Path Planning for Dynamic Environments

Aman Virmani
Maryland Applied Graduate Engineering - Robotics
A. James Clark School of Engineering
University of Maryland
College Park, USA
avirmani@umd.edu

Umang Rastogi
Maryland Applied Graduate Engineering - Robotics
A. James Clark School of Engineering
University of Maryland
College Park, USA
urastogi@umd.edu

Sayani Roy
Maryland Applied Graduate Engineering - Robotics
A. James Clark School of Engineering
University of Maryland
College Park, USA
sroy0108@umd.edu

***Abstract*—Dynamic programming approach to find the shortest path is a method that provides a global optimal solution to the robot path planning problem. It can be used in indoor environments. We use this approach published in 2009 in the paper, Dynamic Programming Agent for Mobile Robot Navigation with Moving Obstacles [1], to expand its applications. We implement the algorithm on a differential-drive robot to simulate its collision-free movement in a dynamic environment. In doing so, we challenge the underlying assumptions proposed for the algorithm.**

***Index Terms*—dynamic programming , path planning, dynamic environment, moving obstacles, differential-drive robots, robot operating system, gazebo**

I. Introduction

In various fields of application and research like computer games and virtual environments, CAD-design, molecular biology and robotics, path planning plays an important role. In most general form, the goal of path planning is to plan a path for some moving entity between a start position and a goal position in some environment. The motion planning problem for mobile robots can be typically formulated as given a robot and a description of an environment, plan a path of the robot between two specified locations, which is collision-free and satisfies certain optimization criteria [2]. Path planning in presence of dynamic obstacles is a challenging problem due to the added time dimension in the search space. The approaches which choose to ignore the time dimension and treat dynamic obstacles as static has to re-plan its path frequently. These solution paths are generally sub-optimal and are often incomplete. To achieve both optimality and completeness, it is necessary to consider the time dimension during planning.

Dynamic programming or DP is an algorithmic paradigm which can be used to solve any given complex problem. It breaks the problem into different sub-problems and stores the results of these problems to avoid repetitive computation of the same results. Dynamic Programming is heavily used in optimization problems and its applications range from financial modeling, operation research to biology and basic algorithm research.The DP path planning algorithm can be used for mobile robot navigation in both static and dynamic environments. The algorithm works in real time and does not require any prior knowledge of obstacle locations. It can also provide a global optimal solution using the shortest path algorithm to the robot in an indoor environment with both static and dynamic obstacles. It is a powerful technique that allows solving of problems in time O(n^2) or O(n^3) [3].

The method proposed by the paper, Dynamic programming agent for mobile robot navigation with moving obstacles [1], is for real-time and provides collision-free path planning, even for mobile obstacles. The methodology used in this paper [1] has been applied on a point robot to get optimal shortest path that the robot can traverse in a dynamic environment.

II. Dynamic Programming Approach

A. Algorithm

The dynamic programming algorithm involves the following steps.

a The environment space is discretized as a 2-D array of M grid points.

b d_{min} and d_{max} is defined as the minimum and maximum distance between any two neighbours ($d_{min} = 1$ and $d_{max} = 1.414$ respectively for our 8 neighbor configuration of point robot).

c 2 arrays named x and B are created.
- x_i denotes the distance of index i from the index of target point.
- B_i denotes the set of adjacent free neighbors for index i in the original grid.

d The distance array, X is initialized as:

$$x_i = \begin{cases} 0, & \text{for i = target} \\ D \text{ (a large maximal distance)}, & \text{otherwise} \end{cases} \quad (1)$$

where $D > (M-1) * d_{max}$

e The grid points in a dynamic system are updated in an order depending upon the distance to the target.

$$x_i = \begin{cases} 0, & \text{for i = target} \\ D, & \text{for i = obstacles} \\ \min(D, f(i,n)), & \text{otherwise} \end{cases} \quad (2)$$

where n is the time step and

$$f(i,n) = \min_{j \in B_i}(d_{ij} + x_j(n)) \quad (3)$$

B. Environment

The environment is represented as a $M \times M$ grid, each grid point records the distance to the target. The information stored at each point is a current estimate of the distance to the nearest target and neighbor from which this distance was determined. The distance estimate at each grid point are updated using the information gathered from the neighbors of the point. The dynamic programming agent only requires the distance information in the neighbor grid points. It needs no prior knowledge about the obstacle position. At each time step collision-free shortest path is generated by propagating the distance to the target and updating them in an order depending on the distance to the target for intelligent navigation.

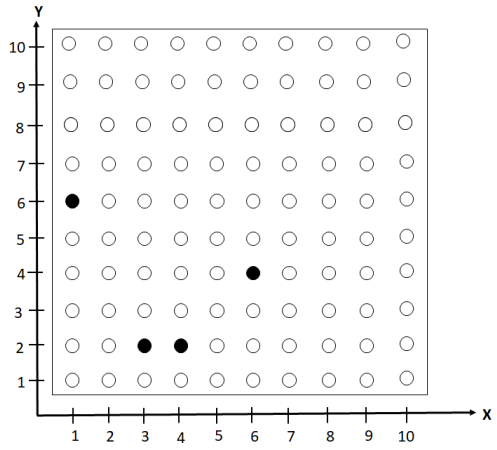

Fig. 1: Grid environment

Fig. 1 shows the sample environment of 10x10 grid map which was used for implementing DP algorithm. Each grid point, labeled by an index *i*, is either a free space or obstacle. Here, free space is defined by clear grid points and obstacle location by black grid points. The robot and target can occupy any free space. d_{min} and d_{max} are defined as the minimum and maximum distances between any two adjacent neighbors in the grid. B is defined as the set of free spaces that are adjacent neighbors to grid point *i*. The distance between any two free space *i* and *j* is the minimum Euclidean length of all paths joining *i* and *j* through non obstacle adjacent neighbors and is represented as d_{ij}.

C. Distance propagation

Each grid point *i* is associated with a real value $x_i(n)$ which records the distance to the nearest target at time step *n*. The system is initialized by setting $x_i(n)$ at all locations other than the target locations to some large distance *D* (*D* is taken as *infinite* during implementation) and $x_i(n)$ at all target locations to 0. The system then evolves by updating the grid points in an order depending on the distance to the target.

After the first time-step, target locations are zero. The adjacent neighbors of the targets record the correct distance. After the second time-step, the adjacent neighbors of the targets' neighbors record the correct distance. This goes on until the whole grid is covered. The distance to target information is thus propagated outward by one grid step at each time step. The distance value at index i is updated after the number of time-steps exceeds the number of grid steps along the minimal distance path from the index, i to the target. This value then does not change with further time steps.

D. Path selection

It is assumed that the robot has 8 action space movement as seen in Fig. 2, i.e., the robot can move move from any grid point to any neighboring grid point.

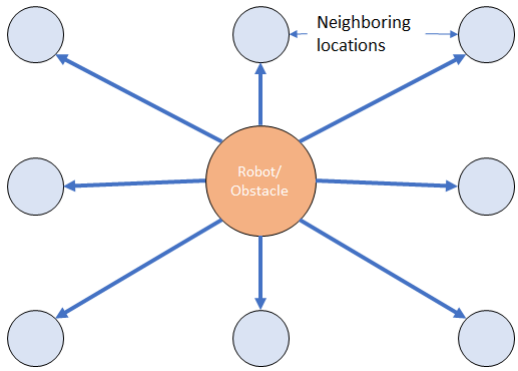

Fig. 2: Action space for robot/obstacle

The robot location, r(t), is a function of real time $t > t_0$. It is first initialized as $r(t_0) = i_0$. At time t_k, the next update time t_{k+1} ,and next update location, r(t_{k+1}), are determined using,

$$r(t_{k+1}) = Ind(r(t_{k+1}), n(r(t_k), t_k)) \quad (4)$$

where, n(i, t_k) means the highest time step n for which xi has been computed for up to time t_k, and Ind(i, n) is the index of the closest neighbor through which the value of x_i(n) was calculated. Then,

$$Ind(i,n) = \begin{cases} i, & \text{if } x_i(n)\varepsilon[0,D] \\ j\varepsilon B_i(n)|d_{ij} - min, & \text{otherwise} \end{cases} \quad (5)$$

where $B_i(n)$ is assumed as,

$$B_i(n) = j\varepsilon B_i | f(i, n-1) = (d_{ij} + x_j(n-1)) \quad (6)$$

For dynamic obstacles, the robot location can still be updated accordingly by keeping track of Ind(i,n) for robot location i, while updated x_i.

III. Results and Analysis

In this section, we present the simulation results and the issues faced during implementation of the algorithm. The figures in this section depict path planning using the algorithm in both static as well as dynamic environments. The obstacles, similar to the robot, have an action space consisting of 8 actions. They can randomly move to any of the their neighboring positions in the next time-step using these 8 actions. The following assumptions were made to successfully implement the dynamic path planning algorithm:

- The robot can move to any of its empty neighboring locations.
- The movements of obstacles are unknown to the robot.
- The obstacles do not collide at any time instance, t.

The authors of the paper [1] implemented the algorithm assuming point robot dynamics. We have expanded the application of the algorithm by employing it on rigid, non-holonomic, and differential-drive robots. While these robots present shared challenges, each of the robots has its unique problems. Moreover, these complications are incremental if we go from rigid to differential-drive robots. For instance, unlike a point robot, a rigid-robot cannot avoid an obstacle by moving through the neighboring non-obstacle point. A rigid robot occupies some space that is marked by a circular region of the robot's radius. Likewise, a non-holonomic rigid robot cannot take an instant 180-degree turn to start moving in the opposite direction, and a differential-drive robot has a turning radius.The algorithm was put to test for various cases that have been listed below:

- Path planning in a static environment where the positions of all obstacles were known.
- Path planning in a dynamic environment where obstacles could randomly move around in the available space.
- Path planning to simulate movement of a robot in both static as well as dynamic environments where other robots were randomly moving.

In contrast to the simulation of the algorithm in the paper [1], we have used OpenCV [4] and Python to provide animation of path planning in the first two test cases of static and dynamic environments. The robot position for each time step for static environment is depicted in Fig. 3, whereas, the robot position during the start, intermediate and end stages is depicted in Fig. 4. Description of all the colors used during animation is given in Table I.

TABLE I

Notation	Description
Green circle	Robot position at i^{th} time step
Red circle	Target
Black circle	Obstacles
Blue lines	Robot trajectory

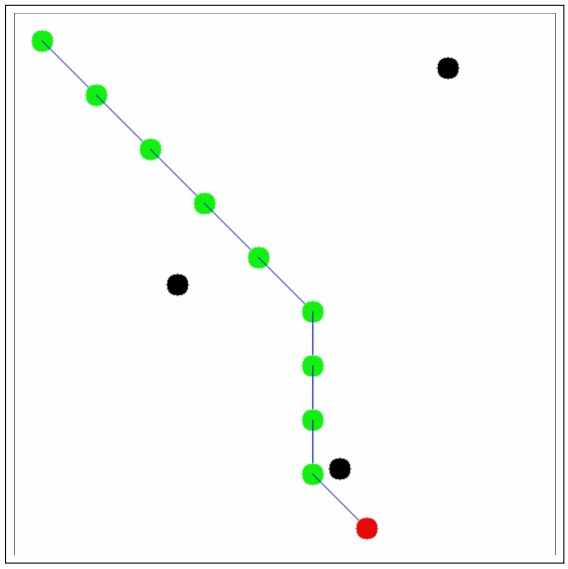

Fig. 3: Path planning in a static environment

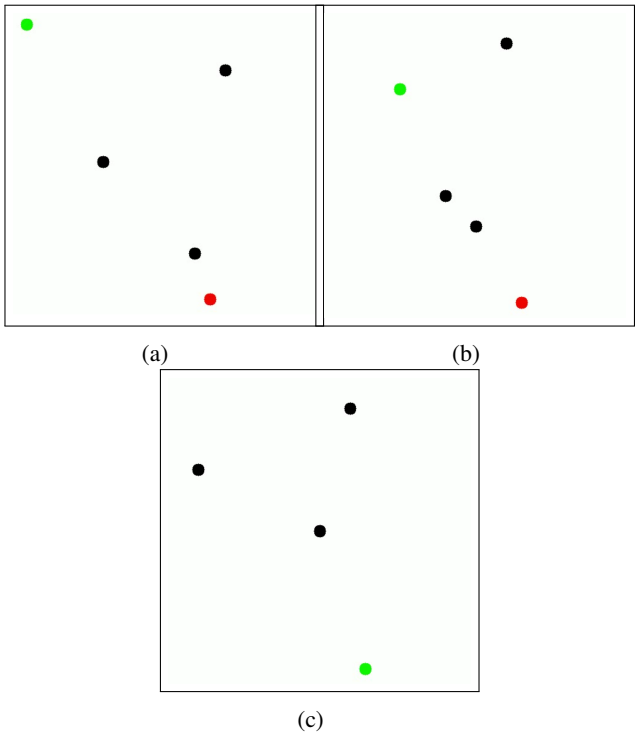

Fig. 4: The robot in a dynamic environment in (a) start, (b) intermediate and (c) end stage.

The algorithm was also deployed in a 3D environment by converting 3D world into 2D. We considered a clearance threshold of $0.25m$ between the robot and obstacles. Additionally, it is assumed that no obstacle inside the work-space does not occupy an area of more than $0.25 \times 0.25m^2$. The simulation of the movement of the robot along the path derived using our algorithm was generated using the Melodic version of the Robot Operating System (ROS) [5] and Gazebo. They can be seen in Figure 6.

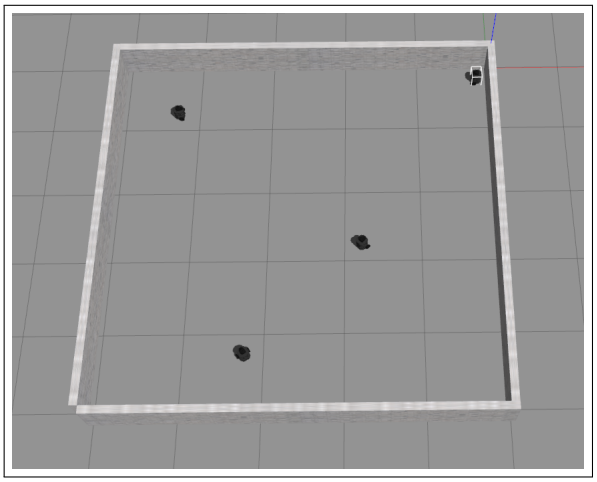

Fig. 5: Path planning in a static environment

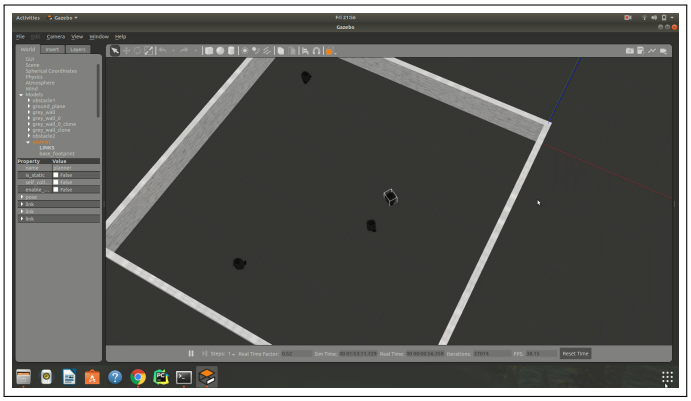

Fig. 6: Path planning in a static environment

IV. Conclusion

The DP algorithm used in the paper [1] has been successfully implemented and simulated in 2D world with both static as well as dynamically moving obstacles. The implementation for the same is based on OpenCV [4] and Python. The algorithm was also successfully deployed on a differential-drive turtlebot3 burger robot in 3D static and dynamic environment using ROS Melodic [5] and Gazebo. The results show effective path selection to reach the target from start point in real time. The algorithm can be integrated with mapping concepts, such as SLAM, to employ it on real-time systems. The results obtained are promising and have real world application. In addition to that, the integration of this algorithm with mapping data in real-time presents an interesting line of future work.

References

[1] D. Tamilselvi, P. Rajalakshmi, and S. M. Shalinie, "Dynamic programming agent for mobile robot navigation with moving obstacles," in *2009 International Conference on Intelligent Agent & Multi-Agent Systems*. IEEE, 2009, pp. 1–5.

[2] A. Vemula, "Safe and efficient navigation in dynamic environments," 2017.

[3] CMU, "Dynamic programming," https://www.cs.cmu.edu/ avrim/451f12/lectures/lect1002.pdf.

[4] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[5] Stanford Artificial Intelligence Laboratory et al., "Robotic operating system." [Online]. Available: https://www.ros.org