moving amidst obstacles of arbitrary shape," in *Algorithmica*, vol. 2, 1987, pp. 403–430.

[20] V. J. Lumelsky and T. Skewis, "Incorporating range sensing in the robot navigation func-tion," *IEEE Trans. Systems, Man and Cybernetics*, vol. 20, pp. 1058–1068, 1990.

[21] I. Kamon and E. Rivlin, "Sensory-based motion planning with global proofs," *IEEE Transations on Robotics and Automation*, vol. 13, no. 6, pp. 814–822, 1997.

[22] J. C. Latombe, *Robot Motion Planning*. Academic Publishers, 1991.