Motion Planning Using Maximum Clearance Roadmap in Configuration

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 the environment which contains obstacles require a more sophisticated motion planning algorithm to produce a safer, faster trajectory. Many popular motion planing algorithm such as RRT and RRT* require recomputing and re-growing the tree from every starting points, and A* algorithm is computationally-inefficient in constrained environment which has continuous or high resolution configuration space and can also result in sub-optimal trajectory. Our work utilizes the robot configuration space and pre-compute a Maximum Clearance Roadmap – a map that contains paths that is as far away from the surrounding obstacles as possible. With this map, it no longer needs to recompute the map/tree from every novel starting position and it is more practical for real-time tasks.**

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

I. Introduction

Motion Planning is a fundamental task for robots. It aims to find a best 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 C-Space represents a possible robot configuration (the angle between two links, for example). Motion planing requires to find an optimal path from starting configuration to goal configuration without passing through configurations that are impossible to reach.

In this context, many algorithm first discretize the continuous C-Space to a grid and compute the optimal trajectory. The Dijkstra algorithm [1] can find the optimal trajectory by traversing the grid in all directions from the starting position. The Dijkstra 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] or all its variants, 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, A* algorithm suffers in a large, high resolution C-Space and can also results in sub-optimal solution when a narrow passage exists in C-Space and is neglected by low resolution discretization.

In this way, sampling-based algorithm begins to prevail in motion planing of large C-Space. The Rapidly exploring Random Tree(RRT) [3] grows tree 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. The sampling-based algorithm also inefficient for a C-space that contains narrow passage.

Our method is to precompute a Maximum Clearance Roadmap in C-Space, this road map will contain all the possible paths that have maximum distance from the surrounding obstacles, including the path in narrow passage. Therefore, the searching space will reduce to a extremely low resolution roadmap, the proposed algorithm will be both time efficient and computationally efficient.

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 3D location $[x, y, z]$ where the end-effector is expected to reach. 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 by sampling 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, with 1 denotes for no collision and 0 denotes for collision.

C. Skeletonization and Signed Distance Field

After we get the discrete configuration space, we will do skeletonization of the pattern in it. Skeletonization is an image processing technic which reduces binary objects to 1 pixel wide representations with connectivity, where every pixel is as far away from the edge as possible. It is also called maximum clearance roadmap in robotics. 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.

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

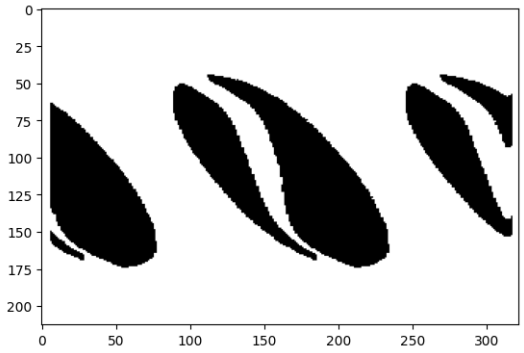

Fig. 2: Example of a 2D configuration space.

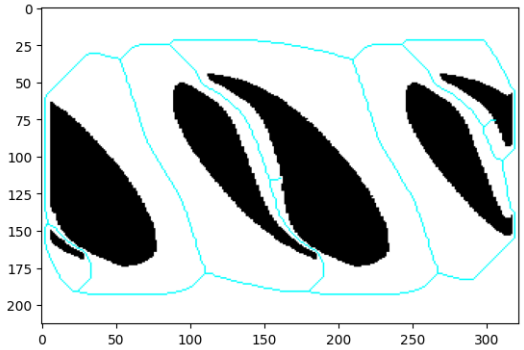

Fig. 3: Example of a skeletonization on a configuration space.

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.

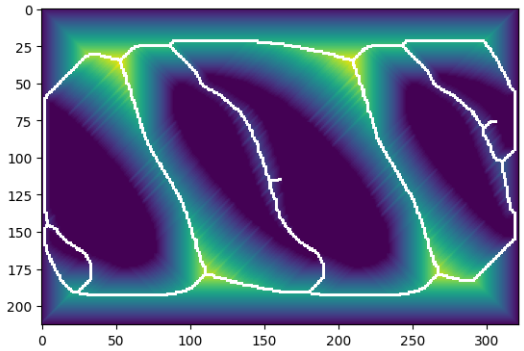

Fig. 4: Example of a signed distance field on a configuration space.

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. 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.

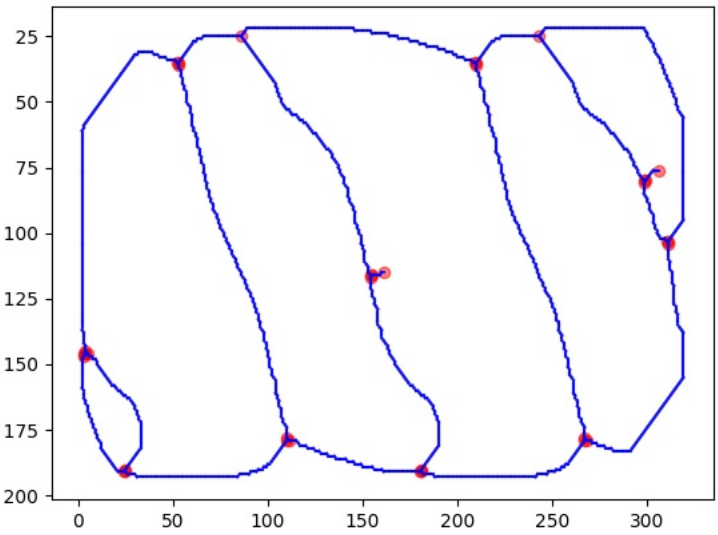

Fig. 5: The line intersection result, the red point denotes the intersection position, i.e. pivot point.

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 reach 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.

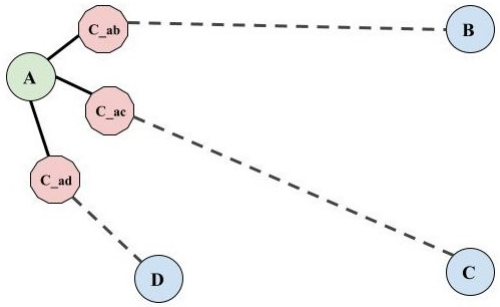

Fig. 6: This figure shows, intersection points (A, B, C, D) and their corresponding connector points, which are shown in red.

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

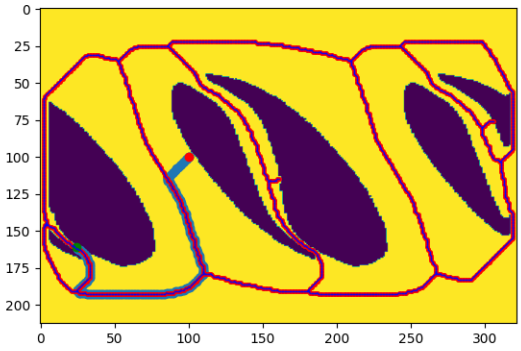

Fig. 7: Our path result when goal position is in a narrow passage (Start = red dot; Goal = green dot)

V. Conclusion and Acknowledgement

In conclusion, our proposed method is superior to PRM and RRT* in terms of both preprocessing time and the path planning time. However, there is still limitation of our method in some environment. For example, if an environment with large width and length, the sampling resolution we have may not cover the narrow passages. What is more, if a environment contains fewer or no obstacles, the collision configuration space will be small, which makes the skeletonization deviate far away from the optimal path.

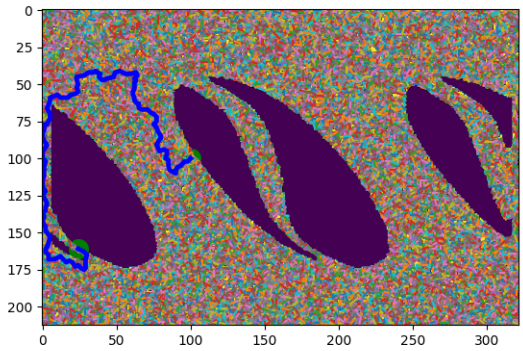

Fig. 8: PRM path result when goal position is in a narrow passage (Start & Goal = green dot)

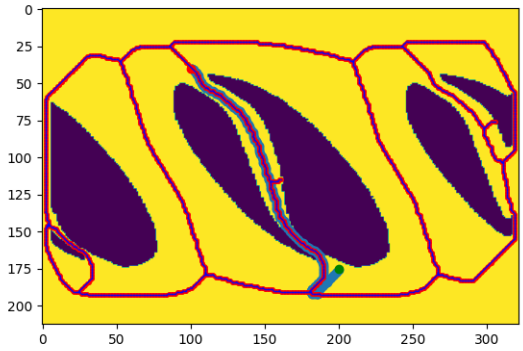

Fig. 9: Our result of finding path when goal is in open space (Start = red dot; Goal = green dot)

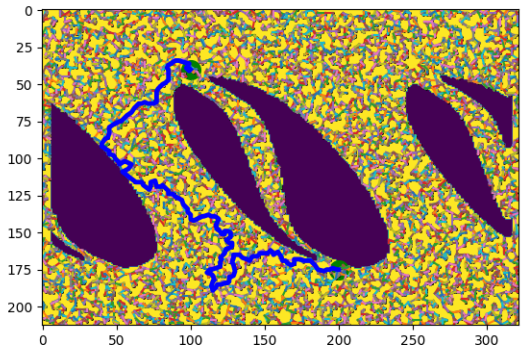

Fig. 10: PRM result of finding path when goal is in open space (Start & Goal = green dot)

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.

[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.