Motion Planning Using Maximum Clearance Roadmap in Configuration Space

1st Alexander Paskal
Electrical Computer Engineering
University of California, San Diego
La Jolla, USA
apaskal@ucsd.edu

2nd Unduwap KandageDon
Electrical Computer Engineering
University of California, San Diego
La Jolla, USA
ukandage@ucsd.edu

3rd Jianyu Tao
Electrical Computer Engineering
University of California, San Diego
La Jolla, USA
jit124@ucsd.edu

***Abstract*—Robots that move in environments which contains obstacles require more sophisticated motion planning algorithms to produce trajectories that are safe and efficient. Many popular motion-planning algorithms are forced to trade off either path quality for computational tractability, with no consideration for path safety or repeat planning operations. In this work, we present an approach for constructing a maximum clearance roadmap from a configuration space, and a method for converting that roadmap into a sparse bidirectional graph that can be used for efficiently computing paths in difficult environments. We show paths planned by using A* this sparse graph representation are able to find small passage ways in environments with many obstacles, and do so much faster and in a safer manner than existing sampling-based methods.**

***Index Terms*—robotics, motion planing, optimal control, maximum clearance**

I. Introduction

Motion Planning is a fundamental task for robots. It aims to find a collision-free trajectory from a known starting position and goal position in any environment which contains various obstacles. For a complex robot which is not simply a point in the environment, motion planning is often tackled in configuration space (C-Space), where every point in the C-Space represents a possible robot configuration. Paths produced by a motion-planning algorithm should ideally be optimal or near-optimal (shortest paths), should be safe and should be smooth. Producing paths that satisfy all of these characteristics can be difficult in complicated environments.

In this context, many search-based algorithms first discretize the continuous C-Space to a grid and compute the optimal trajectory. Dijkstra's algorithm [1] can find the optimal trajectory by traversing the grid in all directions from the starting position. Dijkstra's algorithm is not computationally efficient in a grid with high resolution or dimension since it will traverse all positions in a grid and find the nearest path to all intermediate positions. In order to tackle this problem, A* algorithm [2] adds a heuristics function to bias the direction that points to the goal position, so the process will not traverse the whole grid to find the optimal trajectory. However, despite the improvements offered by A*, search-based planning suffers from the curse of dimensionality and quickly becomes intractable in spaces with high resolution or dimensionality.

An alternative approach is represented by sampling-based algorithms. A staple approach in sampling-based planning is the Rapidly=exploring Random Tree(RRT) [3] algorithm, which grows trees from the starting position randomly and gradually cover the whole space. It is efficient in high resolution space but the result is not smooth and sometimes the trajectory is sub-optimal. Its upgrade RRT* [4] adds a rewiring process to make the path more optimal and smooth. These sampling-based algorithms are able to compute plausible paths much more quickly in higher dimensions but often miss narrow passageways, since they rely on probabilistically sampling a large space. As a result, pathways produced by these sampling-based methods are often suboptimal.

We propose a motion-planning algorithm that performs as efficiently (or moreso) than sampling-based approaches while also being able to identify narrow passageways. We reconstruct the signed distance field of a configuration space and use this information to compute a dense graph that traverses the skeleton of the space. We then transform this dense graph into a sparse weighted graph of intersections and use this graph for A*. We show that with the signed distance field information, we are able to arbitrarily connect any point in the free space to our graph using hill-climbing, and that paths produced using this represetnation are smoother and shorter than those produced by generated probablistic roadmaps. We also show that by construction, paths produced are inherently safe and maximize the space between robots and obstacles.

II. Method

A. Problem Formulation

We formulate the task using a two-revolutional-link robot, and each link has an angle between $-\pi$ and π. The goal is a 2D location $[x, y]$ where the end-effector is expected to reach in the workspace. The objective is to find a trajectory for the robot end-effector from the starting position to the goal position. The motion planing is often done in the configuration space, so the objective becomes to find a robot configuration sequence$\{[q_1^0, q_2^0], [q_1^1, q_2^1], ..., [q_1^n, q_2^n]\}$, where each entry represents the robot configuration of each joint, q_1 and q_2, at time n. In our problem setting, the robot will have a 2D configuration space.

B. Discretization of Configuration Space

We sample the configuration uniformly along a discrete grid. In our experiment, we sample the configuration every single degree of each joint movement. We use robot forward kinematics to test the collision with robot and the environment. The resulting configuration space will be a binary 2D matrix in our setting.

C. Skeletonization and Signed Distance Field

After we compute the discrete configuration space, we reconstruct the binary image skeleton. Skeletonization is an image processing technique which reduces binary objects to 1 pixel wide representations with connectivity, where every pixel is as far away from the edge as possible. We decided to use T.Y. Zhang's algorithm for thinning the digital patterns [5]. The main idea is to iteratively remove all the contour points of the picture except those points that belong to the skeleton in parallel for all the points. To be specific, for a point P_1 and its 8 neighbor $[P_2, P_3, ..., P_9]$, which is defined clock-wisely from the north of P_1, shown in Fig. 1.

P9	P2	P3
P8	P1	P4
P7	P6	P5

Fig. 1: Example of a 2D configuration space.

In order to maintain the connectivity, there are two subiteration in one interation. In the first subiteration, P_1 is deleted when satisfies all the following conditions:

$$\begin{aligned} &(a)\ 2 \leq B(P_1) \leq 6 \\ &(b)\ A(P_1) = 1 \\ &(c)\ P_2 * P_4 * P_6 = 0 \\ &(d)\ P_4 * P_6 * P_8 = 0 \end{aligned} \tag{1}$$

where $A(P_1)$ is the number of 01 patterns in the ordered set $P_2, P_3, ..., P_9$ and $B(P_1)$ is the number of nonzero neighbors of P_1. In the second iteration, the condition $(c), (d)$ above become:

$$\begin{aligned} &(c')\ P_2 * P_4 * P_8 = 0 \\ &(d')\ P_2 * P_6 * P_8 = 0 \end{aligned} \tag{2}$$

Iteratively perform the two steps described above and stop when there is no point is deleted. The resulting set of points will be the skeletonization of the configuration space.

We implement Zhang's algorithm based on Sklearn's implementation in our experiment [6]. For the orginal 2D discrete configuration space in Fig. 2, the result for skeletonization algorithm will be Fig. 3.

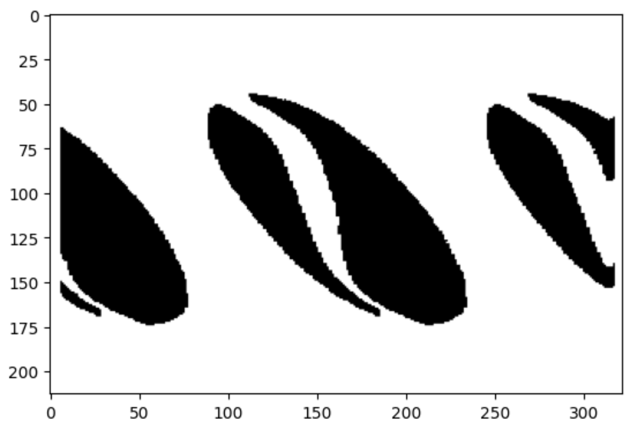

Fig. 2: Example of a 2D configuration space.

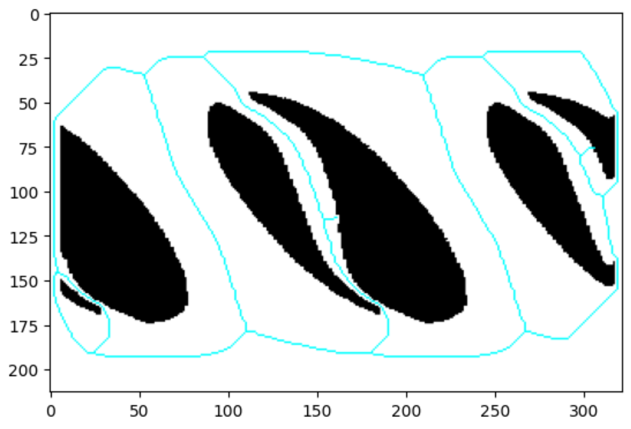

Fig. 3: Example of a skeletonization on a configuration space.

We also need a signed distance field from the configuration space to connect the start position and goal position to the skeletonization image efficiently by gradiant ascending. We modify the method proposed by Zhang to output both skeletonization and the signed distance field. Basically, start a distance variable with 0 and in every iteration of Zhang's algorithm, replace the removed pixel with current distance value and then increment the distance by 1. The result matrix will be the estimated signed distance field in configuration space, as shown in Fig. 4.

D. Detecting Intersection Node

We perform the line junction detection on the skeletonization of configuration space to find the pivot configurations. The pivot configuration is the point where two or more lines intersect in the skeletonization. The pivot point will be used to contruct a sparse graph for trajectory searching. Usually the number of pivot points will be much less than the points in skeletonization, so doing motion planing on graph consists of pivot point will drastically reduce the computation complexity.

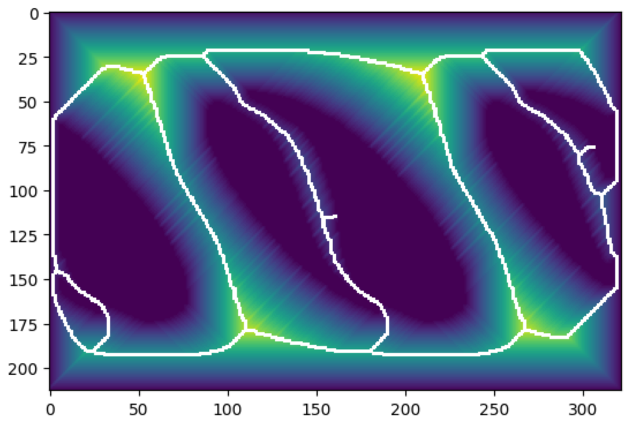
Fig. 4: Example of a signed distance field on a configuration space.

We detect the line intersection by the number of adjacent neighbors of the node. If the there are more than 2 neighbors, then the current node will be labeled as the intersection node. The intersection point (marked red) for previous example is shown in Fig. 5. Additionally, we also included the termination nodes that lead to an end within the graph. The start and goal nodes when added to these graph, would also be considered as termination-intersection nodes.

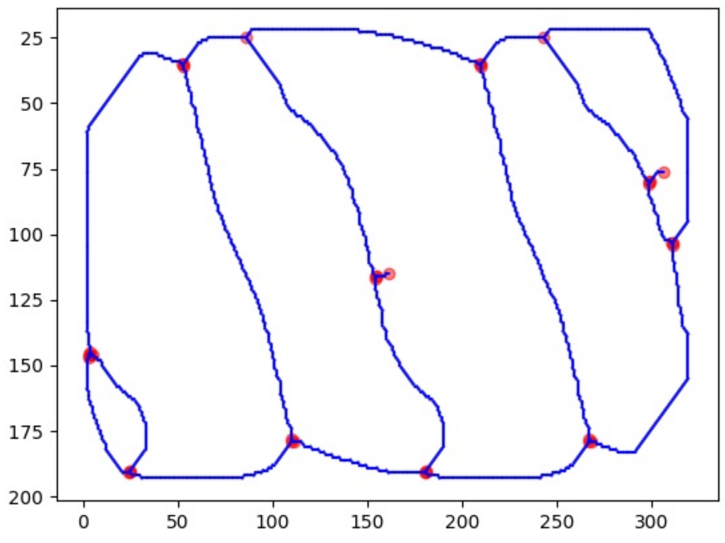
Fig. 5: The line intersection result, the red point denotes the intersection position, i.e. pivot point.

E. Connecting Nodes to Construct Dense Graph

We start with the assumption that any node to be added to a graph is in the free space. Since our dense graph corresponds to the cut locus of the signed distance field of the free space, we are able to navigate from a query node to the dense graph by simple hill-climbing, where we estimate the local gradient of the signed distance function by looking at all neighboring points and move in the direction of maximal ascent. Every newly queried point is then connected bidirectionally with the point that proceeded it. Once a point in the dense graph is reached and the connection is added, the process terminates.

F. Dense Graph to Sparse Graph

Each intersection point was added to a new sparse graph with an edge leading to each the next connected intersection. The connecting intersections were founding using a Breadth First Search (BFS) approach, iterating on every intersection node. Additionally, the first connector node was also recorded in a separate dictionary to help with mapping back to the dense graph later. The first connector, as show in Fig. 6, node is defined as the first node in the path, after the starting intersection, between two intersection points. For example, the connector point for the path between intersection A and B, is C_{ab} Following a forward path from the first connector will lead to the next intersection.

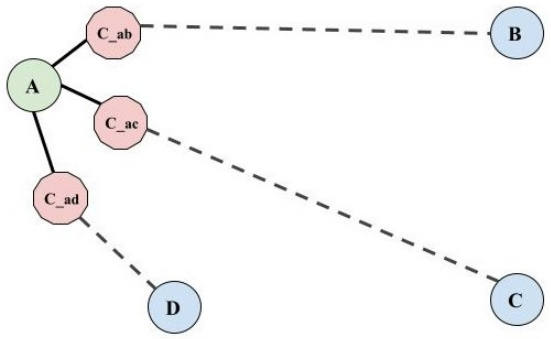

Fig. 6: This figure shows, intersection points (A, B, C, D) and their corresponding connector points, which are shown in red.

G. Motion Planning by Graph Search

After constructing the sparse graph, we will use A* algorithm to find the nearest path from the start position to goal position [2]. The A* algorithm is a search algorithm which finds the shortest path between two nodes. It is considered as an extension of the Dijkstra algorithm [1], but tries to improve the runtime by using a heuristic to find the optimal solution. To decide which path to take, the algorithm will compute cost by the path weight and the heuristic function. For a node with configuration $[q_1, q_2]$ and goal configuration $[q_1^{goal}, q_2^{goal}]$, the algorithm will compute the cost by:

$$f'(q_1, q_2) = g(q_1, q_2) + h'(q_1, q_2) \tag{3}$$

Where:

$$h'(q_1, q_2) = \sqrt{(q_1 - q_1^{goal})^2 + (q_2 - q_2^{goal})^2} \tag{4}$$

and $g(q_1, q_2)$ is the weight of the path that is precomputed when constructing the graph.

III. Experiment

We compare our proposed method with PRM for graph construction time, motion planing time and the total cost for the resulting path. We choose two separate testing case, one is the normal random picked start and end position, while the other's end position is in a narrow passage in configuration space. In Table. I, the motion planing time of our proposed method is shorter than both PRM. This is because we are searching on a sparse graph. Our method is also most efficient when constructing graph, since skeletonization and signed distance field reduce lots of computational complexity from finding nearest neighbors. The resulting paths for this test case are shown in Fig. 10, and Fig. 9. In Table. II, the end position is in a narrow passage, this scenario poses a great challenge for the randomly sampled algorithm like PRM, it generally need to create lots of tree nodes in order to find the path from start to end, so the graph constructing time is higher. Since the number of tree nodes are denser for PRM, the graph search algorithm to find the path will be slower. In comparison, our method remain unchanged when facing this issue, leading to more robust and stable computation. The resulting paths are shown in Fig. 7 and Fig. 8.

Method	Graph Const Time (s)	Planning Time (s)	Total Cost
PRM	18.5475	3.1750	437.8835
Ours	0.0646	0.0428	232.3208

TABLE I: The performance comparison for the baseline algorithm and proposed method at random picked starting and goal position

IV. Experiment

Method	Graph Const Time (s)	Planning Time (s)	Total Cost
PRM	103.9323	3.3023	359.6009
Ours	0.0638	0.0414	227.8233

TABLE II: The performance comparison for the baseline algorithm and proposed method when goal position is in narrow passage

V. Conclusion and Acknowledgement

In conclusion, our proposed method performed better to PRM and RRT* in terms of both pre-processing time and the path planning time in the simulations we tested. However, there is still some limitation of our method present in some environments. For example, if an environment has a large width and length, the sampling resolution we have may not cover the narrow passages. Additionally, if an environment contains fewer or no obstacles, the collision configuration space will be small, which makes the skeletonization deviate far away from the optimal path.

Our algorithm can be further modified to be optimal for a larger variety of maps in future works. Currently, it's efficiency comes from being able to generate road maps that can be reused in future path planning. These road maps are formed using intersection points which help to create extremely sparse graphs when compared to the whole environment. This initial sparse graph construction is what helps our algorithm stay efficient for every new iteration of path planning.

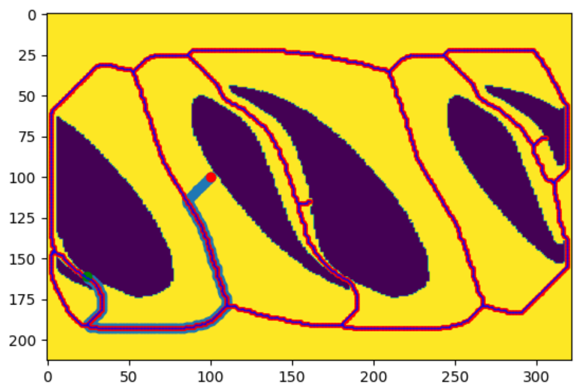

Fig. 7: Our path result when goal position is in a narrow passage (Start = red dot; Goal = green dot)

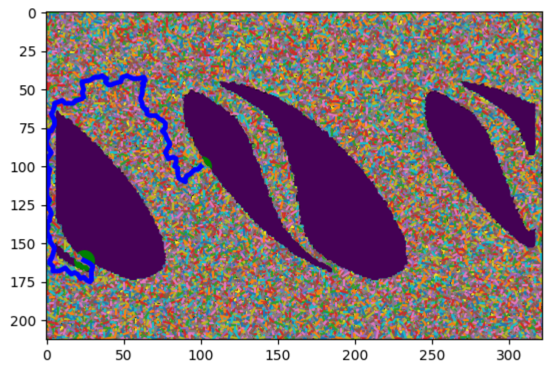

Fig. 8: PRM path result when goal position is in a narrow passage (Start & Goal = green dot)

VI. Supplemental Material

All programming code and other additional material used for the simulations within this report can be found on: https://github.com/alexander-paskal/RoadmapPlanner.git

References

[1] E. W. Dijkstra, "A note on two problems in connexion with graphs," Numerische Mathematik, vol. 1, p. 269–271, 1959.

[2] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," IEEE transactions on Systems Science and Cybernetics, vol. 4, no. 2, pp. 100–107, 1968.

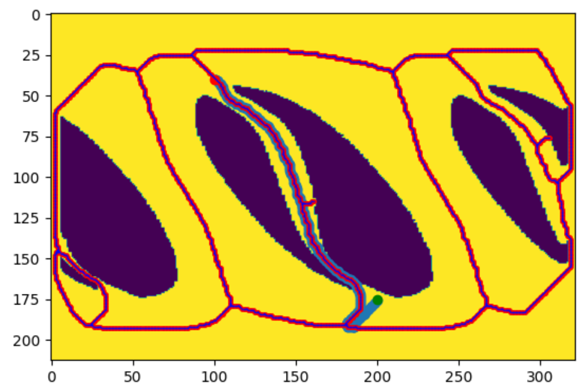

Fig. 9: Our result of finding path when goal is in open space (Start = red dot; Goal = green dot)

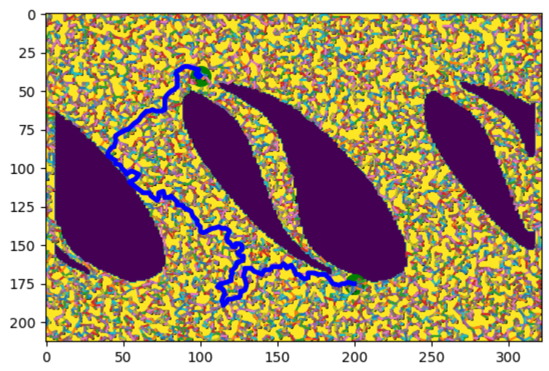

Fig. 10: PRM result of finding path when goal is in open space (Start & Goal = green dot)

[3] S. M. LaValle et al., “Rapidly-exploring random trees: A new tool for path planning,” 1998, [online] Available: https://www.cs.csustan.edu/xliang/Courses/CS4710-21S/Papers/06

[4] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, “Anytime motion planning using the RRT,” in IEEE International Conference on Robotics and Automation, 2011, pp. 1478–1483.

[5] Zhang, Tongjie Y., and Ching Y. Suen. ”A fast parallel algorithm for thinning digital patterns.” Communications of the ACM 27.3 (1984): 236-239.

[6] Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu, and the scikit-image contributors. scikit-image: Image processing in Python. PeerJ 2:e453 (2014) https://doi.org/10.7717/peerj.453

[7] “ImageMagick V6 Examples – Morphology of Shapes.” Morphology of Shapes – IM v6 Examples, legacy.imagemagick.org/Usage/morphology/. Accessed 9 June 2023.