2
Classic Motion Planning Methods

In this chapter, we study three different classic motion planning methods. They may not be used directly in recent applications, but some components of these approaches are still used in recent techniques.

2.1 *Roadmaps*

As its name tells us, the roadmap represents connectivity of the free space based on simple structures like graphs. We commonly use such roadmaps (e.g., tourist map and metro map), since they give how we can move around space. The visibility graph mentioned in Ch. 1.2 is an example of roadmap techniques. In this section, we discuss another type of roadmap, Voronoi diagram.

At a high level, the Voronoi diagram can generate paths that maximize the distance from obstacles. This property is very useful, especially when hitting or becoming close to some obstacles poses critical problems. Inputs to Voronoi computation are called Voronoi sites, which can be points, lines, polygons, etc. In a simple setting, input to Voronoi diagram is points. In this case, each Voronoi region associated with each input point, p, includes a set of points which is closer to the input point p than the rest of other points (Fig. 2.1).

In a simple setting in a 2D case, it has been shown that the optimal path in terms of path lengths can be deformed from one of paths on Voronoi diagram [1]; we call such paths that can be deformed into each other to be in the same homotopy (Ch. **??** YOON: fix). While it is unclear for higher dimensional cases, having such information could be a reasonable choice for understanding the connectivity of the free space. Fig. 2.2 shows an example path computed from the Voronoi diagram in a 2D case.

Unfortunately, computing such Voronoi diagrams with general Voronoi sites such as polygons and point clouds can be very time consuming and requires indepth understanding on geometric computing.

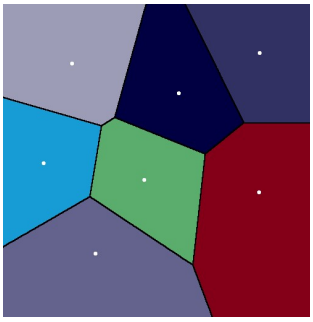

Figure 2.1: This shows the Voronoi diagram with black lines with Voronoi regions shaded in different colors given input white points.

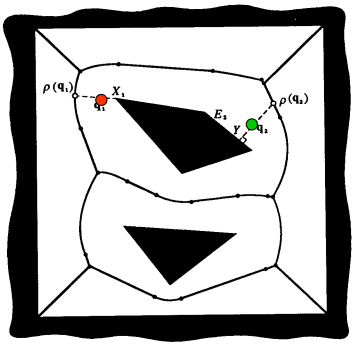

Figure 2.2: This visualizes Voronoi diagram computed from obstacles shown in dark regions.

[1] H. Choset and J. Burdick. Sensor based planning: The hierarchical generalized voronoi graph. *Workshop on Algorithmic Foundations of Robotics*, 1996

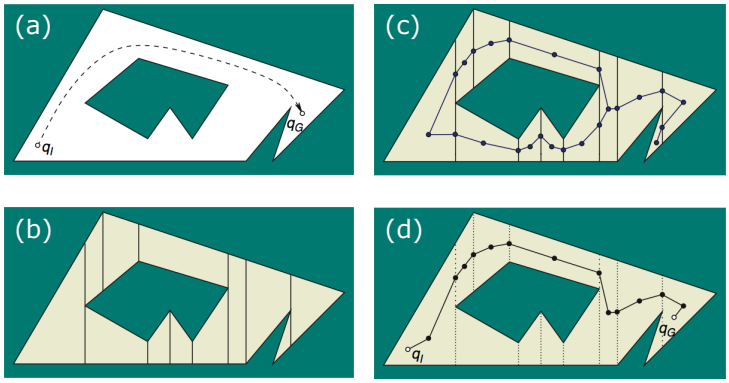

Figure 2.3: This figure visualizes trapezoidal decomposition given a space (b), computing an adjacency graph (c) and computing a collision-free path (d) given the input start and goal positions (a). This figure is from slides of Arras.

Nonetheless, we can utilize some of these ideas for various analysis and useful cues for identifying optimal paths.

While we discussed roadmap techniques mainly with the Voronoi diagram in this section, any techniques representing the connectivity of the free space can be considered as roadmap techniques. These roadmap techniques are useful when obstacles do not change frequently, and one can see how we can move around by looking at the roadmap.

2.2 *Cell Decomposition*

Cell decomposition aims to partition the C-free space into a set of simple, non-overlapping primitives (e.g., boxes) so that we can have connectivity information on those primitives for path planning. Depending on characteristics of various cell decomposition, they can be classified by exact or approximate methods.

The exact cell decomposition aims to partition the space into a set of primitives whose union is equal to the C-free space. On the other hand, approximate techniques aim to compute a set of primitives whose union is a subset of the C-free space.

Trapezoidal decomposition is one of exact cell decomposition methods, and partitions the free space into a set of non-overlapping trapezoids.

An example of the exact cell decomposition is trapezoidal decomposition, which uses trapezoids as non-overlapping primitives. Fig. 2.3 shows an example of computing trapezoidal decomposition. For encoding the connectivity information, we construct an adjacency graph where each trapezoid is a vertex, and an edge is created between two vertices when they are adjacent to each other. Given the start and goal positions, we can compute collision-free paths by searching through the adjacency graph.

In a 2D case, one can notice that trapezoidal decomposition occurs at vertices of input polygons. Based on this observation, one can sort vertices of input polygons and construct trapezoid as necessary. This simple approach can result in $O(n \log n)$ time complexity in the 2D

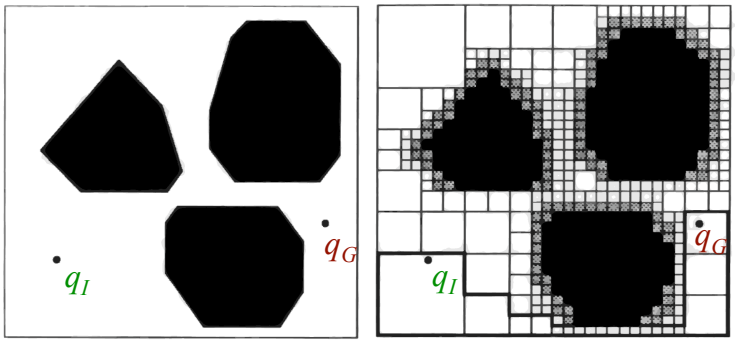

Figure 2.4: This shows an approximate cell decomposition method using a quad-tree in the 2D space. We decompose the space until we found a sequence of free cells, where we can find a collision-free path. This figure is from slides of Arras.

case. Nonetheless, implementing this idea for satisfying the definition of the exact cell decomposition is very tricky, and can have a much higher time complexity in higher dimensions.

Contrary to the exact methods, approximate methods are more commonly used thanks to its simplicity. The main issue of these approximate techniques is that there is potential that we cannot find a path, even though there is a solution. Common approximate methods use a quad tree for 2D space, and octree for 3D space. Fig. 2.4 shows an example of using the quad-tree that partitions the space into cells, whose types are free, mixed, or obstacle; we do not have any obstacles in the free cell, and the obstacle cell is contained in the obstacle. In the mixed cells, some portions of them are free or not.

Using an approximate cell decomposition method, there is potential that we miss to find an actual solution due to the nature of approximation.

Given an initial decomposition where we can find an initial sequence of cells with free or mixed status, we iteratively refine the mixed cells by partitioning them into sub-cells. We repeat this process until we find a sequence of cells only with the free status. While this approach is relatively simpler to understand and implement compared to the exact approach, there is potential that we miss to find a path due to the nature of approximation, while there is a solution actually.

2.3 *Potential Field*

Potential field is one of classical approaches of path planning supporting collision avoidance and simple integration with a low-level controller. Its main idea is to have attractive forces to the goal, while generating repulsive forces from obstacles. Fig. 2.5 illustrates various fields generating these attractive and repulsive forces.

Overall, in the potential field, we navigate our robot from a place to another place with a lower potential. For generating the attractive forces to the goal, we construct an attractive field that has the highest potential value at the start and the lowest value at the goal. In a

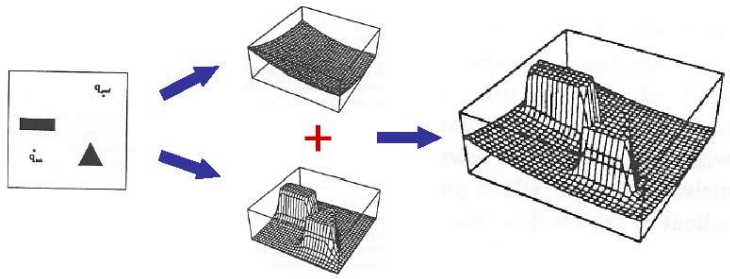

Figure 2.5: This figure shows a potential field (right side) that combines attractive forces (the top field) to the goal and repulsive forces (the bottom field) from obstacles.

similar manner, we generate a repulsive field where we have higher potential values at obstacles that fall down as a cell of the field gets away from those obstacles. The final potential field is simply the sum of those attractive and repulsive fields.

The next question is how to compute such attractive and repulsive forces from the constructed potential field. Since our goal is to direct our robot to a place with a lower potential field, we can use the negated gradient of the field at the current position. By following the direction with the negated gradient of the field, the robot can move to a place with a lower potential field. One of desirable properties of this potential field is that the computed direction and its magnitude can be used directly for a lower-level controller that drives actuators (e.g., motors).

To direct our robot to a place with a lower potential field, we use the negated gradient of the field at the current position.

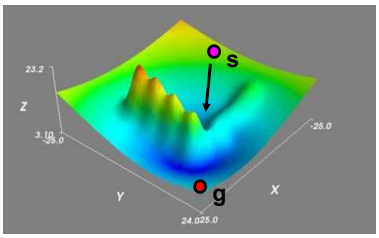

Figure 2.6: Simply following forces generated from the potential field can result in being stuck in a local minimum.

As the downside of the potential field, it can be stuck in a local minimum, failing to reach the goal position. As shown in Fig. 2.6, simply following the negated gradients leads our robot to a point behind the wall and cannot find a path that circumvents the wall. This kind of local minima is very common problems in many optimization problems including the path planning under the study with the potential field. Addressing this kind of problems is not straightforward, unless we have relaxation step or a form of exploration step (Ch. 1.3) that attempts to find a better path.

2.4 *Completeness*

So far we have discussed different classical approaches of computing collision-free paths in simple settings. Depending on cases, some of algorithms can identify multiple paths. In some other cases, algorithms may not even find a single path. In this case having no paths, what conclusion can we extract? Does it mean that there is absolutely no collision-free path or that the employed algorithm is insufficient to find one, even though there is actually a path? Since we have ambiguous outcomes in this case, it will be great if we can resolve such ambiguous cases.

Along this line, we can think of a completeness as a desirable

property for a motion planner. We call a planner complete when it surely identifies a path if there exists, it indicates no solutions otherwise. In this perspective, the visibility graph in the 2D space with a point robot is complete. On the other hand, the potential field is not complete, since it may fail to identify one due to its inability getting out of local minima. In Ch. 7.1, we discuss probabilistic completeness of well-known sampling based planners.