An Instant Path Planning Algorithm for Indoor Mobile Robots Using Adaptive Dynamic Programming and Reinforcement Learning

R.Karthikeyan[1], B.SheelaRani[2], K.Renganathan[3]
[1]Research Scholar, Sathyabama University,
Chennai, Tamilnadu, India
[2]Centre for Research, Sathyabama University,
Chennai, Tamilnadu, India
[3]Department of Electronics & Instrumentation, Sri Sairam Engineering College
Chennai, Tamilnadu, India
[1]karthikeyan.ice@sairam.edu.in, [2]kavi_sheela@yahoo.com,[3]renganathan.ice@sairam.edu.in

Abstract- An Adaptive Dynamic Programming and Reinforcement Learning (ADPRL) based instant path planning algorithm is proposed in this paper. The layout of any indoor environment is always known. This information is converted into a binary matrix containing free space and obstacle space using image processing system. A dynamic program algorithm translates the rough obstacles to expected shaped obstacles so the robot is not confined in motion. A grid policy is used for value evaluation of reward function. Value iteration draws out all possible paths from goal to target. A Q-learning algorithm finds the best possible path from the numerous possible paths determined. A Biezer curve based approximation is done to smoothen the discrete way points for smooth motion and determination of linear and angular velocities for a differential drive robot. The simulation and the results show the proposed algorithm have better processing time, less computational complexity, and instant determination of path, compared to other existing methods.

Keywords: Dynamic Programming, Grid Mapping, Q-learning, Reinforcement Learning.

I. INTRODUCTION

With the enormous growth of mobile robots for Indoor applications in instances such as museums, Industrial material handling, material transfer, domestic assistance and robotic competitions, use of a single robot to be compatible and adaptable to all situations requires complex path planning and repeated programming. This paper proposes a adaptive algorithm which gives best solution for many criteria such as time delay, travelling distance, computational complexity, planned time of arrival, instant start and target locations changeability etc. The algorithm aims in path planning of a mobile robot from an initial position and orientation to a goal position and orientation without obstacle collision in a finite time. The typical way to solve this problem is splitting it in less complex sub problems.[1,2]. The problem is divided into mapping, Value evaluation, Value iteration, Optimal way point generation, curve connection, and navigation. Some methods require the workspace to be two-dimensional and the object shape to be defined. The most common methods are based on road-map, cell decomposition and potential fields[3]. another problem of these approaches is that most of them produce polygonal line paths, and this geometric paths are not good to non-holonomic robot navigation.

Few adaptation techniques were developed to make these paths executable by robots with non-holonomic constraints [4].In the other hand, Reinforcement Learning [5]. Is, learning what to do so as to maximize a reward signal. The learner is not told which actions to take, but instead must discover which actions yield the highest reward by trying them.

II. MAPPING

The layout of the environment to be path planned is given to the controller as an image data. This image data is converted into a binary image of 0's and 1's by using Thresholding technique. The obstacles which are converted to black are assigned 0's and the free space is assigned 1's. The image is partitioned as a grid of 512 by 512 sizes [6]. Were the number of rows & columns is assumed equal as 512. A proposed environment [10]. as shown in fig. 1 is taken for test. The binary data is termed matrix 'O' as shown in fig. 2, which has the entire environment data of the free space and the obstacle.

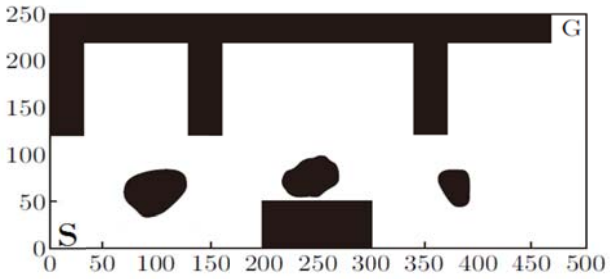


Fig. 1. The proposed test environment with obstacles, the start & goal is indicated as S & G.

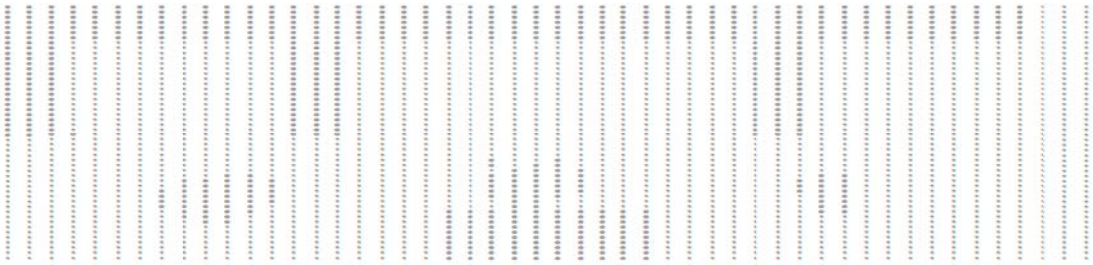


Fig.2. Binary data representation of the test environment the obstacle in black are termed as '0' and the free space in white are termed as '1'.

A. Assumptions Used In Model

The environment is decomposed into 512 by 512 grids of square cells. Each cell in the grid contains node which knows the location based on Integers as shown in fig. 3. The Start of Robot is always at the cell 1 and the goal cell is determined with a numeric node. Obstacles are static and size of the obstacle is of regular shape similar to the size of the cell. Only 5 movements are possible to navigate the entire environment. They are i) Diagonal ii) Right iii) Left iv) Up v) Down as shown in fig. 4.

7	14	21	28	35	42	49
6	13	20	27	34	41	48
5	12	19	26	33	40	47
4	11	18	25	32	39	46
3	10	17	24	31	38	45
2	9	16	23	30	37	44
1	8	15	22	29	36	43

Fig. 3. Proposed environment for test is split into equal cells of NXN matrix represented by numerical data as assigned above. Cell 1 is always the start node.

III. DYNAMIC PROGRAMMING

Dynamic programming is an optimization approach that transforms a complex problem into a sequence of simpler problems [7]. Usually imagination is required before we can identify that a particular hitch can be shed effectively as a dynamic program; and often delicate approaching are necessary to reorganize the formulation so that it can be solved effectively. The rawest approach to solving the problem would be to enumerate. Let us assume the current position of the robot is at state 'S' and the possible actions are shown in fig. 5. N is the number of rows and column. As we have assumed a 7BY7 grid map N is taken as 7

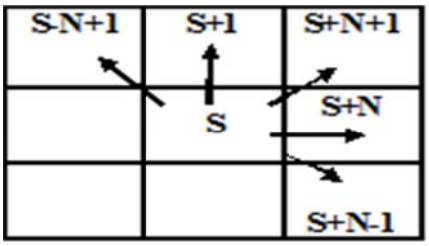
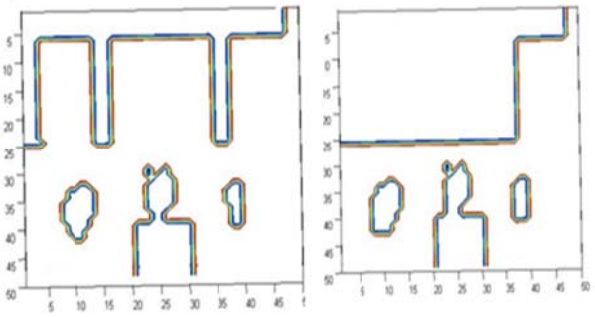


Fig. 5. S is denoted by the current position of the robot corresponding to the numerical value, then an diagonal action reaches the next state as S+N+1. All the action and possible next state are indicated.

There is a possible case when all the next possible actions contain obstacles. This when encountered after a search process will end up in a trap where there is no solution and the robot is trapped. A dynamic programming algorithm is proposed to avoid this trap situation. In DP the optimal-value function O_S^{π} of possible Obstacles over the current and subsequent states possible against each action, is denoted by,

$$O_S^{\pi}=O_S\,[(OR\,(O_{S+1}, O_{S+N}, O_{S+N+1})] \qquad (1)$$

Where '*O*' is the obstacle matrix from binary image data. O_S^{π} is given policy. Each iterations for all state S finally yields an environment as shown in fig. 6 shows the contour were the portions of traps found by the DP algorithm. After nearly 500 iterations the states containing possible trap actions are found and they are termed as obstacles and the new matrix 'O' is constructed.



(a) (b)

Fig. 6. (a) Contour of environment before Dynamic programming algorithm. (b) Contour of environment after Dynamic programming algorithm which eliminates the possible trap points.

IV. REINFORCEMENT LEARNING

Reinforcement learning is learning what to do, how to map situations to actions, so as to maximize a numerical reward signal [8]. The learner is not told which actions to take, as in most forms of machine learning, but instead must discover which actions yield the most reward by trying them. In the most cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. The basic idea behind reinforcement is that the learning system shown in Fig. 7 can learn how to solve a complex task through repeated interaction with the environment.

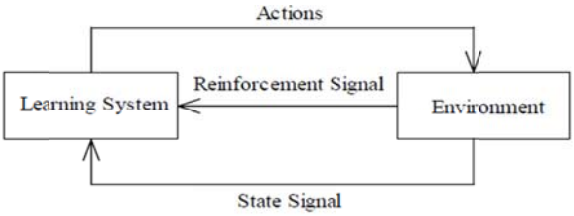


Fig. 7. From current state to next state a reward is assigned based on the possible action determined by the learning system.

A Value Evaluation

The value evaluation is done to construct a reward matrix 'R' which is a N by N matrix. The policy V^{π} is formed using the integer value assigned to each cell. For example if the goal state is 49 and as always the start cell is 1, the policy is determined using

V_1^{π} = [MOD ((S-G), N+1)] ==0; (2)

V_2^{π} = [MOD ((S-G), N)] ==0; (3)

The possible 5 actions from state S to state S' are denoted a0, a1, a2, a3, a4, a5, then from current state S the next state-action is

The reward evaluation algorithms

```
if V1π ==0;
{
        a0=S+N+1;a1=S+N;a2=S+1;
        a3=S+N-1;a4=S-N+1;
}
Else V2π ==0;
        {
                a0=S+N;a1=S+N+1;a2=S+1;
                a3=S+N-1;a4=S-N+1;
        }
        Else
        {
                a0=S+1;a1=S+N+1;a2=S+N;
                a3=S+N-1;a4=S-N+1;
        }
End
```

and the reward from state S to state S' are

R(s, a0) =1,

R(s, a1) =0.8,

R(s, a2) =0.6,

R(s, a3) =0.4,

R(s, a4) =0.2,

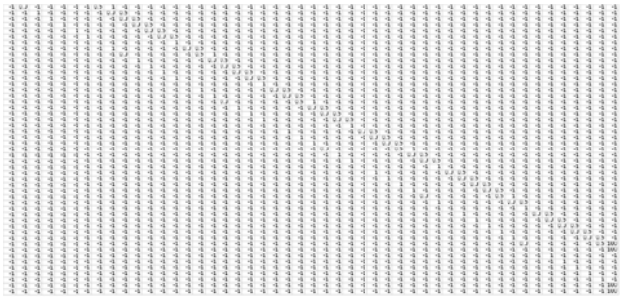


Fig. 8. Reward matrix 'R' generated by the value evaluation algorithm. A sample reward matrix for N as 7 is shown. The Reward matrix is of size N^2XN^2.

The figure 8 shows the reward matrix which is a 49 by 49 matrix gives the reward value from current state S to all possible states. The value is -1, if there is no possible action to other state and the value is 100 if the action yields a goal state

B. Value Iteration

The value iteration is done using Q-Learning algorithm[8]. It is a reinforcement learning algorithm that attempts to learn a the state-action value Q (s,a),whose value is the maximum discounted reward that can be achieved by starting in state S, taking an action a , and following the optimal policy thereafter. The matrix 'Q' which is a N by N matrix is initially initialized to zero. The algorithm is then iterated till the reward converges to a error less than 0.0001. We Iteratively approximate the optimal Q-Function based on our observation of the world.

The Q-Learning algorithm goes as follows:

Initialize the alpha and gamma parameters,

Initialize the environment rewards in matrix R based on the policy V^{π}.

Initialize matrix Q to zero.

For each episode until the error =0.0001

Select a random initial state.

For all state until goal state

Select possible actions a0 to a4 for the current state.

Using this possible action, consider going to the next state.

Get maximum Q value for this next state based on all possible actions.

Compute:

$$Q(s_{t,}a_{t,}) \leftarrow Q(s_{t,}a_{t,}) + \alpha \times R(s_{t,}a_{t,}) + \gamma \times \left\{ \max_{a} Q(s_{t+1,}a_{t,}) - Q(s_{t,}a_{t,}) \right\}$$

Set the next state as the current state.

End for

End episode

C. Optimal Way point generation

The matrix 'Q' contains the optimal value from one state to the other state. Table 1 illustrates the Q matrix generated from a grid of 4 by 4 matrix. After execution of the Q-learning algorithm a matrix O' is constructed with reference to O matrix this matrix is a N^2 by N^2 matrix similar to R and Q matrix. Each column of the O' matrix consists the information of obstacle starting from 1 to N^2, of the 'O' matrix. The matrix 'Q' is then multiplied with O' matrix such that the obstacle states are eliminated from the 'Q' matrix. The optimal policy Is evaluated from the following equation

$$\pi^*(s) = \arg\max_{a} Q(s,a).$$

(4)

Starting from the state 1 the state with maximum value is selected from the table it is found state 6 yield the maximum value and this state is taken as the next state and the optimal policy is evaluated till the goal state is reached.

TABLE I
Q MATRIX GENERATED BY Q-LEARNING ALGORITHM FOR N=4.

Next State / Current State	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	0	81.15	0	0	81.152	81.19	0	0	0	0	0	0	0	0	0	0
2	0	0	81.132	0	0	81.19	81.17	0	0	0	0	0	0	0	0	0
3	0	0	0	81.19	0	0	81.15	81.17	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	90.1	0	0	0	0	0	0	0	0
5	0	90.04	0	0	0	81.19	0	0	81.15	81.17	0	0	0	0	0	0
6	0	0	81.092	0	0	0	81.15	0	81.112	81.17	90.1	0	0	0	0	0
7	0	0	0	81.11	0	0	0	81.15	0	81.13	90.1	90.08	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	90.1	0	0	0	0
9	0	0	0	0	0	81.13	0	0	0	81.19	0	0	81.132	81.17	0	0
10	0	0	0	0	0	0	81.11	0	0	0	90.1	0	90.04	90.06	90.08	0
11	0	0	0	0	0	0	0	90.02	0	81.15	0	0	0	81.13	90.08	100
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100
13	0	0	0	0	0	0	0	0	0	81.17	0	0	0	81.19	0	0
14	0	0	0	0	0	0	0	0	0	0	90.08	0	0	0	90.1	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100

D. B'ezier Curve Fitting

The way points generated by the Q-Learning algorithm are not smooth and motion of mobile robot implemented through such discrete translation and rotation are discontinuous, time consuming and waste of robot power. In order to overcome these disadvantages, many geometric curves based smooth path planning techniques like B´ezier curve [9], have been proposed. The B´ezier curve passes through the start and final control points but not the intermediate ones, which define the start and the final orientation and the shape of the curve.

Consider six control points, P0, P1, P2 P3, P4 and P5, they uniquely define the fifth-order B´ezier curve. The fifth-order Bernstein polynomials,

$$B_i^5(\lambda)(\ i = 0, 1, 2, 3, 4, 5\ ) \qquad (5)$$

and the control points P*i* (i = 0, 1, 2, 3, 4, 5) defined third-order B´ezier curve is expressed as

$$P(\lambda) = (1-\lambda)^5 P_0 + 5\lambda(1-\lambda)^4 P_1 + 10(\lambda)^2(1-\lambda)^3 P_2 + 10(\lambda)^3(1-\lambda)^2 P_3 + 5(\lambda)^4(1-\lambda)P_4 + (\lambda)^5 P_5 \qquad (6)$$

And the velocity vector v(λ) is defined as derivation of the path vector P(λ) with respect to the normalized time λ as

$$\dot{\mathbf{P}}(\lambda) = \frac{d\mathbf{P}(\lambda)}{d\lambda} = \sum_{i=0}^{n-1} n(\mathbf{P}_{i+1} - \mathbf{P}_i) B_i^{n-1}(\lambda). \qquad (7)$$

Consequently, the curvature $k(\lambda)$ at each point on the B´ezier curve could be calculated in terms of the first and second derivatives with respect to λ as

$$\kappa(\lambda) = \frac{\dot{P}_x(\lambda)\ddot{P}_y(\lambda) - \dot{P}_y(\lambda)\ddot{P}_x(\lambda)}{(\dot{P}_x^2(\lambda) + \dot{P}_y^2(\lambda))^{\frac{3}{2}}}, \qquad (8)$$

The linear velocity, X, Y coordinates and angular velocity are also calculated. The Velocity are given two a differential drive robot and the performance is measured. In this paper the simulation results are discussed

V. SIMULATION & RESULTS

The entire simulation is run using MATLAB software. The image data is read and converted to a standard partition of 512 by 512 grids. Matrix 'O', 'O' ", 'R', 'Q' are evolved using a ADPRL. The adaptive nature helps the robot avoid trap situation. The entire process of mapping is completed in 29.835739 seconds. This time is really a fastest solution is any path planning robot working in multiple environments. Fig. 9 shows the path generated by the algorithm avoiding obstacles. The squares are few way points and curve smoothening is done to reduce sharp turning. Fig. 10. is the plot when the dynamic programming is not used. The robot is trapped into a corner and the actions do not allow the robot to return to its previous position, so that another action can be taken. Fig. 11. Show the discrete path resulted from the algorithm before curve fitting is done. The robot simulation is tested with different environment and found the time taken to plan even in a worst case is 48.2683 seconds. The output velocities were given to a differential drive robot and found to have the same path motion. The lack of sensors in robot prevented in displaying results.

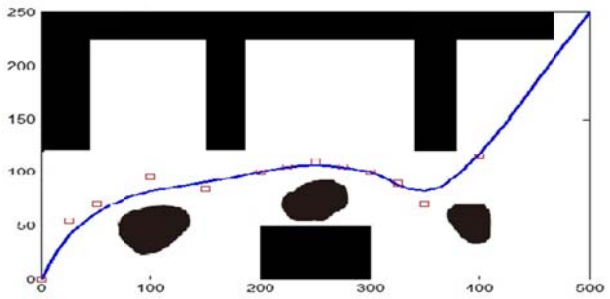


Fig. 9. Path generated by the ADPRL algorithm avoiding obstacles the squares are few way points generated but approximated by cure smoothening

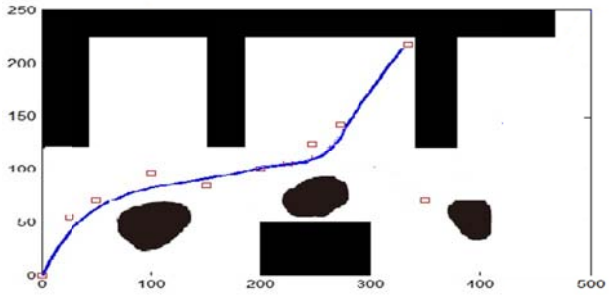


Fig. 10. A trap situation when the dynamic program is not used. The actions were not helpful to back and avoid.

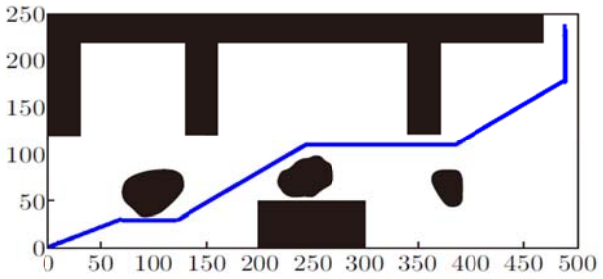


Fig. 11. Path generated without the cure fitting algorithm, The path was in discrete steps.

VI. CONCLUSION

In this paper a adaptive method of Dynamic programming and Reinforcement learning used to readily make a robot learn its environment on the basis of map depicted with static obstacle is studied. The robot is made to avoid the traps which are complex consumer of memory and time is implemented. Static obstacles with shapes assumed to be the size of a cell are only considered in this work. Simulation results show that in all cases path has been found successfully by avoiding collision with obstacles. Only one quadrant movement is only considered in this work. Future work is to implement multiple quadrants and dynamic obstacle of any shape is planned.

REFERENCES

[1] Pedrosa, D. P., Medeiros, A. A., and Alsina, P. J. (2003). “Um metodo ´ de gerac¸ao˜ de trajetoria ´ para robosˆ nao-holon ˜ omicos com acionamento diferencial. In Simposio ´ Brasileiro de Automac¸ao˜ Inteligente,” pages 840–845, Bauru, SP, Brazil.
[2] Latombe, J.-C. (1991). “Robot Motion Planning”. Kluwer Academic Publishers.
[3] Jung-Jun Parkm Ji-Hun Kim, and Jae Bok Song, “Path planning for a Robot Manipulator based on Probabilistic Roadmap and Reinforcement Learning, International Journal of Control, Automation and Systems”, Vol. 5, no. 6, pp. 674-680, Dec 2007.
[4] Alsina, P. J., Gonc¸alves, L. M., Vieira, F. C., Pedrosa, D. P., and Medeiros, A. A. (2002). Minicurso: Navegac¸ao˜ e controle de robosˆ mov´ eis. Congreso Brasileiro de Automatica ´ - CBA 2002.
[5] R.S. Sutton, A.G. Barto, “Reinforcement Learning: An Introduction” Bradford Books/MIT Press, Cambridge, MA, 1998.
[6] Danica Janlova “Neural Networks in Mobile Robot Motion” International Journal of Advanced Robotic Systems, Volume 1 Number 1, 2004, pp.15-22

[7] Ronald A. Howard , “Dynamic Programming and Markov Processes” John Wiley & Sons, 1960,
[8] C.J.C.H. Watkins, P. Dayan, “Q-learning, Machine Learning” (1992) 279–292
[9] Junwei Yu, Feng Wang ; Dexian Zhang ; Lubin Weng, “Vision heading navigation based on navigation curve”, Intelligent Computing and Integrated Systems (ICISS), 2010 International Conference Page(s):290 - 293Print ISBN:978-1-4244-6834
[10] Fengyu Zhou a, Baoye Song b, Guohui Tian , B_ezier Curve Based Smooth Path Planning for Mobile Robot Journal of Information & Computational Science 8: 12