Path Planning for Mobile Robots using Iterative Artificial Potential Field Method

Hossein Adeli[1], M.H.N. Tabrizi[2], Alborz Mazloomian[3], Ehsan Hajipour[3] and Mehran Jahed[3]

**[1] Computer Science Department, East Carolina University
Greenville, NC, USA**

**[2] Computer Science Department, East Carolina University
Greenville, NC, USA**

**[3] Electrical Engineering Department, Sharif University of Tech
Tehran, Tehran, Iran**

Abstract

In this paper, a new algorithm is proposed for solving the path planning problem of mobile robots. The algorithm is based on Artificial Potential Field (APF) methods that have been widely used for path planning related problems for more than two decades. While keeping the simplicity of traditional APF methods, our algorithm is built upon new potential functions based on the distances from obstacles, destination point and start point. The algorithm uses the potential field values iteratively to find the optimum points in the workspace in order to form the path from start to destination. The number of iterations depends on the size and shape of the workspace. The performance of the proposed algorithm is tested by conducting simulation experiments.

Keywords: *Mobile robot, Path Planning, Artificial Potential Field, Collision Free Path*.

1. Introduction

Mobile Robot path planning is one of the important problems in the field of robotics. Its aim is to find a collision free path, where a robot may follow to reach its destination (goal) from its current position [1]. This kind of problems also exists in applications such as manufacturing, transportation and mobile systems [2].

In each of these applications we want to find a path that satisfies the criteria of optimality. For example, in one application a path with minimum length and in another, a smooth path with maximum distance from obstacles [1] may be required. Therefore the path planning method should be flexible enough to allow the users to plan the suitable path for their application.

Because of their mathematical simplicity and straightforwardness, Potential Field Method (PFM) is one of the mostly studied and used methods in mobile robot path planning. The basic idea is that a negative weight is assigned to the destination and positive weight to obstacles. Then the robot descends down the potential field using gradient descent method to reach its destination while avoiding obstacles [3].

Although these methods are fast and efficient, they have the following drawbacks and limitations as discussed in [4]:

I. Trap situations due to local minima.
II. No passage between closely spaced obstacles.
III. Oscillations in the presence of obstacles.
IV. Oscillations in narrow passages.

To overcome these limitations, several authors [5-10] have tried to solve the local minima problem by presenting new potential functions so that the destination becomes the global minimum. Others have tried to solve these problems by combining the simple potential methods with artificial intelligence models like neural network [1], genetic algorithm [7][2] and fuzzy logic [11]. But unfortunately these methods contribute to increase in the complexity of the algorithms.

Our method involves using a simple potential functions; the workspace is discretized into a grid of rectangular cells where each cell is marked as an obstacle or a non-obstacle. We evaluate the potential functions for each cell based on its distances from the destination, start and obstacles. These values are used to find the optimum points along the entire path. We find these points iteratively until there are enough points that path can be determined as a consecutive sequence of these points beginning from the start location and ending at the destination. Simulations experiments verify that this algorithm is not bound to the limits as is the case with traditional APF methods.

This paper is organized as follows. In section 2, new APF based method is explained. Section 3 examines the performance of this method through simulations, and

finally section 4 discusses the conclusions and further work.

2. Iterative APF Algorithm

2.1 The Method

First, we assume that the workspace is two-dimensional space as shown in Fig 1; the workspace includes start and end points and obstacles. We discretize the workspace to 50*50 cells and assume that the robot will fit in one cell and each cell is either empty or occupied. We present each cell by its coordinates $c = [x, y]$.

Algorithm begins calculating the potential function for every empty cell in the workspace.

$$U_{Total}(c) = U_{Start}(c) + U_{End}(c) - U_{Obs}(c) \quad (1)$$

It is important to note that the distance of the cell from the start point is being used in (1). The individual functions are expressed as

$$U_{Start}(c) = \frac{\alpha}{D(c,Start)} \quad (2)$$

$$U_{End}(c) = \frac{\alpha}{D(c,End)} \quad (3)$$

$$U_{Obs}(c) = \frac{\beta}{D(c,Obs)} \quad (4)$$

Where $D(c, Start)$ is the distance of cell c form the start, $D(c, Start)$ is the distance of cell c from the end point and $D(c, Obs)$ is the distance of cell c from the closest obstacle.

We are using somewhat different approach than traditional PFM; as these functions imply, there is no difference between start and destination positions. The positive constants α, β (as will be discussed later in section 2.3) are used to change the behavior of the generated path. For now we assume they are both equal to 1.

Using the proposed potential functions (1), we evaluate the potential value for each empty cell in Fig 1(a). Then we sort the cells in descending order based on the value assigned to each in the workspace. Considering proposed functions 2-4, we expect the cells around the start and end to have the highest values and cells near an obstacle to have lower values. We have marked all the cells with the values that are in top 50 % of the sorted list in Fig 1(b) and marked all the cells with the values that are in top 60 % of the sorted list in Fig 1(c).

We now define the notion of threshold to use it in the rest of the paper. Setting a threshold means we first pick a value from the sorted list of all values and then we mark every cell with the value more than this threshold in the workspace. For example when we say that threshold value is equal to x, it means that every cell is marked with the value more than x in our workspace.

So if we set the threshold value to be large, we get two distinct clusters of marked cells around start and destination points. Then if we gradually decrease the threshold, these two clusters get bigger and bigger until they run into each other as shown in Fig 1(c).

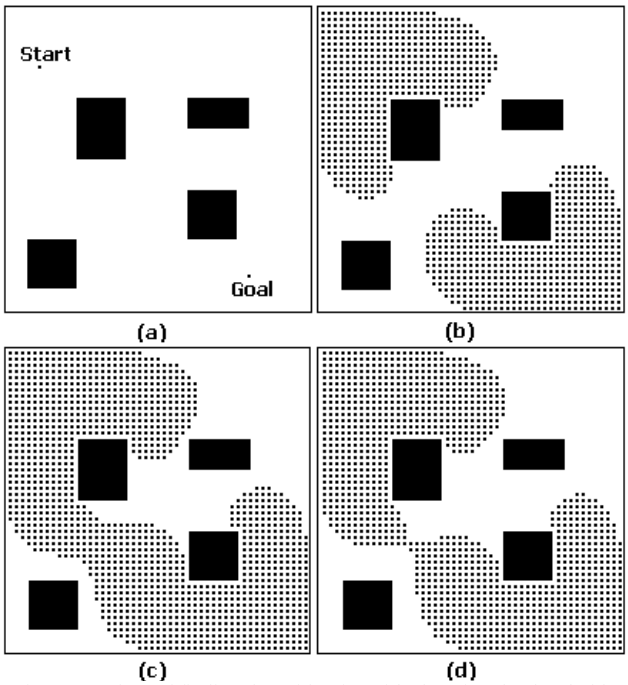

Fig 1 Procedure of finding the Mid-point with changing the threshold value (T). (a) The workspace with start and destination points. (b) The clusters when $T>T_0$. (c) clusters when $T<T_0$. (d) finding Mid-point when $T = T_0$.

What we are looking for is to find the threshold value that guarantees that there is one and only one cell that connects the start point cluster to the end point cluster, see Fig 1(d). Values smaller than this threshold value would make a connection between two clusters but this threshold value is the biggest value that by using all cells that their value is bigger than this value, makes it possible for having a path between start and end points. We believe this point should appear in the path because it optimizes the potential functions while making sure that path exists. The point that connects these clusters, we call it **midpoint** and we are interested in this point mainly because:

a) There is a path from the start point to the midpoint, and there is a path from midpoint to the destination.
b) We know we should be looking for the path within the start and destination point clusters which are formed on either side of midpoint.

But how do we find it? For every threshold value some cells in the workspace are marked and we can check for the existence of path from start to end among marked cells. We do this by using a simple Breadth-First Search (BFS) algorithm. So in order to find the threshold that holds the

midpoint criterion (only one point common between two clusters), we apply a binary search on the sorted list of cells to find the minimum threshold with no connection between clusters. Then the value of the next cell in the list is the maximum threshold that connects two clusters, see Fig 2.

Fig 2. Pseudo code for finding the midpoint

```
Let N be the number of available cells, Evaluate all these cells.
A = Sorted array of all cell's values.
Binary_Search( 1, N , A );

Binary_Search( i, j ,A)
 If ( i == j )
  return A[ i + 1]
 T = A[ ( i + j ) / 2]
 If ( by using simple BFS, Is end point reachable from start
                point using cells with larger value than T ? )
  Binary_Search( i, (( i + j ) / 2 ) - 1 , A )
 Else
  Binary_Search( ( i + j ) / 2, j, A )
```

Notice that we have assumed the values are all distinct but sometime two or more cells might have the same value, in that case we just go through all the cells in the list and change the values to make them different. For example if we have two cells with the value y and the next cell's value is x then we keep one of those cells with value y and change the other to $(y + x)/2$.

So far we know:
The path from the start to the midpoint is in the cells that are marked as the start point cluster and the path from the midpoint to the end is in the cells that are marked as the endpoint cluster. so we need to keep track of what cells have been marked as the start point or end point clusters so for every cell besides keeping the potential value we need to keep one bit for showing if it is marked in the start point cluster and another bit for showing if it is marked in the end point cluster.
Our next step is to find the midpoint between the start and the current midpoint knowing that we should be able to find it in the start point cluster of previous procedure of finding the midpoint. In other words we want to find the midpoint between two points assuming that only some cells of the workspace are available. In order to do that we need to assign another bit for each cell that shows the availability of that cell, see Fig 3.

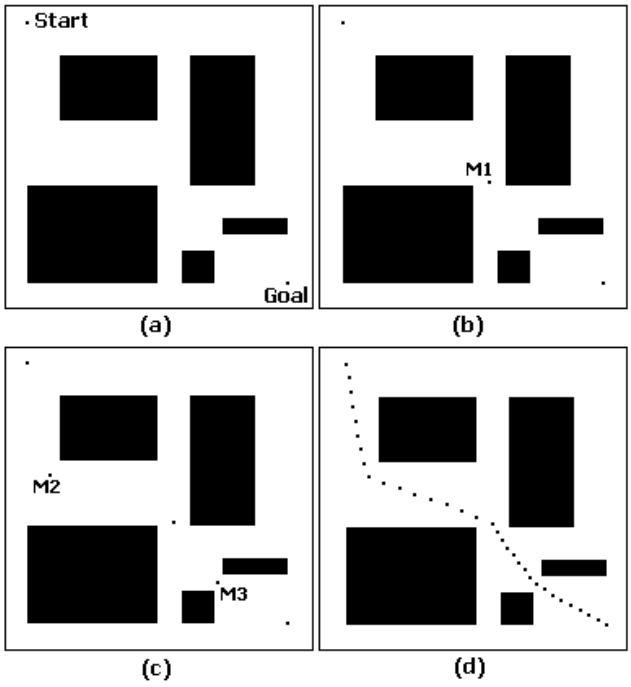

Fig 3. The procedure of finding the path. (a) Workspace with start and destination points. (b) First Mid-point (M_1) is found. (c) M_2 and M_3 are found by running the algorithm on (start, M_1) and (M_1, destination) respectively (d) Final path (M_n = 30)

In Fig 3(c) M2 is the midpoint between start and previous midpoint. We do the same to find the midpoint M3 between previous midpoint and the end point.

We keep executing this algorithm to find midpoints until there is a co llision free path between every two consecutive midpoints and that would be our final path as shown in Fig 3(d). Fig 4 shows the pseudo code for the algorithm.

The number of midpoints, needed for the path, depends on the complexity of the workspace. For a simple workspace like Fig 3, a path can be found with only three midpoints. But usually we let the algorithm to find certain number of midpoints to generate a smoother path. The path shown in Fig 3(d) is created using 30 midpoints.

2.2 Time and Space Complexity

Let N be the number of cells in the workspace. It takes $O(N)$ time to evaluate all the cells. To find the midpoint of a workspace we need to sort all values of the cells. This can be done in $O(N)$ memory and $O(NLgN)$ time.
Then, for different threshold values we check whether the end point is reachable from the start point or not, using BFS algorithm from the start point and by ignoring the cells with value less than the threshold value. Since each cell in the workspace has four neighbors, BFS takes $O(N)$memory and $O(N)$time. Moreover, we search for the

largest of such thresholds using Binary-Search algorithm, so the whole procedure of finding a midpoint for a workspace can be implemented in $O(NLgN)$ time. On the other hand, the maximum number of calling Find_MidPoint function is $O(N)$ as at each call, the midpoint and potentially some other cells are excluded from the existing workspaces. Searching for Midpoints can be performed separately for different workspaces and so the complete algorithm can be implemented in $O(N)$memory and $O(N^2LgN)$time, which is quite fast and efficient.

Fig 4. Pseudo code for finding the path

```
Inputs = Start, Destination, Workspace
Output = Collision free path

Function Find_Path (Start, End, Workspace)
  If < Endpoints are close enough and there is a  collision free
        straight line connecting them >
     Return Segment( Start, End );
  Else
   [ MidPoint, First_Workspace, Second_Workspace]  =
                 Find_MidPoint( Start, End, Workspace )
   First_Path = Find_Path ( Start, MidPoint, First_Workspace )
   Second_Path=FindPath( MidPoint, End, Second_Workspace )
   Return  Merge( First_Path, Second_Path )

Function Find_MidPoint( Start, Destination, Workspace )

    Evaluate all the cells in the Workspace
    A = Sorted array of all available cells in Workspace in
        descending order
    N = Length of A
    T = Binary_Search( 1, N , A )
    Midpoint = Cell having the value T
    First_Workspace = Start point cluster
    Second_Workspace = End point cluster
    Return ( Midpoint, First_Workspace, Second_Workspace )
```

2.3 Adjusting α and β

It can be observed from potential functions that by increasing α, we put more emphasis on the distances from start and end points so that having a large value for α produces a shorter path as shown in Fig 5(b) but the path might be close to obstacles. By increasing β we put more emphasis on the distance from obstacles and it means that selecting a large value for β gives us a longer path with bigger distance from the obstacles. Fig 5(a) shows such a path.

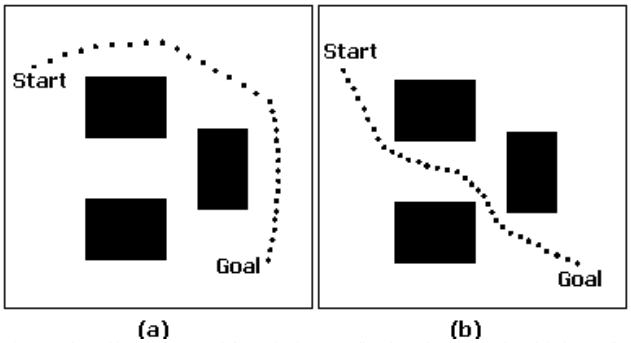

Fig 5. The effect of α and β variation on final path. (a) Path with large β (b) Path with large α .

3. Simulation Experiments

We have tested the performance of our algorithm using several different workspaces. Two of these tests are presented in this paper. We have tried to cover all the possible cases that give rise to limitations of traditional APF methods. Both of these tests have points that can be local minima; they also have closely spaced obstacles, and narrow passages as shown in Fig 6 and Fig 7. In both workspaces the path is found very quickly with less using 40 midpoints.

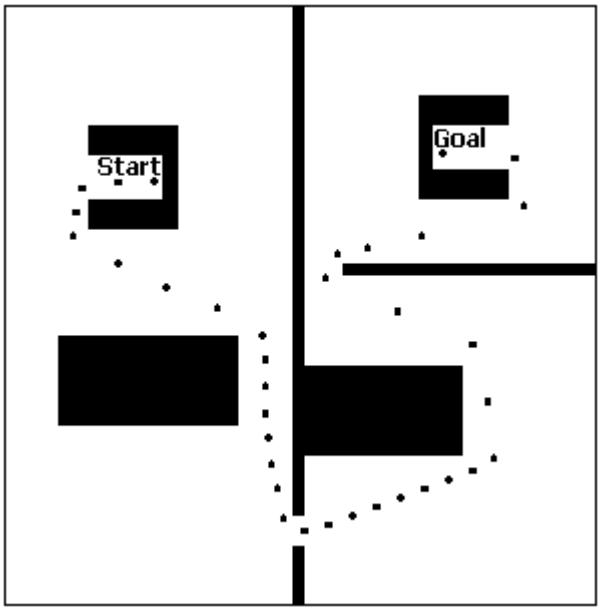

Fig 6. Path planning in a workspace with local minima, closely spaced obstacles and narrow passages

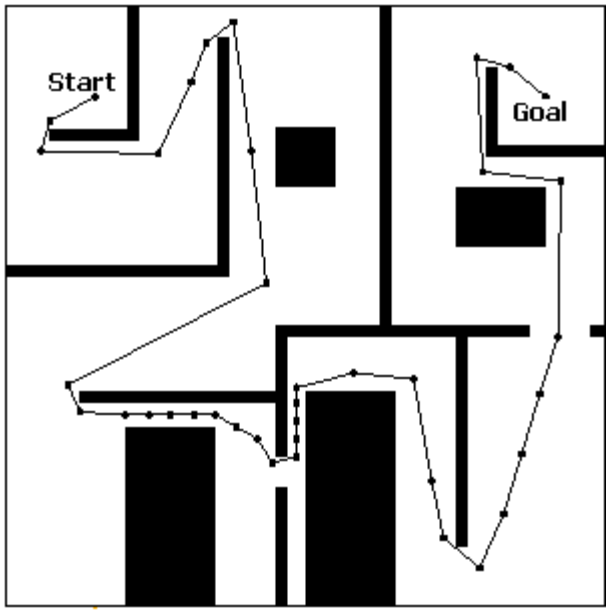

Fig 7. Path planning in a workspace with local minima, closely spaced obstacles and narrow passages

4. Conclusion

In this paper a new method called Iterative Potential Field method is presented for efficient path planning of mobile robots. Using this method, a smooth path with reasonable distance form obstacles is identified while keeping the path as short as possible. Using the values of proposed potential functions the algorithm iteratively finds the optimum points to create the path. We called these points, midpoints and algorithm finds as many of them as needed in order to generate a path from start to destination. Simulations results show the successfully generated paths for two workspaces. For future works, this method can be implemented for real time navigation or in the workspace with moving obstacles.

References

[1] N. Sadati, J. Taheri, "Hopfield Neural Network in Solving the Robot Motion Planning Problem", IASTED Intl. Conf. on Applied Informatics (AI2002), Innsbruck, Austria, Feb 2002.

[2] H. Mahjoubi, F. Bahrami, and C. Lucas, "Path Planning in an environment with static and dynamic obstacles using Genetic Algorithm: A Simplified Search Space Approach," IEEE Congress on Evolutionary, pp. 2483-2489, 2006.

[3] O. Khatib, "Real-time Obstacle Avoidance for Manipulators and Mobile Robots," Intl. J. of Robotics Research, Vol. 5, No. 1, pp 90-98,1986.

[4] Y. Koren and J. Borenstein, "Potential Field Methods and Their Inherent Limitations for Mobile Robot Navigation" Proceedings of the IEEE Conference on R obotics and Automation, pp. 1398-1404, Sacramento, California,April 7-12, 1991.

[5] S.S. Ge and Y.J. Cui,"New potential functions for mobile robot path planning," IEEE Trans. on R obotics and Automation, pp. 615-620,2000.

[6] J. Agirrebeitia, R. Avilés, I.F. de Bustos and G. Ajuria, "A new APF strategy for path planning in environments with obstacles", Mechanism and Machine Theory (2005), pp. 645–658

[7] Cosio A. F., Castaneda P. A. "Autonomous robot navigation using adaptive potential fields", Mathematical and Computer Modelling. – 2004. – Volume 40. – P. 1141–1156.

[8] Barraquand, J., Langlois, B., and Latombe, J. C.. " Numerical potential field techniques for robot path planning".IEEE Transactions on Systems, Man, and Cybernetics 22:224–241 1992

[9] Benamati, L.; Cosma, C.; Fiorini, P.; , "Path planning using flat potential field approach," Advanced Robotics, 2005. ICAR '05. Proceedings, 12th International Conference on , vol., no., pp.103-108, 18-20 July 2005

[10] Cheol-Taek Kim; Ju-Jang Lee; , "Mobile robot navigation using multi-resolution electrostatic potential field," Industrial Electronics Society, 2005. IECON 2005. 31st Annual Conference of IEEE, vol., no., pp. 5 pp., 6-10 Nov. 2005

[11] Meng, Rui, Su, Wei-Jun, Lian, Xiao-Feng. "Mobile robot path planning based on dynamic fuzzy artificial potential field method" Computer Engineering and Design, Vol. 31, no. 7, pp. 1558-1561. 16 Apr 2010

Hossein Adeli received his B.Sc. degree in Electrical Engineering from Sharif University of Technology, Iran, 2009. He is currently a computer science Master's student at East Carolina University, USA. His research interests are in the areas of Robotics, Artificial Intelligence and Machine Learning.

M.H.N Tabrizi received his B.S. degree in Computer Science from Manchester University, UK. He then completed his M.S. and Ph.D. from Automatic Control and Systems Engineering Department, Sheffield University, UK. He worked in Manchester University for two years prior to his appointment at East Carolina University in 1984. He is the Graduate Program Director of Computer Science and founder and director of Software Engineering program at East Carolina University. His research interests are in the areas of Cloud Computing, Virtual Reality, Modeling and Simulation, Computer Vision, Signal and Image Processing, Software Engineering, and Computer Science Education. His publications include diverse areas of research in computer science, technology, and software engineering. He was named ECU's scholar teacher in 2000 and has received best paper award.