A Comparison of Path Planning Algorithms for Omni-Directional Robots in Dynamic Environments

Felipe Haro and Miguel Torres,
Department of Electrical Engineering
Universidad Católica de Chile
Santiago, Chile
felipeharo@gmail.com, mtorrest@ing.puc.cl

***Abstract*— The scope of this paper is to analyze and compare three path planning methods for omni-directional robots, which are based on a) the Bug algorithm b) the Potential Fields algorithm, and c) the A* algorithm for minimum cost path with multiresolution grids. The approaches are compared in terms of computational costs and the resulting path lengths. Results obtained indicate that the Bug algorithm is a suitable choice for this type of application as its computational cost is lower than that of the other methods. Furthermore, minor modifications of the standard Bug algorithm, such as the tangent following modification, allow the path planner to handle well the situations encountered in typical multi-robot environments.**

***Index Terms*—Bug Algorithm, Potential Fields, A* Algorithm, Local Multiresolution, Path Planning, Obstacle Avoidance**

I. Introduction

In robotics, the ability of a mobile robot to plan its own path in a dynamic environment is essential for achieving a higher degree of autonomy. Several path planning methods have been proposed in last decades, see for example [1] and [2], specifically to solve the path planning problem in a static environment. Thus a comparison assessing the performance and advantages of the main approaches in a dynamic environment would be very useful for engineers working in the design of mobile robots today. The path planning strategies evaluated in this paper are summarized in Table 1. The contribution of this paper is in the comparative assessment of these techniques as applied to robots in a dynamic environment. No previous work has been found in the literature in which the practical advantages and performance of the methods in analyzed under such the conditions of a dynamic environment.

II. Path Planning

Real Time path planning consists in finding a feasible path from one state to another in a sufficiently small time, in such a way that the chances for collision to occur are reduced to a minimum. In each of the methods evaluated here, the agent (controlled robot) is considered to be a particle, while the obstacles are assumed to be larger than what they really are by an amount equal to the radius R of the robots.

TABLE 1
SUMMARY OF PATH PLANNING TECHNIQUES CONSIDERED IN THIS STUDY

Name	*Modified*	*References*
Bug Algorithm	Yes	[1],[2],[3]
Potential Fields	No	[1],[2],[4],[5]
A* With Multiresolution Grids	No	[2],[6],[7]

A. Modified Bug Algorithm

The Bug Algorithm is a non-optimal method to find a route between two points. The standard algorithm consists in generating a direct path from the start position to the goal while surrounding the obstacles always around the same direction. It is useful when finding a secure path to the goal is more important than arriving quickly. The standard Bug Algorithm is illustrated in figure 1.

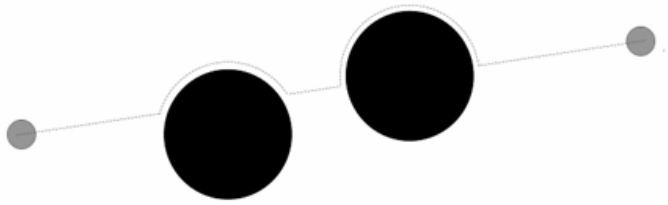

Fig. 1. Standard Bug Algorithm.

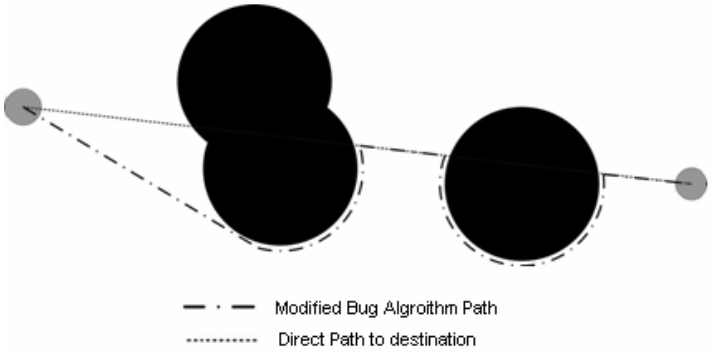

Fig. 2. Modified Bug Algorithm.

It is not difficult to see that this algorithm alone cannot yield optimal (shortest) paths, thus it requires modifications. One such modification consists in generating a tangent to the obstacle from the starting position and then continue with the normal bug algorithm as illustrated in figure 2. To select the side which is most likely to be the shortest one, the area to each side of the line that defines the direct path from the starting position to the goal is computed. Although it is possible that in some situations the smallest area does not imply the path is the shortest one, computing the area is faster than calculating the perimeter. Once the areas are computed, the tangent to the side of smallest area is calculated. With this approach, a better path can be dynamically generated, as also explained by the pseudocode for the Modified Bug Algorithm shown in figure 4.

An important problem to be considered is that this method, in contrast to other methods, implies the robot must in principle follow the walls of the obstacle. Hence, to prevent the robot from bumping the obstacle it is necessary to add a safety margin F to the radio of the obstacle. The margin F will have an arbitrary maximum value $F_{\max}$ and will decrease when the distance from the agent to the obstacle D is less than $R + F_{\max}$, which is possible because the margin is a non-occupied area of the environment and the goal or the start point can perfectly be inside the margin F. In such a case, the new value of F will be $F = F_{\max} - (D - R)$. Figure 3 shows a circular obstacle of radius r, enlarged by the radius of the robot $R + r$ and the safety margin F.

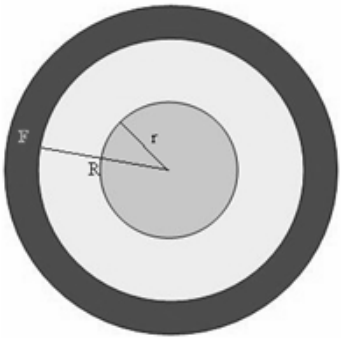

Fig. 3. Obstacles dimensions and the safety margin *F* for the Modified Bug Algorithm.

Modified Bug Algorithm

Initial Parameters

(x_i, y_i)	Initial coordinates of the robot
(x_f, y_f)	Coordinates of the goal position
n	Number of obstacles on the field
$(x[i], y[i])$	Coordinates of the obstacles, $i = 1..n$
T	Planned Trajectory (starting empty)

Set a direct path L from (x_i, y_i) to (x_f, y_f)

Iteration i (for $i = 1..n$)
Group the obstacles that are next to each other (i.e. without free space between them) to form a unique obstacle
Set an ID for every obstacle
End Iteration

Iteration (starting at (x_i, y_i), pixel by pixel along L, until the goal (x_f, y_f) is reached)
IF (next pixel is free)
Append next pixel to T
ELSE IF (next pixel is occupied)
Set the ID of the obstacle if it is the first one encountered as ID 0
Surround the obstacle along its shortest side until L is reached
Append all the pixels to T
END IF
End Iteration

IF (obstacle with ID 0 is defined)
Set a tangent Tg from (x_i, y_i) to the obstacle with ID 0 on the side where T is defined
Set (x_t, y_t) as the position where Tg is tangent to ID 0
Set T as the trajectory concatenating:
- Tg from (x_i, y_i) to (x_t, y_t)
- T from (x_t, y_t) to (x_f, y_f)

END IF

Fig. 4. Modified Bug Algorithm pseudocode.

In a dynamic situation the agent is continuously checking if the goal is in its direct line-of-sight, and if so, the path planner defines a new reference path L as the line from the agent's current location to the goal. This results in a shorter path than the one that would be traversed if the agent returns to the original path L.

A. Potential Fields Method

Several developments relying on potential field methods can be found in the literature [1], [2], [4], and [5]. In this approach, the concept of attraction and repulsion are employed to approach to the goal and evade obstacles, respectively. More specifically, the method assigns a force field value to each location in the environment, which is a combination of an attractive force that increases towards the goal, and a repulsive force that increases towards the obstacles. The force field is defined for each position coordinate according to:

$$F(x, y) = F_a(x, y) - \sum_{i=1}^{r} F_i(x, y)$$

where

$F_a = K_a \sqrt{(p_x - x_f)^2 + (p_y - y_f)^2}$ is the attraction force defined to be proportional to the distance between the current position (p_x, p_y) and the goal location (x_f, y_f), r is the number of obstacles and F_i is the repulsion force.

The repulsion force is arbitrary, and is related to the

distance from the current position (p_x, p_y) to the obstacle i.

In our implementation, we compute the Manhattan distance (L_1-norm) from the agent to the nearest obstacle, and analyze all the pixels in the environment that are at the same Manhattan distance from the agent as illustrated in figure 5. Once the pixel whose potential field gradient has maximum magnitude is found, a direct path to that pixel is set. This procedure is repeated until the goal is reached.

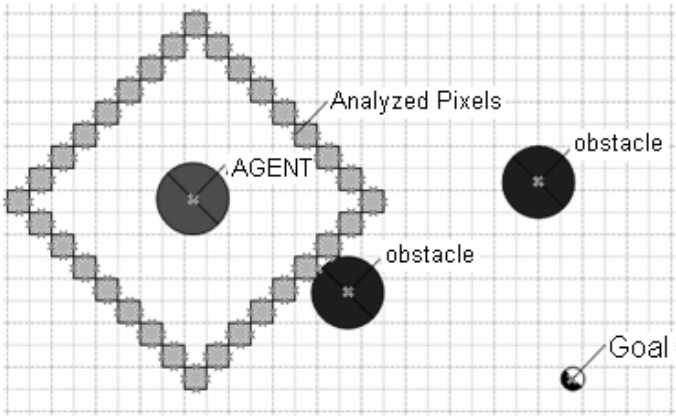

Fig. 5. Pixels analyzed in the gradient calculation of the Potential Fields Algorithm.

Potential Fields Method

Initial Parameters

(x_i, y_i)	Initial coordinates of the robot
(x_f, y_f)	Coordinates of the goal position
n	Number of obstacles on the field
$(x[i], y[i])$	Coordinates of the obstacles, $i = 1..n$
D	Influence distance of an obstacle
R	Obstacle radio
K_1, K_2	Arbitrary constants
T	Planned Trajectory (starting empty)

Set $U_{att}(x_k, y_k) = K_1 \| (xfinal, yfinal) - (x_k, y_k) \|$ as the attraction potential to the goal on the position (x_k, y_k)

Set the repulsion potential of obstacle i on the position (x_k, y_k) as

$$U^i_{aux}(x_k, y_k) = \begin{cases} \infty & if \ \| (x_o[i], y_o[i]) - (x_k, y_k) \| \le R \\ 0 & if \ \| (x_o[i], y_o[i]) - (x_k, y_k) \| > D, \\ K_2 (1 - \frac{\| (x_o[i], y_o[i]) - (x_k, y_k) \|}{D})^2 & else \end{cases}$$

$$U^i_{rep}(x_k, y_k) = \begin{cases} U^i_{aux}(x_k, y_k) if \ \| (x_f, y_f) - (x_k, y_k) \| > R/2 \\ U_{att}(x_k, y_k) if \ \| (x_f, y_f) - (x_k, y_k) \| \le R/2 \end{cases}$$

Iteration (starting at $(x_k, y_k) = (x_i, y_i)$, until goal is reached)

Set M as the Manhattan distance from (x_k, y_k) to the nearest obstacle

Iteration (for each pixel at a distance M from (x_k, y_k) in the field)

- Compute the potential force in the pixel
  $F(x_k, y_k) = \max(U_{att}(x_k, y_k), U^1_{rep}(x_k, y_k), \ldots, U^n_{rep}(x_k, y_k))$

End Iteration

Choose the pixel in the position (x_p, y_p) such that F is minimal

Set a direct path from (x_k, y_k) to (x_p, y_p) and append it to T

$(x_k, y_k) = (x_p, y_p)$

End Iteration

Fig. 6. Potential Fields Method pseudocode.

B. A Algorithm with Multiresolution Grids*

The A* algorithm [10] is a well known method to obtain an optimal path between nodes, where the nodes may be the cells or elements of the grid used to represent the environment. In some applications, such as a robotic soccer, the processing time of the standard A* algorithm can be considerably large because of the large number of nodes (pixels) that may be used to represent the environment. If images are used to represent the environment, reducing their resolution translates into a reduction of the number of nodes that describe the environment. This allows to gain computational speed at the expense of generating less accurate paths that may result in longer paths and less precise paths, leading possibly to collisions. This tradeoff between processing time and motion precision calls for a multiresolution approach, as shown in figure 6, in which a high resolution is employed to represent the environment in a vicinity of the agent, where positioning precision is more important, and a low resolution is used to represent the environment far from the agent, where changes in the obstacles locations are more likely to occur in the long run as they move, and hence, a coarse representation suffices to obtain approximated paths.

To move from one node to another the A* algorithm considers a cost function of the form $F=G+H$ in order to decide which is the best next node to move to from the current node. This cost function is the composition of two aggregated costs: G is the real accumulated cost to move to the next node from the initial node, while H is an optimistic heuristic function representing the cost of moving from the next node to the goal node. For path planning with obstacle avoidance applications, a suitable choice of G, is $G=D_t+D_o$, where D_t is the actual traversed distance from the initial node to the current node, and D_o is a penalizing cost added to the current node depending on its distance to nearby obstacles. Typical functions for the penalizing cost D_o are shown in figure 7. On the other hand, the heuristic function H is commonly defined as the Euclidian distance to the goal node.

The A* algorithm is explained in terms of its pseudocode shown in figure 8, see [10] for details.

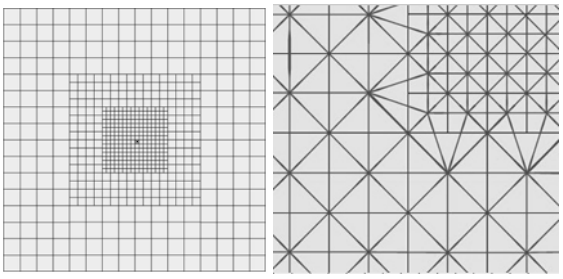

Fig. 6. a) Multiresolution grid, b) paths between nodes.

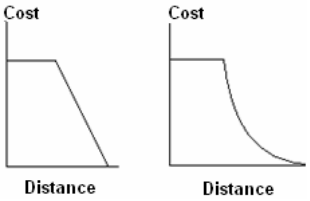

Fig. 7. Obstacle Proximity Penalizing Cost Function D_o.

A* Algorithm with Multiresolution Grids

Initial Parameters

(x_i, y_i)	Initial coordinates of the robot
(x_f, y_f)	Coordinates of the goal position
n	Number of obstacles on the field
$(x[i], y[i])$	Coordinates of the obstacles, $i = 1..n$
OPEN	List of nodes that need to be examined
CLOSE	List of nodes that have already been examined. Initially empty.

Set a multiresolution grid where (x_i, y_i) is the start node n_o

Create a search graph G, consisting solely of the start node, n_o.
Put n_o on OPEN

Iteration (until goal is reached)

- If OPEN is empty, exit the iteration with failure
- Select the first node on OPEN, remove it from OPEN, and put it on CLOSED. Call this node n
- If n is a goal node, exit the iteration successfully with the solution obtained by tracing a path along the pointers from n to n_o in G
- Expand node n, generating the set M, of its successors that are not already ancestors of n in G. Install these members of M as successors of n in G
- Establish a pointer to n from each of those members of M that were not already in G (i.e., not already on either OPEN or CLOSED). Add these members of M to OPEN. For each member, m, of M that was already on OPEN or CLOSED, redirect its pointer to n if the best path to m found so far is through n. For each member of M already on CLOSED, redirect the pointers of each of its descendants in G so that they point backward along the best paths found so far to these descendants
- Reorder the list OPEN in order of increasing f values. (Ties among minimal f values are resolved in favour of the deepest node in the search tree.)

End Iteration

Fig. 8. A Algorithm with Multiresolution Grids pseudocode [10].*

III. Testing Methodology

In order to assess the performance of the three algorithms, a qualitative comparison of the behavior of the agents moving against different types of static obstacles, including non-convex ones, is evaluated first. Then their specific performance measured in terms of the processing time and the planned path length is evaluated under the same three static scenarios: a) Case I: One obstacle obstructing the goal, b) Case II: Free path between the agent and the goal with non-blocking obstacle, c) Case III: Two obstacles contacting each other. All these scenarios are representative of situations commonly found in Small Size category of the RoboCup competitions. The algorithms are then tested in dynamic environments in which the obstacles move to obstruct the path to the goal. Their performance is then tested in terms of the feasibility of the generated trajectory and the final path length. Simulations were made using Matlab® with the simplifying assumption of perfect position information, i.e. without considering sensors or actuator noises. All the simulations were executed using 256x256 pixel images to represent the environment and obstacles represented by discs of 15 pixels radius.

Finally, the Bug Algorithm, which resulted to be the fastest one to compute, was implemented in C++ and tested for real-time performance in a dynamic environment involving two and five moving robots. The discussion of the results obtained is presented next.

IV. Experimental Results

The behavior of the agents using each of the algorithms with simple and complex non-convex static obstacles is shown in figure 9. It is possible to see that the Potential Fields Method has the big disadvantage of generating paths that get trapped on local minima of the potential function when the agent faces non-convex regions of the obstacle. It is worth mentioning that these trajectories can be regarded as an initial path which is updated as the agent and the obstacles of the dynamic environment move. In practice the resulting trajectories may be smoother than those corresponding to the initial planned path, since the trajectory is continuously updated. This may be seen in Figure 10 which shows a sequence of planned trajectories changing as the moving obstacles attempt to block the direct path from the initial agent's position to the goal. The trajectory finally traversed by the agent is smooth as is shown in frame 11 of figure 10. Figure 11 shows a similar sequence in which the Potential Fields Methods is used instead to plan the path in a simulation considering an obstacle that continuously attempts to block the goal.

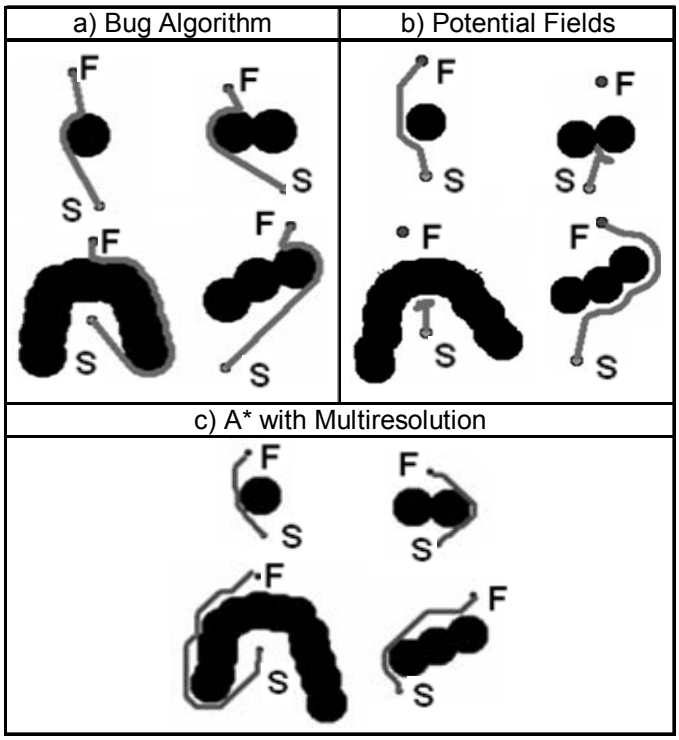

Fig. 9. Three algorithms under similar conditions, where S is the start and F is the goal.

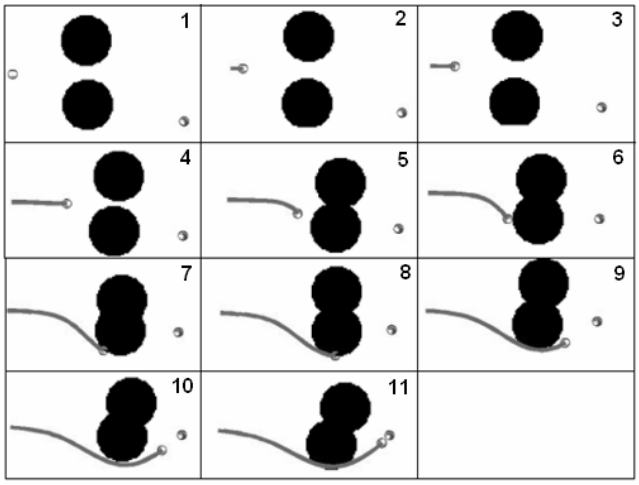

Fig. 10. Bug Algorithm, dynamic trajectory towards a goal.

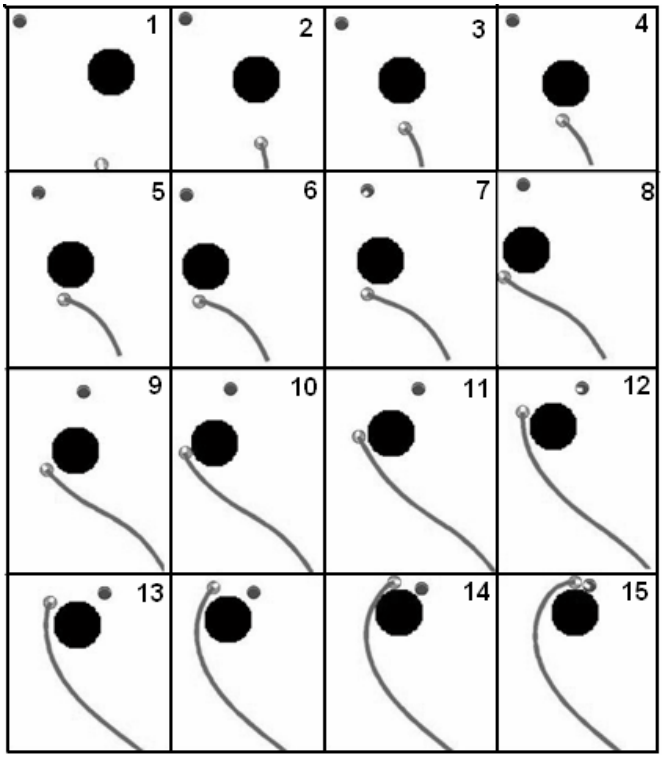

Fig. 11. Potential Fields, dynamic trajectory towards a goal.

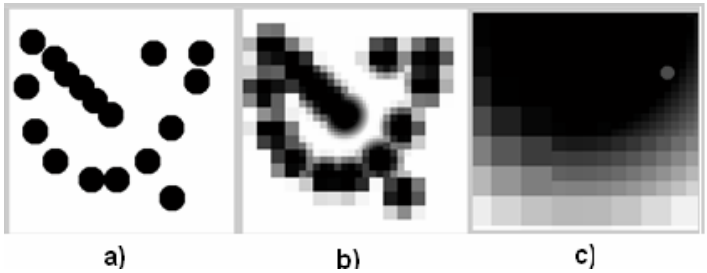

Fig. 12. a) Obstacles with full resolution, b) obstacles with multiresolution and costs associated to A algorithm, c) zoom to the center of the multiresolution image.*

Figure 12 shows a different resolution levels employed by the A* algorithm for path planning. In the method, the pixels represent nodes and the penalizing cost associated to the distance to the obstacle D_o (see figure 7) is represented here in terms of a grey scale in which darker pixels have a higher cost D_o.

A. Static obstacles

The performance of the algorithms is compared in terms of the three scenarios shown in Table 2. In the first situation one obstacle is standing on the direct path to the goal forcing the agent to move around. In the second situation, the obstacle is away from the path to the goal and in the third scenario two obstacles contacting each other block the goal. The results obtained in each of the cases are summarized in Table 3. The computation times T and path length D are expressed relative to the best one.

TABLE 2
COMPARISON OF THE PATH PLANNING ALGORITHMS FOR STANDARD SCENARIOS

Case I		Bug Algorithm	Potential Fields	A* with Multiresolution
Characteristic	One obstacle in the way			
Start	[128 128]			
Goal	[127 211]			
Position Obstacle 1	[122 174]			
Position Obstacle 2	None			
Case II:		Bug Algorithm	Potential Fields	A* with Multiresolution
Characteristic	No obstacles in the way, but one near			
Start	[128 128]			
Goal	[110 62]			
Position Obstacle 1	[122 174]			
Position Obstacle 2	None			
Case III:		Bug Algorithm	Potential Fields	A* with Multiresolution
Characteristic	Two obstacles in the way, near to each other			
Start	[128 128]			
Goal	[110 62]			
Position Obstacle 1	[100 100]			
Position Obstacle 2	[126 87]			

TABLE 3
COMPARISON OF THE METHODS EXPRESSED RELATIVE TO THE BEST RESULT, WHERE T IS THE PROCESSING TIME AND D THE LENGTH OF THE TRAJECTORY

	CASE I		**CASE II**		**CASE III**	
Method	T	D	T	D	T	D
Bug Algorithm	1,21	1	1	1	1	1
Potential Fields	1	1	1,2	1,1	∞	---
A* with Multiresolution	22,2	1,1	19,9	1	22,1	1,07

As a reference, the processing time of the bug algorithm in case II was of 179 ms and the length of the resulting path was of 68 pixels. The 179 ms processing time was obtained on an PC with a 2GHz AMD Athlon 32 bits processor, 250MB RAM, running Matlab 7.0 under Windows XP. It is worth pointing out that in Case II, the presence of an obstacle near the agent affects the processing time because of how potential

field was chosen.

B. Dynamic obstacles

Many are the possible situations an agent can face during a soccer match. We show here two cases chosen arbitrarily to verify the path feasibility of the generated trajectory as well as the path length. In the first case, one obstacle moves horizontally to the right and the second to the opposite direction, blocking the way to the goal (Figures 13, 14 and 15).

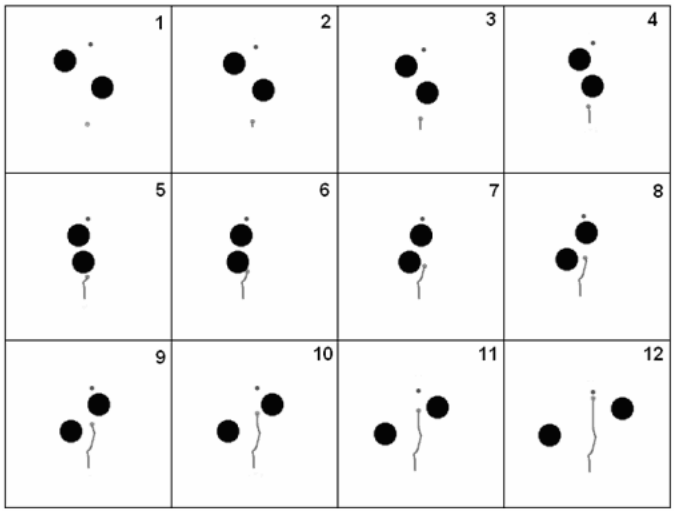

Fig. 13. Case I, Bug Algorithm, dynamic obstacles.

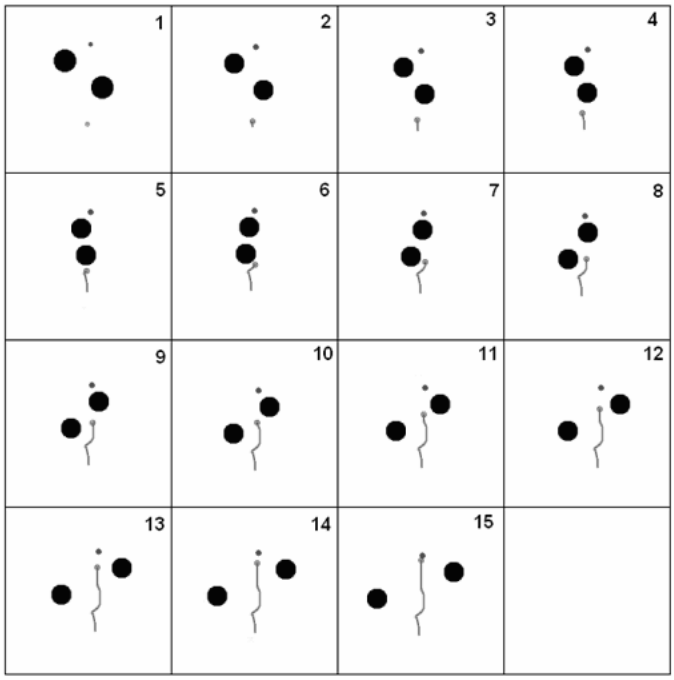

Fig. 14. Case I, Potential Fields, dynamic obstacles.

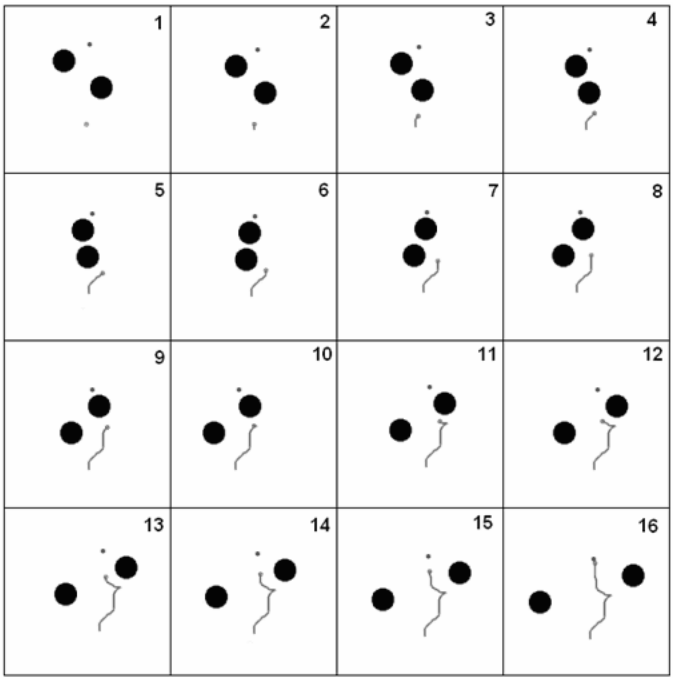

Fig. 11. Case I, A algorithm, dynamic obstacles.*

In the second situation, one of the obstacles moves vertically to collide with the agent (Figures 16, 17 and 18). To test the feasibility of the generated path, a common control method has to be chosen. The control laws employed are based on [8], and an example graph of the desired and obtained trajectories is shown on figure 19. This graph represents the case shown on Figure 13, where the Bug Algorithm is used. Trajectory control will not be analyzed, since is not the scope of this paper, and will be used for comparison purpose.

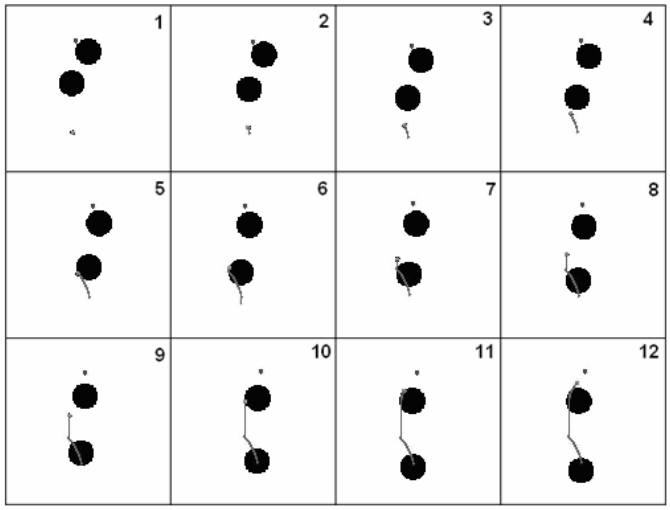

Fig. 16. Case II, Bug Algorithm, dynamic obstacles.

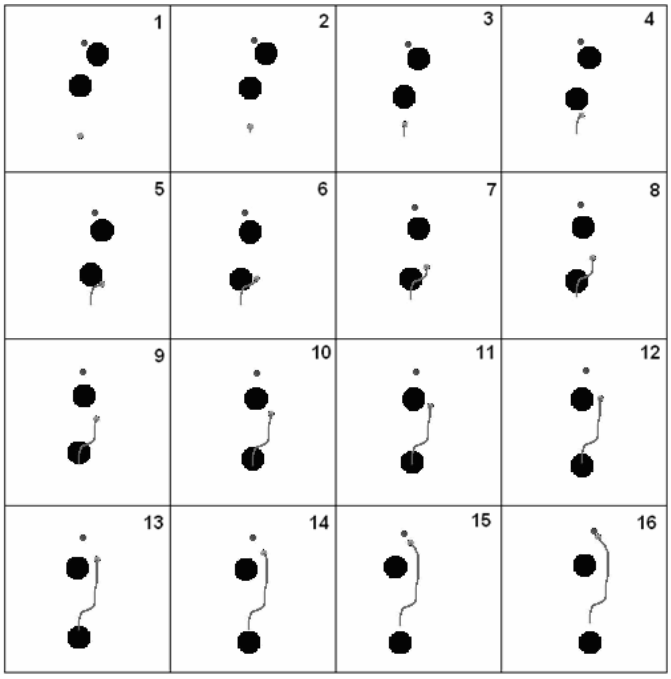

Fig. 17. Case II, Potential Fields, dynamic obstacles.

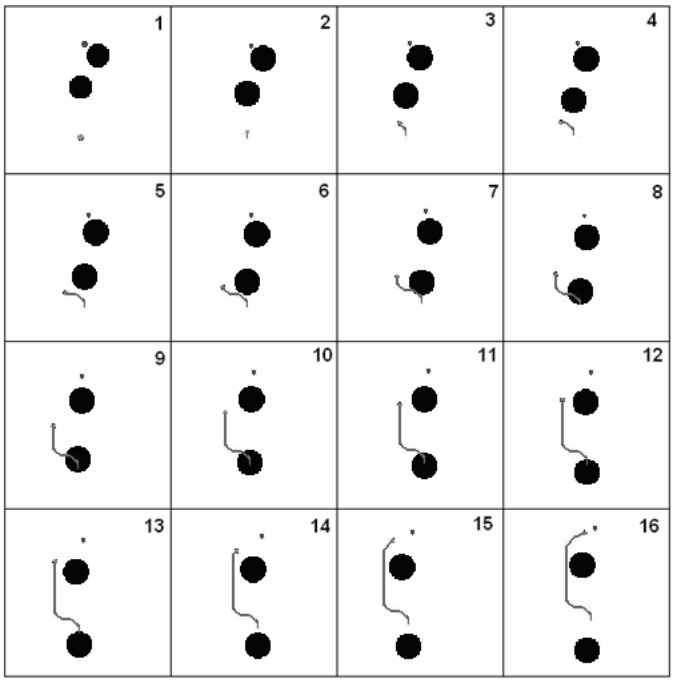

Fig. 18. Case II, A algorithm, dynamic obstacles.*

The results obtained in each of the cases are summarized in Table 4. The important value in this table is the mean square error (MSE), this value represent the tracking error in each of the cases, and demonstrate in numbers the feasibility of the generated paths.

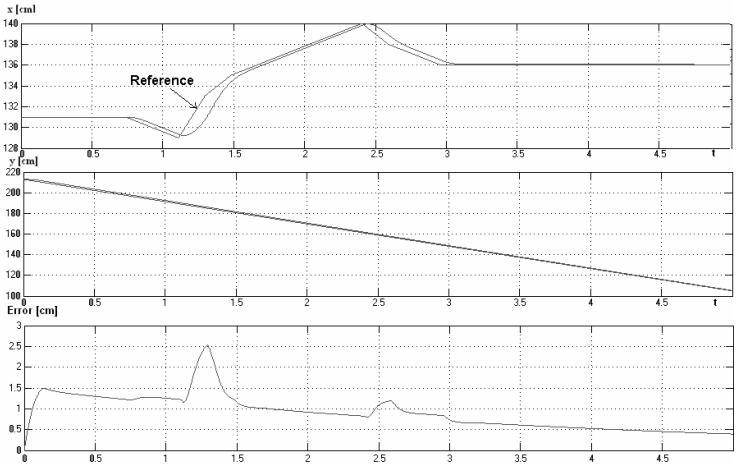

Fig. 19. Case I. Bug Algorithm with dynamic obstacles. The first graph shows the X axis movement, the second graph shows the Y axis movement and the third graph shows the tracking error as the distance between the followed and desired trajectories.

TABLE 4
COMPARISON OF THE METHODS EXPRESSED, WHERE MSE IS THE MEAN SQUARE ERROR OF THE TRACKING

	Case I		Case II	
	Distance (cm)	MSE	Distance (cm)	MSE
Bug Algorithm	140	1,0038	135	2,1713
Potential Fields	145	1,4515	150	1,4054
A* Algorithm	165	4,8957	175	2,7627

V. BUG ALGORITHM IN C++

Since the Modified Bug Algorithm yielded the best results in the simulations, it was selected for implementation in C++ in order to test its real-time performance as part of the artificial intelligence system of our RoboCup's F-180 team. The dynamical model and actuator control laws employed are based on [8] and [11]. Figures 20 and 21 show two different sets of trajectories generated using the Modified Bug Algorithm. The results are encouraging since there are no collisions no matter the complexity of the motion of the other robots and their number. The processing time of the Modified Bug Algorithm for the five-robots scenario is less than 4 ms, and less than 1 ms for two robot in average.

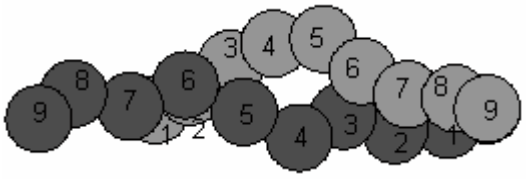

Fig. 20. Two-robot path planning and collision avoidance simulation using the Modified Bug Algorithm.

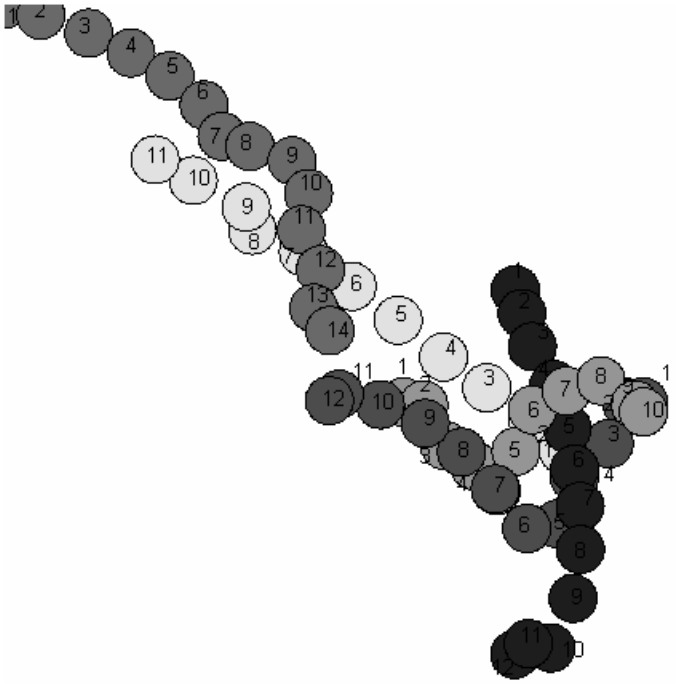

Fig. 21. Five-robot path planning and collision avoidance simulation using the Modified Bug Algorithm.

VI. CONCLUSIONS

The results show that the Modified Bug Algorithm is a suitable method for dynamic path planning applications and should be a good choice for many applications requiring fast path plan updates, such as the robotic soccer competitions. The A* algorithm yield the worst performance, due to the large computational effort it demands. In principle, the A* should result in optimal paths, but this is relative to the choice of the cost function *G* and in particular of the heurist cost function *H*. The main advantages and disadvantages of the different methods evaluated are summarized below in table 5.

TABLE 5
COMPARISON OF THE METHODS

Method	Modified Bug Algorithm	Potencial Fields Method	A* with Multiresolution Grids
Advantages	▪ Speed of execution. ▪ Dynmically gives an optimal path to the goal ▪ Easy implementation, parameters easy to adjust.	▪ Speed of execution	▪ Optimal Path (subject to adequate choice of cost functions). ▪ Reduced collision probabilities.
Disadvantages	▪ Binary method. Forced to increase the radius of the obstacles to avoid collisions.	▪ Local minima. ▪ Potential field forces must be set	▪ Slow processing. ▪ Costs have to be adjusted

Future works considers improving the modified algorithm to calculate multiple tangents to multiple obstacles and exploring ways of combining pursuit and evasion techniques that could result in more efficient obstacle avoiding trajectories leading to the goal in optimal time.

It is also considered to add prediction to the algorithms, taking in account the velocities of the obstacles. This would lead to better paths.

Finally, an important work that wasn't discussed on this paper is an analysis of the collision risks of the different path planning methods.

REFERENCES

[1] R. Siegwart and I.R. Nourbakhsh, *Introduction to Autonomous Mobile Robots*, MIT Press, 2004.

[2] S.M. Lavalle, *Planning Algorithms,*Cambridge University Press, 2004.

[3] S. Rajko and S.M. Lavalle, "A pursuit-evasion bug algorithm," In *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 1954-1960, 2001.

[4] N.D. Rodstein and A.J. García, "Evasión de obstáculos con bajo costo computacional para un equipo de fútbol de robots," *X Congress of Computer Science* (CACiC 2004), 2004.

[5] E. Rimon and D.E. Koditschek, "Exact robot navigation using artificial potential fields," *IEEE Transactions on Robotics & Automation,* v. 8, n. 5, pp. 501-518, October 1992.

[6] N.J.Nilsson, *Problem-Solving Methods in Artificial Intelligence*, McGraw Hill, 1971.

[7] S.Behnke, "Local Multiresolution Path Planning," *7th International Workshop on Robocup,* 2003.

[8] Y Liu, X. Wu, J.J. Zim and J Lew, "Omni-directional mobile robot controller design by trajectory linearization," *Proceedings of the American Control Conference 2003,* Vol. 4 Nº. 4-6, Jun. 2003, pp. 3423-3428.

[9] J Wu, *Dynamic Path Planning of an Omnidirectional Robot in a Dynamic Environment*, Ph.D. Thesis, Ohio University, March 2005.

[10] N.J. Nilsson, *Artificial Intelligence: A New Synthesis*, Morgan Kafmann, 1998.

[11] J.L. Peralta, M. Guarini and M. Torres, "A Comparison of Bayesian Prediction Techniques for Mobile Robot Trajectory Tracking". In *Prodeedings of the 3rd IEEE International Conference on Mechatronics*, Budapest, Hungary, 2006.